Design, analysis and implementation of cryptographic symmetric encryption algorithms on FPGA

Loïc Besson

université
PARIS-SACLAY

Conception, analyse et implémentation d'algorithmes de chiffrement symétrique sur FPGA

Design, analysis and implementation of cryptographic symmetric encryption algorithms on FPGA

Thèse de doctorat de l'Université Paris-Saclay

École doctorale n° 580,
Sciences et Technologies de l'Information et de la Communication (STIC)
Spécialité de doctorat: Mathématiques et informatique
Graduate School : Informatique et science du numérique.
Référent: : Université de Versailles Saint-Quentin-en-Yvelines

Thèse préparée dans l'unité de recherche **Laboratoire de Mathématiques de Versailles (LMV)**, sous la direction de **Louis GOUBIN**, professeur à l'Université de Versailles-St-Quentin-en-Yvelines, la co-direction de **Jacques PATARIN**, professeur à l'Université de Versailles-St-Quentin-en-Yvelines et la co-supervision de **Patrice SAUSSAY**, Product Manager Crypto, HENSOLDT France SAS

Thèse présentée et soutenue à Versailles, le 8 décembre 2021, par

Loic BESSON

Composition du jury:

Marine MINIER Professeure, Université de Lorraine	Présidente
Christophe CLAVIER Professeur des Universités, Université de Limoges	Rapporteur et examinateur
Viktor FISCHER Professeur, Université de Saint-Etienne	Rapporteur et examinateur
Jean-Baptiste RIGAUD Professeur, Ecole Nationale Supérieure des Mines de Saint-Etienne	Examinateur
Louis GOUBIN Professeur, Université de Versailles-St-Quentin-en-Yvelines	Directeur
Jacques PATARIN Professeur, Université de Versailles-St-Quentin-en-Yvelines	Co-directeur

THESE DE DOCTORAT

NNT : 2021UPASG104

université PARIS-SACLAY

ÉCOLE DOCTORALE
Sciences et technologies de l'information et de la communication (STIC)

Titre: Conception, analyse et implémentation d'algorithmes de chiffrement symétrique sur FPGA
Mots clés: Cryptographie légère, Chiffrement par blocs, Fonctions éponges

Résumé: Cette thèse explore différents aspects de construction d'algorithmes de chiffrement symétrique. Les travaux portent sur le design et l'implémentation d'algorithmes de chiffrement par blocs dits légers, ainsi que sur les fonctions éponges permettant de réaliser du chiffrement authentifié. Le but recherché dans les deux notions est de définir des solutions permettant de garantir des bornes de sécurité similaires à celles des algorithmes standards de la littérature cryptographique tout en obtenant des performances et un ratio débit sur surface utilisée le meilleur possible. La première partie étudie les algorithmes de chiffrement par blocs légers et les différentes techniques existantes pour développer un nouvel algorithme avec les propriétés souhaitées. Nous définissons également un nouveau mode d'opération permettant de garantir une sécurité équivalente à celle des modes d'opération standardisés par le NIST ou l'ANSSI tout en offrant la possibilité d'une application n'échangeant pas de vecteur d'initialisation. Pour finir, après une comparaison des différents modes d'opération ainsi que les permutations existantes dans la littérature, le but est de définir les meilleurs candidats possibles selon le cas d'usage.

Title: Design, analysis and implementation of symmetric encryption algorithms on FPGA
Keywords: Lightweight cryptography, Block cipher, Sponge Function

Abstract: This work studies several aspects of design and implementation of symmetric cryptography. The focus was brought on two different kinds of construction, namely lightweight block ciphers and sponge functions providing authenticated encryption. For both the goal is to define solutions ensuring similar security bounds as standards algorithms while achieving good performances towards throughput and low area occupation. The first part of this thesis focuses on the state-of-the art in designing block ciphers and which parameters and construction may lead to the desired performances. We then define a new mode of operation achieving the same security margins as the mode of operation standardized by the NIST and the ANSSI while allowing application where the initialization vector cannot be sent to both correspondents. The second half is based on the study of sponge functions, from the SHA-3 competition to the NIST LWC standardization process, of both mode of operation and permutation to achieve the best performances as possible for different use cases.

Acknowledgments:
First of all, I would like to thank Julien Francq, without whom I would have never dared going for this thesis, that Nancy's beer really was what made me do it, either the alcohol of the euphoria of seeing a project like that come to an end.
Then I am very grateful for Louis Goubin and Jacques Patarin for their help and support during those three and a half year of research, I hope that we can continue working together in the future.
Moreover, I would like to thanks Viktor Fischer and Christophe Clavier for agreeing on being rapporteurs of my thesis and their precious comments that helped make this work more valuable.
Marine Minier for agreeing on being part on my jury. And most of all, thanks you and Julien, for including me and making me contribute to the Stanislas team, this first step in the cryptographic world has really been life changing for me.
Jean-Baptiste Rigaud for also accepting to be part of my jury and for the help and support just before the beginning of this thesis, in my decision of going for it and on VHDL and FPGA implementations.
Thanks to Patrice Saussay, for supporting me all along, for letting a youngster teach you cryptography, without judging him on its age. Thanks also for all the backup at HENSOLDT, the evening spent on texts correction, it's really been an adventure and it is not ending soon I hope!
Of course, I can't speak about HENSOLDT without quoting Helene the Pink Panther, best of chiefs and presenters, trusting my choices and entrusting me with my own decisions. Thank you also to all coffee teammates, for all the laughter, coping with all the irrelevant fun facts on animal sex lives at 8 am. Don't worry I'll go back to pastry cooking at some point, and not just make you drool over pictures on Instagram. Particular thanks to Paul for trying to make me lost the overweight, hang on there mate, we still have a lot to do!
A big shout out to my friends who are still here even without any living sign from me for months, always cheering me up and encouraging at the best of times. Hopefully we may all go back to concerts, headbangs, eat waffles or sushis or even bungee jump in Greece. Jordi, don't worry, I'll have some free time to crush your birds and dragons some more. To all my online fishermen, I don't promise I'll come back to the battle with you, be the time spent trying to master the trident surely was a necessary time spent to not go mad during 2020. Thanks you also for all the frightening hours spent to try and save the world from Nyarlathotep and the vampires, I died a few time but that was a lot of fun!
To my family of course for letting me be a student for three more years, don't worry, it's finally over now, you won't have to worry too much anymore.
And finally to my favorite dance teacher, you've made it until here, congrats, you can now enjoy a brand new, stress-less boy for all of the fore-planned adventures that await us.

Contents

List of Figures

List of Tables

Part I

Introduction

Chapter 1

Context of this thesis

I have been able to conduct those three years of research in collaboration between HENSOLDT France and the Laboratoire de Mathématiques de Versailles, in the Université Versailles-Saint-Quentin.

A quick introduction into what HENSOLDT France is. HENSOLDT France is a french company, subsidiary company from HENSOLDT GmbH, a German group. The company originates from the electronics business unit of the defense division of the Airbus Group. At the end of February 2017, Airbus sold this business unit to an US financial investor, KKR. Since 2017, the company has been known as HENSOLDT. The company's name can be traced back to Moritz Carl Hensoldt (1821-1903), a German pioneer of optics and precision mechanics in the 19th century. In 1852, he founded an optical workshop for telescopes, astronomical equipment and microscopes.

HENSOLDT focuses on sensor technologies for protection and surveillance missions in the defense, security and aerospace fields. The main product areas are radars, optoelectronics and avionics. HENSOLDT France and its predecessor companies (SECRE, MATRA, EADS and AIRBUS DS) have started developing and producing crypto-computers, to become one of the first manufacturer of IFF crypto-computers worldwide and the 1st in Europe. Several generations of crypto-computers (Mode 4, Mode 5, and Secure Mode) have been developed to equip platforms such as the Mirage 2000, the Rafale, or the Tiger helicopter.

This work is part of a new project from HENSOLDT to develop the next generation of crypto-computers. The point is to develop a new family of lightweight block ciphers which will be integrated into a constrained hardware platform. Moreover, those algorithms need to be parametrized at will for a given user, while remaining secure. In order to secure the parametrization of these algorithms, a solution of authenticated encryption has to be developed. This solution also needs to be lightweight.

Some of these algorithms can not be disclosed due to the field of activity of HENSOLDT. In the work described in this thesis, the techniques we use will sometimes be illustrated on standard algorithms instead of the proprietary solutions from HENSOLDT.

Usually, in cryptographic papers, three people are introduced to explain and schematize the protocols and definitions. They are named Alice, Bob and either Charlie or Eve (A, B, C and Eve comes from eavesdrop). In this work, I've chosen different protagonists just to personalize this work and thank the people I've been working with. As you may notice in reading further into this paper, Moritz will appear again, as the mascot from HENSOLDT, a little bat (bats are known to use sonar to localize themselves and their environment); he will replace the usual Alice in some schemes and protocols definitions. Bob will still be called Bob and will be represented with Sponge Bob dressed as a pilot. The reader of this work will soon realize that sponge functions are an important part of my field of study and planes and pilots are a wink to HENSOLDT and its past. Lastly, Charlie will be represented as the detective from the famous cartoon Pink Panther, my manager is a huge fan of this cartoon and it seemed the least I could do to thank her.

Once all the characters have been introduced, we can move on to the first part of this work.

Figure 1.1: Moritz

Figure 1.2: Pilot Sponge Bob

Figure 1.3: Detective Clouseau

Chapter 2

Introduction

Cryptographic applications can be found in our everyday life in many objects we do not even think about. Cryptography's goal is to ensure the confidentiality of a message, its authenticity and the identity of the sender. Our credit card is just an embedded crypto system that communicates through secure channel with a terminal to decide whether we are allowed to pay for this meal at a restaurant or for this ticket at a theater. Every Internet request also carries hidden cryptographic protocols that are not noticeable by the user but are used at any moment, to send mails, secure on-line trades, ...
Modern cryptography began in the twentieth century with the definition of classic security notions and the development of new techniques and protocols. Moreover, standards have been defined by national and international institutes to standardize algorithms and protocols to be used in commercial and public applications. Symmetric structures are now well-known and most users can find the candidate that suits their application at best.

One of the remaining challenges in cryptography is to find solutions that can be implemented in small devices and provide adequate security. Indeed, with the Internet of Things, more and more devices are developed, gathering multiple applications in a size which tends towards the smallest as possible. But the size of a device does not mean that breaking its security is not critical. Those devices can be used to secure our cars, our homes, and need to ensure that no attacker can overcome them. It is more than necessary to develop cryptographic solutions for those environments and this is one of the main subject of these three years : develop lightweight solutions.

During this thesis, I had two main objectives. The first one was to develop a new family of lightweight block ciphers that was suitable for my company's applications. This meant to research the state of the art in block ciphers and their lightweight counterparts as well as the mode of operations that can be used using these block ciphers. Lastly the resistance against side-channel analysis and the ease of masking of the adopted solution was a determinant point to tell whether or not the candidate was acceptable for our use case.
The second objective was to design a new solution for authenticated encryption while keeping in mind the lightweight capabilities of our use case. This was achieved by studying the candidates proposed at different cryptographic competition and which elements were of interest for our particular application.

In this work, I start by introducing the context of this thesis, and what was the goal for HENSOLDT France, then I present a quick history of cryptography and which ideas and discoveries led to our modern cryptography. I end this chapter with a quick look at what the future of cryptography may look like with the apparition of quantum computers and post-quantum cryptography. In part II, I present some important definitions for cryptography and cryptographic usage, and give a state-of-the-art of the cryptographic objects that I have used through this work. I start by defining the basic notions of cryptography, cryptographic constructions and security. I then focus on block ciphers, how they can be built, what major security properties they need to achieve and continue by introducing the standards used worldwide. Lastly in part II, I introduce authenticated encryption, in what way it differs from plain encryption, and continue by presenting the current standards and the cryptographic processes that are still at the moment running to define the future standards. Then in Part III, I present my contributions, divided into three chapters: the first one on a self-synchronyzing stream cipher (SSSC) Stanislas, the second one on some hardware implementation of sponge-based candidates to CAESAR

and NIST cryptographic competition on authenticated encryption, and the third one on an attack on the SIV mode of operation. I also give in Appendix A another topic I worked on about: a blockchain protocol to reward useful works. Lastly I conclude this work and give some incensive on the work left to be done on these topics and what I may be focusing on in a near future.

Chapter 3

A brief history of cryptography

Before digging into the subject of this work, let's start with a definition and a bit of history. Cryptography comes from the ancient Greek *krypto* (hidden) and *graphein* (writing) and means writing a secret or a hidden message. Cryptology is the science of secret and is composed of cryptography; the study of techniques and technologies enabling to create a secret; and cryptanalysis: the analysis of cryptography or the science to break a secret. It also includes creating systems enabling secure communication which doesn't include only encrypting but also authentication, or signatures.
The history of cryptography and cryptology is commonly divided into three phases, which are defined not only by a time stamp but also by the means and materials available.
The first phase is about manual cryptography. The techniques used during this phase are limited in both size and complexity. This is the longest one, with more than 5000 years.
The second phase is about mechanized cryptography with the use of mechanic and electronic devices enhancing the maximum security while speeding up the encryption process.
The third phase is not only focused on the means of performing cryptographic operations (computer-aided cryptography) but also on developing new ones like digital signature, authentication, or the public-key cryptography. Not only new algorithms are designed during this phase but also security protocols defined to protect some assets. This phase follows the development of the computer and online communication.
Adding to these three phases, a fourth one can be defined, focused on quantum capabilities and the use of quantum physics to define communication protocols or to create and develop cryptographic algorithms able to resist attacks from a quantum capable adversary.

3.1 First phase, manual cryptography

Cryptography is supposed to be born along with the development of writing, almost six thousand years ago. The first forms of hidden message have been found in ancient stone inscriptions, tablets, or papyrus from different civilizations.

The first recorded correspondence, making use of cryptography was developed by the Spartans, using a cipher device called the *scytale* for secret communication between military commanders. It consists of a baton, spirally wrapped with a strip of parchment or tissue, on which the message is written. Once unwound, the letters of the message are scrambled and the message can only be retrieved by winding the parchment on a baton of same diameter. The scytale is considered as the first transposition cipher.

Figure 3.1: Greek scytale used for secrete communication

Apart from the scytale, most of the cryptographic technique used up to the beginning of the twentieth century consist of substitution ciphers.

Mono-alphabetical substitution The first forms of substitution ciphers can be traced back to the Greeks with the Polybius square. It is a 5x5 square table, each line and column numerated from 1 to 5. The letters were encrypted by giving their coordinates on the grid formed with the table.
The Romans are also famous for their substitution ciphers with the well-known Caesar cipher, a mono-alphabetic substitution with a shift of three positions in the alphabet so that an 'a' became a 'd', a 'b' became an 'e' and so on.

Poly-alphabetical substitution Another known example of historical ciphers is the Vigenère's cipher, published in 1586 by Blaise de Vigenère in Traicté des Chiffres, ou secrètes manières d'escrire (Treaty on Ciphers and secrete writing). This is a poly-alphabetic cipher using a word as the secrete key. In order to encrypt or decrypt a word using the Vigenere's technique, you first need the following table :

	Letters of the message																								
	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
Letters of the Key	Encrypted letters (crossing of column message and line key)																								
A	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
B	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A
C	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B
D	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C
E	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D
F	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E
G	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F
H	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G
I	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H
J	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I
K	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J
L	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K
M	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L
N	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M
O	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N
P	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
Q	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
R	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
S	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
T	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
U	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
V	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
W	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
X	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
Y	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X
Z	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y

Figure 3.2: Vigenere's table

For each letter, find the position in the table corresponding to the crossing of the column for the message the line for the key to get the ciphered letter. To decrypt the message, start with the line corresponding to the first letter of the key, then cross the column corresponding to the letter from the ciphertext to find the decrypted letter.

The first use of cryptanalysis The Arabs were the first people to look more deeply into the principles of cryptography, and to introduce the beginnings of cryptanalysis. They worked on both substitution and transposition ciphers and even discovered the use of letter frequency distributions and probable plaintext in cryptanalysis. [16] They noticed that some letters were more frequently used than others in every language, e.g. 'e' is the most used letters in both English and French. With a statistical analysis of the frequency at which each letter appears, it is possible to make assumptions to break the cipher and find both the plaintext and the key used to encrypt it.

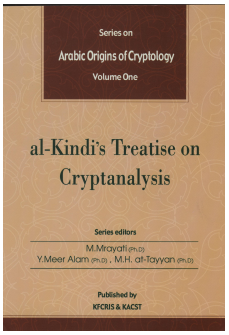


Figure 3.3: al-Kindi's Treatise on Cryptanalysis

3.2 Second phase, mechanized cryptography

Manual cryptography lasted until the beginning of the twentieth century, mostly limited by the complexity of what a code clerk could reasonably do, aided by simple mnemonic devices. As a result, ciphers were limited in size, up to a few thousands characters. Moreover, the complexity of such encryption techniques was also limited in a way that any cryptography scheme could have been used by the ancients if they had known of it.
General principles for both cryptography and cryptanalysis were known, and security was always limited by what could be done manually. Therefore, most systems could be cryptanalyzed, given sufficient ciphertexts and effort.
Moreover, for military purposes, several armies created teams of experts to decrypt opponents' encrypted messages, mastering the cryptanalysis techniques, although the speed to create secrete messages has almost not changed through centuries. The end of the first world war and the use of machines dedicated to cryptography starts the beginning of the second phase.

The rotor machines brought a big change by providing a practical way to mechanize the encryption thus allowing the use of a much larger number of alphabets and enhancing the reachable security. Hugo Koch, Dutch engineer, was the first to patent an electromechanical cipher machine. This then led to the creation of a company developing a cipher machine for public use : the Enigma. The idea will be latter reused by military forces, especially the German ones.

The machine will scramble the 26 letters of the alphabet, using a rotor mechanism. The mechanical subsystem consists of a keyboard, a set of rotors and a series of lamps, representing each letters. This subsystem is linked with an electrical circuit which carries the information. An action on the keyboard will lead to the rotation of the rotors. Those rotors all have electrical contacts on their side, which will create an electrical route when aligned to light up one of the lamps.

The Enigma machine was designed to be symmetrical, i.e. the same key is used to encrypt and decrypt the message. A machine with only one rotor can be seen as a classical substitution cipher with a single shift. Thus, with an alphabet of 26 letters, with one rotor, the machine will have the same security as a hand-written substitution cipher. The security of such machines can be easily increased by adding rotors, which will all spin one position away from the previous one. This system gives a longer alphabet. With each added rotor, the available alphabet is multiplied by itself, in this case by 26, meaning 676 possibility with two rotors, and 17576 with three. In its last version, the Enigma machine was equipped with eight rotors, leading to $26^8 = 208827064576$ possibilities.

The U.S Army had designed another kind of rotor machine: the SIGABA or Converter M-134 that adds randomness on the movement of the rotors. The key is a pierced ribbon. Each time a letter is pressed on the keyboard, the rotors are moved according to the holes in the ribbon, significantly increasing the security of the cipher.

Figure 3.4: An Enigma Machine

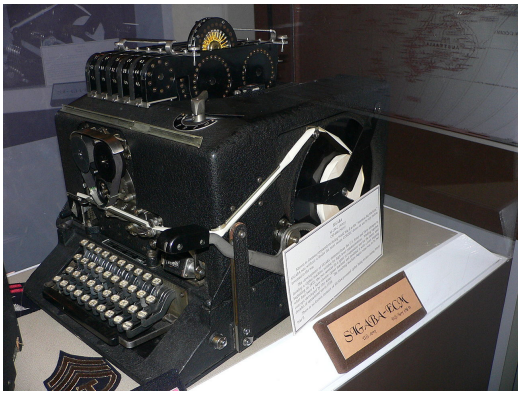


Figure 3.5: The SIGABA Machine

3.3 Third phase, modern and computed cryptography

Until the Second World War, most of the work on cryptography was developed by and for military purposes, to hide secrete and tactical information. However, at the end of the war, even if the commercial use of the Enigma machine was a fiasco, cryptography started to attract public attention, with businesses trying to secure their data from competitors.

Cryptography and Information Theory One of the basis and the conceptualization of modern cryptography is the work of Claude Shannon. First of all, his book called "A mathematical theory of cryptography" [216], explains how to apply information theory to cryptography. From now on, encryption techniques rely on and shall be defined by mathematical properties explaining their security. He also defined two terms which are the basis of defining the security of numerous cipher algorithms:

- Confusion. In its original definition, the confusion states the aim of decorrelating as much as possible the encryption key from the ciphertext. Each character or bit from the ciphertext shall depend on several parts of the key. An algorithm with good confusion should make it difficult to find the key knowing the ciphertext. Moreover, if a single bit of the key is changed, the calculation of the value of most, if not all of the ciphertext shall be affected.

- Diffusion. The idea of diffusion is to hide the relationship between the ciphertext and the plaintext. The goal is to make it hard for an attacker to find the plaintext knowing only the ciphertext. An algorithm with good diffusion means that if the plaintext changes by a single bit, then statistically half of the bits in the ciphertext should change, and similarly, if the ciphertext is changed by one bit, then approximately one half of the plaintext bits should change.

The development of modern technologies and especially computers helped to develop cryptography for public purposes, which lead to three major public advances :

- A public encryption standard (standard for public and commercialized cryptography),

- The first key-exchange protocol,

- The first public-key algorithm.

The Data Encryption Standard In the early 1970's, IBM created a crypto-group to create a form of encryption to secure their customers data, this group was lead by Horst Feistel. They designed a cipher called Lucifer. This algorithm was tweaked by the NSA, later in 1976 and presented as the first public standard for encryption, the Data Encryption Standard (DES) [186]. This standard is described in order to help companies and banks to develop secure electronic communications. The DES is defined with a key of 56 bits and based on a Feistel scheme, from its author's name.

The Diffie-Hellman key exchange In 1976, Whitfield Diffie and Martin Hellman published a research paper on what would be defined as the Diffie-Hellman key exchange [102]. For the first time, they described a system which would allow two correspondents to securely exchange messages without relying on pre-shared keys. The Diffie–Hellman key exchange method allows two parties that have no prior knowledge of each other to jointly establish a shared secret key over an insecure channel.
Take Moritz and Bob who want to create a mutual secret. They both agree on two numbers: p a large prime number et g called the generator which are sent over the communication channel. They then both chose a random large number, which they keep secrete, x for Moritz and y for Bob. Moritz will compute $P_1 = g^x \pmod p$ and Bob $P_2 = g^y \pmod p$ and transmit it. Then Moritz computes $K_1 = P_2^x \pmod p$ and Bob $K_2 = P_1^y \pmod p$. As $K_1 = K_2$ they have managed to compute a secrete value known by them only that can be used as a secrete key for symmetric cryptosystems.
This led almost immediately to the development of a new kind of cryptography : public-key cryptography.

The RSA The RSA cryptosystem is a public-key cryptosystem named after its three inventors: Ronald Rivest, Adi Shamir and Leonard Adleman. It was created in 1977 [195], shortly after the publication of the Diffie-Hellman key exchange. It uses a pair of keys, a public key to encrypt data and a private key to decrypt it. The public and the private keys are mathematically linked. The private key is computationally hard to find, knowing the public one, and the data cannot be decrypted with only the public key.
The security of RSA relies on the practical difficulty of factoring the product of two large prime numbers, the factoring problem. Breaking RSA encryption is known as the RSA problem. Nowadays, the recommended key size is at least 2048 bits.
The RSA algorithm involves four steps: key generation, key distribution, encryption, and decryption. The basic principle behind RSA is the observation that it is practical to find three very large positive integers e, d, and n, such that with modular exponentiation for all integers m (with $0 \leq m < n$):

$$(m^e)^d \equiv m \pmod n$$

and that it is hard to find d, knowing e and n, or even m. The public key is represented by the integers n and e; and, the private key, by the integer d. m represents the message.
The RSA can be used to either encrypt and decrypt messages or to generate a digital signature.

Cryptographic Hash functions Computers also helped developing hash functions and making it readily accessible to the public. The point is to take any string of text or vector of bits, and hash it, to create an output of finite size. This hash value or digest is a footprint of the message and can be used for different purposes. The mathematical properties of a cryptographic hash function make it computationally hard to reverse or at least not in an acceptable time span. Thus, hash functions have seen multiple uses in cryptography :

- Verification of a message integrity
- Digital signature
- Password validation

The cryptographic hash functions have seen a lot of work through the years to create and adapt new standards. The last one is SHA-3, based on the Keccak algorithm [58].

Standards, protocols and new platforms Numerous standards have been developed to inform and give good incentives for implementing secure communications or keeping sensitive information hidden. Even if current standards such as the AES, SHA-3 or RSA are still considered to be secured for years to come, new technologies may require new cryptographic solutions due to their limited size and battery.
The Internet of Things (IoT) development have brought a huge number of little objects which are very limited in size, don't have access to power sources and need to be updated easily. This created the need to develop new protocols of secure communication, and started the search for lightweight cryptographic solutions which may supersede the AES [97] in some cases.

3.4 Towards a fourth phase? Quantum and post-quantum cryptography

Quantum cryptography is a set of techniques aiming to create key exchanges whose security is based on physics rather than on mathematics as it is for classical cryptographic key exchanges. Post-quantum cryptography is the principle consisting in developing cryptographic algorithms and solutions that resist an attacker capable of quantum computation. The third part, quantum computing is about the innovations and difficulties met by the companies trying to develop quantum computers. I have chosen to gather those different points into the same part because they are all linked in a way or another to the quantum technologies but they are not linked neither directly nor chronologically. Moreover, even if post-quantum cryptography is directly linked to the evolution of the quantum computer, it is important for cryptography to be as much as possible in advance of the different existing and future threats.

3.4.1 Quantum cryptography

Quantum cryptography describes the use of quantum physics properties in order to create or enhance cryptographic protocols achieving better security bounds than those created using properties from classical (or causal) physic. One of the main goal is to achieve a secure quantum key distribution between two distant correspondents.
The idea comes from Stephen Wiesner explained in [230] with the concept of conjugated coding.
After the publication of this article, Charles H. Bennett and Gilles Brassard worked out the first quantum key-exchange protocol, based on conjugated observations. This protocol is named BB84 [50] after the name of its authors and the year of publication. In 1990, Artur Ekert developed a key distribution technique based on the quantum correlations between two photons, also called quantum entanglement. This led to the E90 protocol [116].

Protocol Take as an example Moritz and Bob. They are two distant correspondents both having at their disposal

- quantum objects which can be modeled as light pulses (photons, coherent states, entangled pairs of photons),
- a quantum channel which can transmit the light pulses (like an optic fiber),
- a classical channel of communication which needs to be authenticated.

Let's consider a spy trying to crack the system. He can have access to everything that transits between Moritz and Bob, only limited by the laws of physics but cannot access their systems.

The security of the systems relies on two fundamental aspects of the quantum mechanics :

- the no-cloning theorem, stating that it is impossible to duplicate any unknown quantum object,
- the wave function collapse postulate, explaining that any measure realized on a quantum object will modify its quantum properties.

According to the laws specified above, Moritz and Bob can automatically know if anyone is trying to spy on their exchanges, and which quantity of information has been eavesdropped.

Moritz starts by coding a random information on each light pulse and sends it to Bob, through the quantum channel. Bob receives and measures the quantity of information carried by the light pulse and will possess a set of measurements correlated to the data sent by Moritz, which may have been spied on. Depending of the quantity of information left secured between them two, they can create a cryptographic key with a size, at most of the quantity left untouched and unseen.

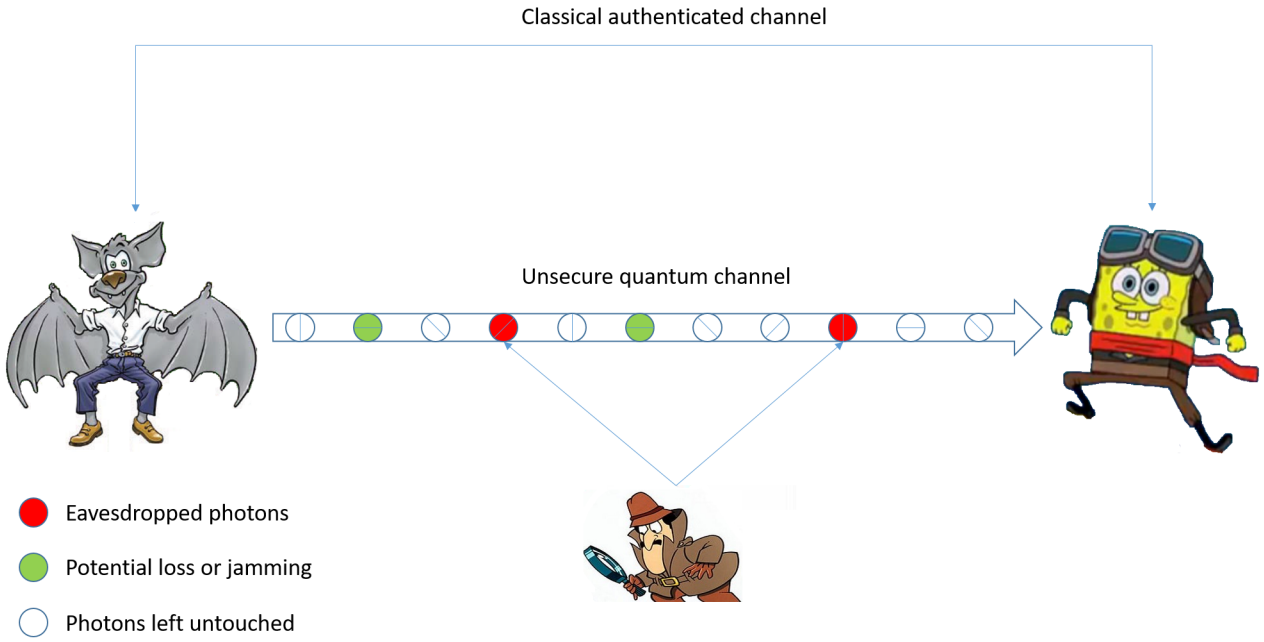


Figure 3.6: Quantum cryptography protocol

Implementation and first results The implementation of a quantum key exchange mainly relies on the ability to generate quickly enough a certain number of quantum objects, measure and transmit them over acceptable distances. Several networks have been implemented first for experimental purposes but nowadays also for commercial uses. The first bank transfer using quantum key distribution was carried out in Vienna in 2004. In 2007, the NIST announced an implementation over an optic fiber across nearly 150km. The same year, a quantum encryption technology was used in Geneva to transmit ballot results for the Swiss national elections. The University of Geneva and Corning Inc. holds the actual record of physical distance, since 2015, with a 307 km-long optic fiber capable of a speed of 12.7kbits/s. In 2016, China sent its second Space Laboratory that carries 14 missions and experimental packages to verify the feasibility of space-to-ground quantum communication. Among all its equipment, Tiangong-2 brought a Space-Earth quantum key distribution and a laser communication experiment. With this satellite, as part of the Quantum Experiments at Space Scale (QUESS), the team managed to measure entangled photons over a distance of 1203 km, sent from a ground station to the satellite and sent back to another ground station. Later in 2017, they managed to implement the BB84 protocol over satellite link between Beijing and Vienna.

3.4.2 Post-quantum cryptography

Post-quantum cryptography is a field of cryptography aiming to provide security against an adversary using a quantum calculator. The cryptographers started from the observation that an opponent with access to a quantum computer can more easily or swiftly break some well-known algorithms used to define security margins of public-key algorithms.
The Shor algorithm is used to solve the discrete logarithm problem and factor integers. This algorithm can be executed in polynomial time if used on a quantum calculator. Thus, the algorithm would be able to break a cipher with a key of n bits in n times the unit of time necessary to do one operation. In comparison, current computers can do it in exponential time, i.e. 2^n operations. This leads to a serious threat problem: every cryptographic primitive based on the discrete logarithm problem (like the Diffie-Hellmann key exchange or the ECDSA signature [149] for example) or the problem of integers factoring (RSA signature) would be vulnerable to an adversary capable of performing quantum computation.
The Grover algorithm enhances the efficiency of research problems, providing a quadratic advantage for an adversary, dividing the security level of most symmetric algorithms by two. In order to manage/lessen this disadvantage, symmetric algorithms need to double the size of their key.
For a symmetric algorithm to be secured in a quantum world, it would need to double the size of its current key, if described with an internal mathematical block suitable for this size (i.e. the AES [97]

would be considered secure with its version with a key of 256 bits).

Scheme	Affect
Symmetric Key	Security halved (Grover)
Hash (SHA-3)	Security decreased (Grover)
Public Key (RSA, DSA)	Completely broken (Shor)
Lattice-based cryptography	Quantum safe
Multivariate Cryptography	Qauntum safe
Hash-based Cryptography	Quantum safe
Code-based cryptography	Quantum safe
Isogeny Cryptography	Quantum Safe

Table 3.1: Classic structure security in the post-quantum world

In order to prevent such attacks and prepare the future of cryptography, the NIST (the American National Institute of Standards and Technology) and the CACR (the Chinese Association for Cryptographic Research) both launched a competition to develop and standardize post-quantum public-key algorithms. The Chinese post-quantum competition ended in 2019 with two different use cases: Public-key algorithms and grouping algorithms (digital signatures schemes or Key-exchange mechanisms). Winners are respectively uBlock [236] and Ballet [92], and Aigis-sig [239], LAC.PKE [173] and Aigis-enc. The Post-Quantum Cryptography Standardization is a competition launched by the NIST, announced at PQCrypto 2016 [144]. The first submission deadline at the end of 2017 met a total of 23 signature and 59 encryption schemes, among which 69 (19 signature and 45 KEM/encryption schemes) were deemed well enough designed to enter competition. In July 2020, the NIST announced seven finalists and eight alternate algorithms. These seven finalists are thought to be the most promising and will be considered for standardization at the end of the third round. The second track is still to be considered if attacks on the finalists are found feasible before the end of the standardization process, or if any intellectual property is still left to be solved on a finalist candidate.

Type	PKE/KEM	Signature
FINALISTS		
Lattice	CRYSTALS-KYBER [72] NTRU [87] SABER [42]	CRYSTALS-DILITHIUM [35] FALCON [140]
Code-based	Classic MacEliece [17]	
Multivariate		Rainbow [103]
ALTERNATE CANDIDATES		
Lattice	FrodoKEM [18] NTRU Prime [53]	
Code-based	BIKE [27] HQC [181]	
hash-based		SPHINCS+ [30]
Multivariate		GeMSS [77]
Supersingular Elliptic Curve Isogeny	SIKE [99]	
Zero-knowledge proofs		Picnic [85]

Table 3.2: NIST PQC finalists and alternate algorithms

3.4.3 The quantum computer

The quantum computer can provide an enormous computing capacity. While a classical register of N bits may only store one array of N binary values at a time, a N qubits register may, thanks to the

quantum superposition, store every possible combinations which represents 2^N arrays of binary values, each array associated with a given probability. When the register is read, the quantum superposition collapses and only the good result is shown.
Currently, even for supercomputers, it is unfeasible to break RSA keys with 2048 or more bits, which is the recommended size. In 2003, Stéphane Beauregard proved that to factor an N bits integer, $2N + 3$ qubits are enough [45]. Breaking a 2048 bits RSA key would then only require 4099 qubits.
The quantum computer could be used not only for cryptography, but in many fields including biology, physics, chemistry, mathematics, which explains the billions of dollars invested in the research and development of the future quantum computers. One main issue with current quantum computers is their lack of efficiency. The superposition phase is highly unstable and the systems are too prone to errors. Thus the systems require numerous physical qubits in order to get one stable logical qubit (hundreds or maybe thousands). The first issue to solve is then to find an appropriate solution for quantum error correcting code; two methods have been published, by Peter Shor [220] and Andrew Steane [222] which are the first step to efficient quantum error correcting. In 2019, a team of researchers showed that to break a RSA code with 2048 bits within eight hours, one would need to have twenty million physical qubits.
In 2019, Google advertised their quantum supremacy by achieving the computing of one precise computing in around 200 seconds where a supercomputer would need 10,000 years, using 53 qubits. This is, to the best of our knowledge, the best advances in the field of quantum computers to this day. To achieve an efficient and useful quantum computer, a lot of technological issues are yet to solve and it may be impossible to see a functional quantum computing in the years or even decades to come.

Part II

State-of-the-Art

Chapter 4

A few definitions

4.1 Asymmetric and symmetric cryptography

Asymmetric cryptography or public-key cryptography is a system that uses a pair of keys: a public one and a private one. The public key can be openly distributed or shared with a group of users while the private key is known only by its owner. The public key is generated using a cryptographic algorithm producing a one-way function from the private key, ensuring the security of the system.
In a public-key system, a message can be encrypted using the public key of the recipient. This message can then only be decrypted using the private key associated. This results in a secure channel between two users created from public data.
Public-key cryptography is also used as an efficient method to create a robust authentication to create a digital signature of a message, using the private key. Anyone with the sender's public key can verify the signature of the message.

Symmetric cryptography mechanisms use algorithms that wield the same cryptographic keys to perform both encryption and decryption of a message. The key or set of keys is a shared secret between the parties that is used to maintain a secure communication link. The symmetric primitives can be either block ciphers, stream ciphers or hash functions. The main difficulty with symmetric cryptography is the way used to obtain the shared secret and thus establish the secure link between the different users. Therefore symmetric and asymmetric cryptography are often used in conjunction: the asymmetric part is used to generate and securely exchange the symmetric keys later used to communicate with the shared secret.

4.2 Block ciphers

A symmetric cryptographic algorithm usually takes as input two vectors, namely a message and a key. In the case of block ciphers, the set of all possible blocks of messages are of fixed length. The algorithm then applies a permutation to the block. Commonly, the application of the permutation is called encryption and converts a plaintext into a ciphertext, while the inverse permutation is called decryption and converts a ciphertext into a plaintext. Computing the inverse of the permutation usually have identical cost to computing the forward permutation.

4.3 Stream ciphers

As opposed to block ciphers that operate on fixed length, a stream cipher computes a ciphertext stream by combining a digit of the plaintext with the corresponding digit of the keystream. Typically, the digit is a bit and the combination of the plaintext and the keystream an exclusive-or (XOR). But other solutions exist using bytes or half-bytes and more complex combinations of the message and the key. The keystream is often generated from a random seed value (the initial key), extended using different operations such as shift registers or filters.
A stream cipher can also be constructed from a block cipher, using a mode of operation such as the Counter Mode (CTR).
Stream ciphers have received less interest than block ciphers from the cryptographic community point of view and many primitives have been found insecure up to the point that in 2004, at Asiacrypt crypto-conference, Adi Shamir asked whether stream ciphers were dead or not. This discussion ended up on a new cryptographic project called eStream [7] launched by the European Network of Excellence in Cryptology (ECRYPT), started in October 2004 and finished in September 2007. From the 35 candidates, 7 were selected as winners, covering two categories: primitives oriented towards software or towards hardware platforms.

Software	Hardware
HC-128 [232]	Grain v1 [138]
Rabbit [65]	MICKEY v2 [34]
Salsa20/12 [55]	Trivium [101]
Sosemanuk [51]	

4.4 Modes of operation

A mode of operation is an algorithm that takes as core part a block cipher and uses it in a way to provide capabilities unachievable with a block cipher. A block cipher is only suitable for the cryptographic transformation of one fixed-length block.
Mode of operations can be used for different purposes: create a stream cipher, generate authenticated encryption, to create shorter or larger blocks. Several modes are defined and standardized by the NIST to achieve confidentiality while using a properly chosen block cipher [111].

Electronic Code Book (ECB) mode This is the simplest of the mode of operations, the message is divided into blocks, and each block is encrypted separately. The main flaw of this mode is the lack of diffusion. ECB encrypts identical plaintext blocks into identical ciphertext blocks, and does not hide data patterns. ECB is not recommended for use in cryptographic protocols. It can also make protocols without integrity protection even more susceptible to replay attacks, since each block gets decrypted in exactly the same way.

The Cipher Block Chaining (CBC) mode was invented and patented in 1976 [115] by Ehrsam, Meyer, Smith and Tuchman. Each block of plaintext is XORed with the previous ciphertext block before being encrypted. This way, each block of ciphertext depends on all the plaintext blocks processed up to that point. To make each message unique, an initialization vector must be used with the first block.

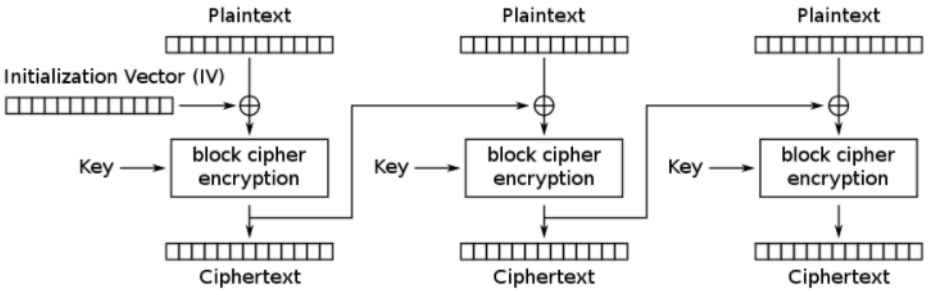


Figure 4.1: Cipher Block Chaining mode of operation

By denoting P_i with $i \in \{0, \ldots, n\}$ the i^{th} block of plaintext, C_i the i^{th} block of ciphertext and E_K the block cipher E with the key K, we have $C_0 = E_k(IV \oplus P_0)$ and $C_i = E_k(C_{i-1} \oplus P_i)$.
CBC is the most commonly used mode of operation. In opposition to the ECB mode of operation, any change on either the plaintext or the IV will have an effect on all of the ciphertext blocks.
Its main drawbacks are that encryption is sequential which means that it cannot be parallelized. Moreover, the message must be padded to a multiple of the cipher block size or the mode must use the ciphertext stealing method.

Cipher Feedback (CFB) mode is a mode of operation, operating as a stream cipher. It is quite similar to CBC with the difference that the plaintext is XORed with the output of the encryption operation. In CFB, using same notations as in the previous paragraph, $C_0 = E_k(IV) \oplus P_0$ and $C_i = E_K(C_{i-1}) \oplus P_i$.

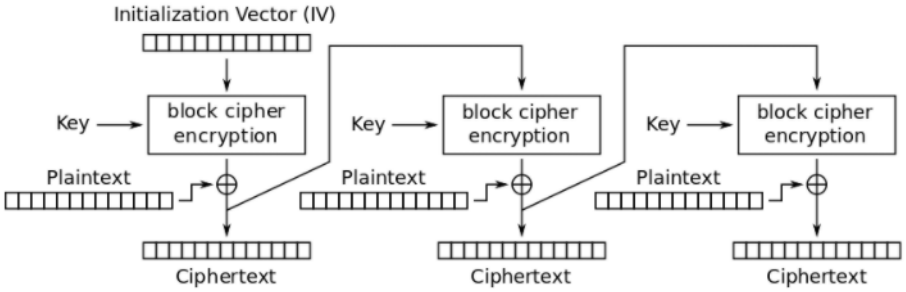


Figure 4.2: Cipher Feedback mode of operation

The main advantage of this mode of operation is that only the encryption part of the block cipher algorithm is needed, which can save some memory if the encryption and decryption operations are not the same (as it is the case for the AES). Moreover, the message does not need to be padded to a multiple of the cipher block size.
Like with the CBC mode, changes in the plaintext propagate forever in the ciphertext, and encryption cannot be parallelized. Also like CBC, decryption can be parallelized.

Output Feedback (OFB) mode is also a synchronous stream cipher mode of operation. It processes the data in the same way as CFB, but instead of carrying the ciphertext computed at step i to the next one, the data carried is the output of the encryption computation. In other words, $C_0 = E_K(IV) \oplus P_0$, $C_i = E_K^i(IV) \oplus P_i$ with E_K^i the composition i times of the E_K operation.

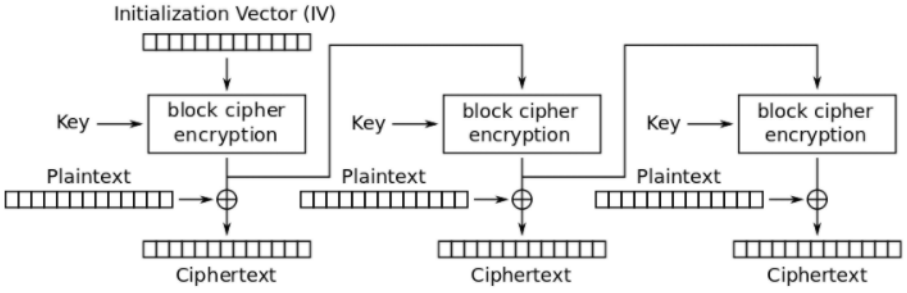


Figure 4.3: Output Feedback mode of operation

Because of the symmetry of the XOR operation, encryption and decryption are exactly the same. Each output feedback block cipher operation depends on every previous ones, and so can not be performed in parallel. However, because the plaintext or ciphertext is only used for the final XOR, the block cipher operations may be performed in advance, allowing the final step to be performed in parallel once the plaintext or ciphertext is available.
One can obtain an OFB mode keystream by using CBC mode with a constant vector of zeroes as input. It can allow fast hardware implementations of CBC mode for OFB mode encryption.

CounTeR (CTR) mode was introduced by Whitfield Diffie and Martin Hellman in 1979. It turns a block cipher into a stream cipher and generates the next keystream block by encrypting successive values of a "counter". The counter can be any function which produces a sequence which is guaranteed not to repeat for a long time, although an actual increment-by-one counter is the simplest and most popular.

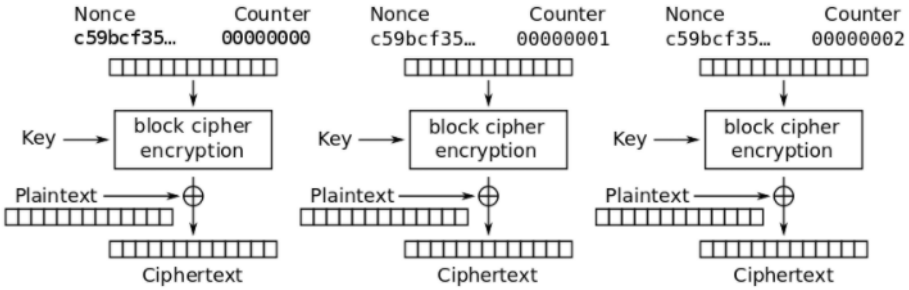


Figure 4.4: CounTer mode of operation

$C_i = E_K(N||ctr_i) \oplus P_i$ with ctr_i the i^{th} counter (usually the value i coded on the desired size), and N the nonce (number only used once) that must be picked at random and vary for each call to the CTR mode. The nonce does not need to be kept secret to ensure the security of the mode.
CTR mode is particularly well suited for multi-processor machines where blocks can be encrypted in parallel.

4.5 Hash functions

A hash function is a mathematical function that computes a digital print of a data taken as input. This print can be used to identify the data it is calculated from. A cryptographic hash function will take as input a data of arbitrary size and compute a hash value or message digest of fixed size. It is a one-way function, meaning that it is supposedly really hard to invert. The ideal cryptographic hash function has the following main properties:

- it is deterministic, the same message always results in the same hash
- it is quick to compute the hash value for any given message
- it is unfeasible to generate a message that yields a given hash value (i.e. to reverse the process that generated the given hash value)
- it is unfeasible to find two different messages with the same hash value
- a small change to a message should change the hash value so extensively that the new hash value appears uncorrelated with the old hash value (avalanche effect)

Those functions can be used for digital signatures, message authentication codes (MAC) or be part of digital authentication protocols.

Hash function security Whereas ciphers protect data confidentiality in an effort to guarantee that data sent in the clear can not be read, hash functions protect data integrity in an effort to guarantee that data has not been modified. If a hash function is secure, two distinct pieces of data should always have different hashes. A hash can thus serve as an identifier.
The most common application of a hash function is digital signatures. When digital signatures are used, applications process the hash of the message to be signed rather than the message itself. If a single bit is changed in the message, the hash of the message will be totally different. The hash function thus helps ensure that the message has not been modified.
The strength of a hash function is directly linked to the unpredictability of its outputs. By definition, a secure hash function behaves as much as possible like a truly random function (sometimes called a random oracle). Specifically, a secure hash function should not have any property or pattern that a random function does not. This definition is helpful for theoreticians, but in practice we need more specific notions: namely, preimage resistance and collision resistance.

Preimage Resistance A preimage of a given hash value, H, is any message, M, such as $Hash(M) = H$. Preimage resistance describes the security guarantee that given a random hash value, an attacker will not find a preimage of that hash value. Hash functions are sometimes called one-way functions because it is easy to obtain a hash from the message but it is computationally hard to find the message from the hash.
Even given unlimited computing power, one would not be able to determine the message picked to produce a particular hash, since there are many messages hashing to the same value. For example, there are 2^{256} possible values of a 256-bit hash (the typical length with hash functions used in practice), but there are many more values of 1024-bit messages (2^{1024} possible values). Therefore, each possible 256-bit hash value will have about $2^{1024}/2^{256} = 2^{1024-256} = 2^{768}$ preimages of 1024 bits each.
In practice, the goal is to be assured that it is practically impossible to find any message that maps to a given hash value, which is what preimage resistance actually stands for. Specifically, two terms exist: first-preimage and second-preimage resistance. First-preimage resistance (or just preimage resistance) describes cases where it is practically impossible to find a message that hashes to a given value. Second-preimage resistance, on the other hand, describes the case that when given a message, M_1, it is practically impossible to find another message, M_2, that hashes to the same value that M_1 does.

Collision Resistance Whatever chosen hash function, collisions will inevitably exist due to the pigeonhole principle. However, collisions should be hard to find as it is for the original message in order for a hash function to be considered collision resistant (an attacker should not be able to find two distinct messages that hash to the same value). A hash of n bits can be broken in $\mathcal{O}(2^{n/2})$ time

steps while a preimage can be found in $\mathcal{O}(2^n)$
The notion of collision resistance is related to the notion of second preimage resistance: if it is possible to find second preimages for a hash function, it is also possible to find collisions. Thus, any collision-resistant hash is also second preimage resistant. If this was not the case, there would be an efficient solve-second-preimage algorithm that could be used to break collision resistance.

The SHA Family Several cryptographic hash algorithms have been standardized by the American Institute of Standards and Technology (NIST) through the years as Federal Information Processing Standards (FIPS).
The Secure Hash Algorithm (SHA) hash functions are standards defined by the NIST for use by non-military federal government agencies in the US.
They are considered worldwide standards, and only certain non-US governments opt for their own hash algorithms (such as China's SM3, Russia's Streebog, and Ukraine's Kupyna) for reasons of sovereignty rather than a lack of trust in SHA's security. The US SHAs have been more extensively reviewed by cryptanalysts than the non-US ones.

- SHA-1 The SHA-1 standard comes from the failure in the NSA's original SHA-0 hash function. In 1993, NIST standardized the SHA-0 hash algorithm, but in 1995 the NSA released SHA-1 to fix an unidentified security issue in SHA-0. The reason for the tweak became clear when in 1998, two researchers discovered how to find collisions for SHA-0 in about 2^{60} operations instead of the 2^{80} expected for 160-bit hash functions such as SHA-0 and SHA-1. Later attacks reduced the complexity to around 2^{33} operations for near-collisions and aroud 2^{39} for real collisions, leading to actual collisions in less than an hour for SHA-0 [228].
  SHA-1 is based on a Merkle–Damgård hash function. That is, SHA-1 works by iterating the following operation over 512-bit message blocks $(M) : H = E(M, H) + H$. SHA-1 is the basis of the block cipher, called SHACAL.

- SHA-2, the successor to SHA-1, was designed by the NSA and standardized by the NIST. SHA-2 is a family of four hash functions: SHA-224, SHA-256, SHA-384, and SHA-512. The three-digit numbers represent the bit lengths of each hash.
  The initial motivation behind the development of SHA-2 was to generate longer hashes and thus deliver higher security levels than SHA-1.

The SHA-3 Competition In 2007, the NIST announced a Hash Function Competition (the official name of the SHA-3 competition) that began with a call for submissions and some basic requirements: hash submissions were to be at least as secure and as fast as SHA-2, and they should be able to do at least as much as SHA-2. By 2008, the NIST had received 64 submissions from around the world, including those from universities and large corporations. Of these 64 submissions, 51 matched the requirements and entered the first round of the competition.
During the first weeks of the competition, cryptanalysts mercilessly attacked the submissions. In July 2009, NIST announced 14 candidates for the second round. After spending 15 months analyzing and evaluating the performance of these candidates, the NIST chose five finalists:

- BLAKE [32] An enhanced Merkle–Damgård hash whose compression function is based on a block cipher, which is in turn based on the core function of the stream cipher ChaCha, a chain of additions, XORs, and word rotations. BLAKE was designed by a team of academic researchers based in Switzerland and the UK.

- Grøstl [125] An enhanced Merkle–Damgård hash whose compression function uses two permutations (or fixed-key block ciphers) based on the core function of the AES [97] block cipher. Grøstl was designed by a team of seven academic researchers from Denmark and Austria.

- JH [233] A tweaked sponge function construction wherein message blocks are injected before and after the permutation rather than just before. The permutation also performs operations similar to a substitution-permutation block cipher (as discussed in Chapter 4). JH was designed by a cryptographer from a university in Singapore.

- Keccak [58] A sponge function whose permutation performs only bitwise operations. Keccak was designed by a team of four cryptographers working for a semiconductor company based in Belgium and Italy, and included one of the two designers of AES.

- Skein [121] A hash function based on a different mode of operation than Merkle–Damgård, and whose compression function is based on a block cipher that uses only integer addition, XOR, and word rotation. Skein was designed by a team of eight cryptographers from academia and industry, all but one of whom are based in the US, including the renowned Bruce Schneier.

After extensive analysis of the five finalists, the NIST announced a winner: Keccak. NIST's report rewarded Keccak for its elegant design, large security margin, good general performance, excellent efficiency in hardware, and its flexibility.

Keccak (SHA-3) One of the reasons that NIST chose Keccak is that it is completely different from SHA-1 and SHA-2. Its design is based on the sponge function. Keccak's core algorithm is a permutation of a 1600-bit state that ingests blocks of 1152, 1088, 832, or 576 bits, producing hash values of 224, 256, 384, or 512 bits, respectively, the same four lengths produced by SHA-2 hash functions. But unlike SHA-2, SHA-3 uses a single core algorithm rather than two algorithms for all four hash lengths.
Another reason is that Keccak is more than just a hash. The SHA-3 standard document FIPS 202 defines four hashes: SHA3-224, SHA3- 256, SHA3-384, and SHA3-512 and two algorithms called SHAKE128 and SHAKE256. (The name SHAKE stands for Secure Hash Algorithm with Keccak.) These two algorithms are extendable-output functions (XOFs), or hash functions that can produce hashes of variable length, even very long ones. The numbers 128 and 256 represent the security level of each algorithm.

4.6 Message Authentication Code

Definition A message authentication code (MAC) is a block of information, or tag, used to authenticate a message (ensure that the message comes from the stated sender) and verify that it has not been modified. The MAC value protects both the integrity and the authenticity of a message, by allowing verifiers (who possess the secret key used to generate the MAC) to detect any changes to the message content.
A message authentication code system consists of three algorithms:

- A key generation algorithm that selects a key randomly from the key space featuring uniform distribution.
- A signing algorithm that returns the tag computed from the key and the message.
- A verifying algorithm that verifies the authenticity of the message given the key and the tag. It returns either accepted if the MAC is correct or $\perp$ if not.

Standards Various standards defining MAC algorithms exist. These include:

- FIPS PUB 113 Computer Data Authentication, withdrawn in 2002, defined an algorithm based on DES.
- FIPS PUB 198-1 The Keyed-Hash Message Authentication Code (HMAC)
- ISO/IEC 9797-1 Mechanisms using a block cipher
- ISO/IEC 9797-2 Mechanisms using a dedicated hash-function
- ISO/IEC 9797-3 Mechanisms using a universal hash-function
- ISO/IEC 29192-6 Lightweight cryptography - Message authentication codes

ISO/IEC 9797-1 and -2 define generic models and algorithms that can be used with any block cipher or hash function, and a variety of different parameters. These models and parameters allow more specific algorithms to be defined by nominating the parameters. For example, the FIPS PUB 113 algorithm is functionally equivalent to ISO/IEC 9797-1 MAC algorithm 1 with padding method 1 and a block cipher algorithm of DES.

4.7 Security definitions

4.7.1 Basic security notions

Before introducing the main families of symmetric primitives in the next chapter, we formalize three theoretical objects : Pseudo Random Function (PRF), Pseudo Random Permutation (PRP) and Pseudo Random Number Generator (PRNG). These notions are needed to define the security requiremens on symmetric primitives.

Pseudo Random Function Let's consider a function f_K from n bits to m bits, with n and m two integers and with K a secret key taken from the set of all k-bit secret keys under the uniform distribution and f^* a random function from the space of all possible functions of n bits to m bits. We denote A a probabilistic algorithm that outputs either 1 or 0 when given function f as input. The algorithm aims to distinguish a function f_K from f^*.
The collection $F = \{f_K | K \in \mathcal{K}\}$ is said to be a PRF if the advantage of any probabilistic algorithm A for distinguishing f_K from f^* is small, with the following equality :

$$Adv_F^{PRF}(A) = |Pr[A(f_K) = 1] - Pr[A(f^*) = 1]|$$

Pseudo Random Permutation The definition of a PRP derives from the definition of a PRF. Let's consider π_K a permutation of n bits with K a secret key taken from the set of all k-bit secret keys under the uniform distribution and π^* a random permutation from the space of all possible permutation of n bits to n bits. We denote A a probabilistic algorithm that outputs either 1 or 0 when given a permutation π as input.
The collection $\Pi = \{\pi_K | K \in \mathcal{K}\}$ is said to be a PRP if the advantage of any probabilistic algorithm A for distinguishing π_K from π^* is small, with the following equality :

$$Adv_\Pi^{PRP}(A) = |Pr[A(\pi_K) = 1] - Pr[A(\pi^*) = 1]|$$

Pseudo Random Number Generator Let's consider a function s that takes as input a secret key K of k bits and outputs a binary string of m bits and s^* a random binary string of m bits. We denote A a probabilistic algorithm that outputs either 1 or 0 when given a binary string.
s is a PRNG if the advantage of any probabilistic algorithm A for distinguishing $s(K)$ from s^* is small, with the following equality :

$$Adv_s^{PRNG}(A) = |Pr[A(s(K)) = 1] - Pr[A(s^*) = 1]|$$

4.7.2 Indistinguishability

In order to judge the usability of a cryptographic primitive, the main property to evaluate is its security. The first definition of security was given by Claude Shannon in 1949 [215]. A cipher is considered as secure if no one not knowing the key is able to retrieve the plaintext knowing the ciphertext nor gain any information on either the key or the message. To achieve Shannon's security, one needs to use a key at least as long as the message to encrypt and to use a new key for each new message. This model is not realistic thus another security definition is needed.
The concept of impossible in cryptography can be defined with the two following notions: informational security and computational security. The first notion is about theoretical impossibility whereas the second about practical impossibility.
Informational security tries to define whether it is conceivable to break a cipher or not. Thus, a cipher is informationally secure if it cannot be broken even if an attacker is given an unlimited computation time and memory. The one-time pad is informationally secure. It encrypts a plaintext P to a ciphertext C using a key K which is a random bit-string that is unique to each plaintext, such as $C = P \oplus K$. Knowing C, even given unlimited time to try all possible keys, and compute the corresponding plaintext, it is still unfeasible to identify the right key as there are as many possible P as there are K.
Computational security defines a cipher as secure if it cannot be broken given a reasonable amount of time and resources, such as memory, budget, energy, etc. Consider a cipher E, with a known plaintext-ciphertext pair (P, C) such as $C = E(P, K)$. The plaintext, key and ciphertext are all 128-bit long.

This cipher is not informationally secure as you can break it using brute-force and the 2^{128} possibles keys. In practice, testing all the keys is unfeasible, as, even having access to all the computational capability on earth it would still require an unreasonable time to test all keys. The cipher is then computationally secure.
We then need to define notions and means to calculate the ability of a cipher to resist to attacks and to quantify the security. Usually, the values used are the computation time it takes to break a cipher, usually named t and the memory needed to gather of the information needed to break it. The probability of success of an attack ϵ is also often used.
We then say that a cryptographic scheme is (t, ϵ)-secure if an attacker performing at most t operations has a probability of success that is no higher than ϵ, with $0 \leq \epsilon \leq 1$. t and ϵ are just limits: if a cipher is (t, ϵ)-secure, then no attacker performing fewer than t operations will succeed (with probability higher than ϵ). But that does not imply that an attacker doing exactly t operations will succeed, and it does not tell you how many operations are needed, which may be much larger than t. It is simply a lower bound on the computation effort needed, because one would need at least t operations to compromise security.
Sometimes it is possible to know precisely how much effort it takes to break a cipher. In such cases, a (t, ϵ)-security gives a tight bound when an attack exists that breaks the cipher with probability ϵ and exactly t operations. For example, consider a symmetric cipher with a 128-bit key. Ideally, this cipher should be $(t, t/2^{128})$-secure for any value of t between 1 and 2^{128}. The best attack should be brute force (trying all keys until you find the correct one). Any better attack would have to exploit some imperfection in the cipher, the goal is thus to create ciphers where brute force is the best possible attack.
Given the statement $(t, t/2^{128})$-secure, let's examine the probability of success of three possible attacks:

- $t = 1$, an attacker tries one key and succeeds with a probability of $\epsilon = 1/2^{128}$.
- $t = 2^{128}$, an attacker tries all 2^{128} keys and one succeeds. Thus, the probability $\epsilon = 1$ (if the attacker tries all keys, obviously the right one must be one of them).
- an attacker tries only $t = 2^{64}$ keys, and succeeds with a probability of $\epsilon = 2^{64}/2^{128} = 2^{-64}$.

When an attacker only tries a fraction of all keys, the success probability is proportional to the number of keys tried. A cipher with a key of n bits is at best $(t, t/2^n)$-secure, for any t between 1 and 2^n, because no matter how strong the cipher, a brute-force attack against it will always succeed. The key thus needs be long enough to make brute-force attacks unfeasible in practice. But the key size does not always match the security level, it only gives an upper bound, or the highest possible security level.
Choosing a security level using current standard block ciphers is equivalent to selecting between 128-bit and 256-bit security. 128-bit security means that 2^{128} operations are needed to break that cryptosystem. Consider the fact that the universe is approximately 2^{88} nanoseconds old. Since testing a key with current technology takes no less than a nanosecond, an attacker would need several times the age of the universe to succeed if it takes exactly one nanosecond to test a key. Even forecasting the Moore's law still true for decades to come (every two years, a computer is able to compute 1 more bit key in one hour) the current 128-bits security will hold enough for years.

Computational indistinguishability Let's have X and Y, two distributions over $\{0,1\}^n$. For every $A : \{0,1\}^n \to \{0,1\}$, X and Y are computationally indistinguishable if for every polynomial-time A and every polynomially bounded ϵ we have:

$$|Pr[(A(X) = 1] - Pr[A(Y) = 1]| \leq \epsilon$$

Ciphertext indistinguishability is a property of many encryption schemes. Intuitively, if a cryptosystem possesses the property of indistinguishability, then an adversary will be unable to distinguish pairs of ciphertexts based on the message they encrypt. The property of indistinguishability under chosen plaintext attack is considered a basic requirement for most provably secure cryptosystems, though some schemes also provide indistinguishability under chosen ciphertext attack and adaptive chosen ciphertext attack. Indistinguishability under chosen plaintext attack is equivalent to the property of semantic security, and many cryptographic proofs use these definitions interchangeably.

A cryptosystem is considered secure in terms of indistinguishability if no adversary, given the result of the encryption of one message randomly chosen in a two-element space, is able to identify which of the two elements was chosen with a probability significantly better than random guessing (1/2). If any adversary can succeed in distinguishing the chosen message (respectively ciphertext) with a probability greater than 1/2, then this adversary has an advantage in distinguishing the message (respectively ciphertext), and the cryptosystem is not considered secure in terms of indistinguishability. By definition, with a secure scheme, no adversary should be able to learn any information from seeing a ciphertext, therefore no adversary should be able to do better than just guessing randomly.

4.7.3 Cryptanalysis

Results of a cryptanalysis - Partial vs total break Attacks can be classified based on what type of information the attacker has available.

- Ciphertext-only: the attacker has access only to a collection of ciphertexts.
- Known-plaintext: the attacker has a set of ciphertexts and their corresponding plaintext.
- Chosen-plaintext (chosen-ciphertext): the attacker can obtain the ciphertexts (plaintexts) corresponding to an arbitrary set of plaintexts (ciphertexts) they may choose.
- Adaptive chosen-plaintext: a chosen-plaintext attack where the attacker can choose plaintexts based on information learned from previous encryptions, similarly to the Adaptive chosen ciphertext attack.
- Related-key attack: a chosen-plaintext attack where the attacker can obtain ciphertexts encrypted under two different keys. The keys are unknown, but the relationship between them is known (for example two keys that differ in the one bit).

The results of cryptanalysis can also vary in usefulness. Lars Knudsen classified various types of attack on block ciphers according to the amount and quality of secret information that was discovered:

- Total break — the attacker deduces the secret key.
- Global deduction — the attacker discovers a functionally equivalent algorithm for encryption and decryption, without learning the key.
- Instance deduction — the attacker discovers plaintexts (or ciphertexts) not previously known.
- Information deduction — the attacker gains some Shannon information about plaintexts (or ciphertexts) not previously known.
- Distinguishing algorithm — the attacker can distinguish the cipher from a random permutation.

Academic attacks are often against weakened versions of a cryptosystem, such as a block cipher or hash function with less rounds. Many attacks become exponentially more difficult to execute as rounds are added to a cryptosystem, so it's possible for the full cryptosystem to be strong even though reduced-round variants are weak. Nonetheless, partial breaks that come close to breaking the original cryptosystem may mean that a full break will follow. For example, the successful attacks on DES, MD5, or SHA-1 were all preceded by attacks on weakened versions.

In academic cryptography, a weakness or a break in a scheme is usually defined quite conservatively: it might require impractical amounts of time, memory, or known plaintexts. It also might require the attacker to be able to do things many real-world attackers can not: for example, the attacker may need to choose particular plaintexts to be encrypted or even to ask for plaintexts to be encrypted using several keys related to the secret key. Furthermore, it might only reveal a small amount of information, enough to prove the cryptosystem imperfect but too little to be useful to real-world attackers. Finally, an attack might only apply to a weakened version of cryptographic tools, like a reduced-round block cipher, as a step towards breaking of the full system. As long as the attack is more efficient than brute force (i.e. trying all possible keys), the attack is considered valid.

Brief history of cryptanalysis Given some encrypted data, the goal of the cryptanalyst is to gain as much information as possible about the unencrypted data.
Historically, cryptography was first academically studied in 1981 with the first CRYPTO conference where observations on some undesirable properties of the DES were established. Practically, most cryptanalytic techniques were developed in the 1990s. A number of them are variants from two main discoveries. First, differential cryptanalysis was found by Biham and Shamir and presented at CRYPTO 90 [60].
Then, linear cryptanalysis was developed by Matsui and presented at EUROCRYPT 93 [177].
The cryptographers community then tried to either use or enhance these techniques to break public ciphers and some introduced original improvements. These two techniques also led to the development of criteria for security evaluation of block ciphers.

Differential cryptanalysis In 1990, Eli Biham and Adi Shamir introduced differential cryptanalysis, a chosen-plaintext attack for analyzing ciphers based on substitutions and permutations. Applied to DES, the attack is more efficient than brute force, but it requires a large number of chosen plaintexts. As compared to brute force, which requires a single known plaintext/ciphertext pair and takes 2^{56} operations, differential cryptanalysis requires 2^{36} chosen plaintext/ciphertext pairs and 2^{37} operations.
Since the publication of differential cryptanalysis, one of DES's designers revealed that the DES design team already knew about differential cryptanalysis when the cipher was published as standard, but had to keep it secret for reasons of national security. He also revealed that they chose the specific substitution and permutation parameters of DES to provide as much resistance to differential cryptanalysis as possible.
The goal of differential cryptanalysis is to use several plaintexts with a fixed XOR difference and analyze the difference in the resulting ciphertext given by each plaintext. Using these differences, each possible key is assigned a probability of success. The more plaintext/ciphertext couples are used, the more the attack can focus on a smaller number of probable keys. Once a sufficiently small number of keys is found, the attacker can perform a brute force search on the remaining probabilities.
Applied to block ciphers, differential analysis can be described as a set of techniques aiming at tracing differences through the network of transformation, discovering where the cipher exhibits what is known as non-random behavior and exploiting such details to recover the secrete key (cryptography key).
For any particular cipher, the input difference must be keenly selected for the attack to be successful. An analysis of the algorithm's internals is undertaken; the standard method is to trace a path of highly probable differences through the various stages of encryption, referred to as differential characteristic. In the process, observing the desired output difference between the two chosen or unknown plaintext inputs suggests possible key values.

Linear cryptanalysis Linear cryptanalysis is basically a known plaintext attack, in which an attacker studies the linear relations between the plaintext and the ciphertext that hold with a certain probability. This approximation can be used to assign probabilities to the keys and find the most probable one.
The focus of linear cryptanalysis is the statistical analysis against one round of decrypted ciphertext. Usually the only non-linear part of a block cipher is the substition layer made of one or several Sboxes. The idea is to approximate these boxes with a linear expression of the form:

$$\bigoplus_{i=0}^{n} X_i \oplus \bigoplus_{i=0}^{m} Y_i = 0$$

Where X_i and Y_i respectively are the inputs and outputs of the Sbox. For a n-bit input, m-bit output Sbox, there are $(2^n - 1) \times (2^m - 1)$ possible linear approximations. The attack then consists in investigating all possible linear approximations and the probabilities that these approximations hold. As Sboxes have 2^n possible inputs, if x is the number of times a linear approximation holds, the resulting probability is computed by $p = x/2^n$ and the corresponding bias is defined as $\epsilon = p - 1/2$. Once the approximation of the Sboxes found, the goal is to combine them so that a final approximation of the cipher only involves plaintext bits, ciphertext bits and key bits.
Linear cryptanalysis requires a $r - 1$ rounds linear approximation of a cipher if the cipher iterates in r rounds. Then only a key guess on some bits of the key are needed to expand the linear approximation to r rounds.
Matsui showed that the number of known plaintexts required in the attack is proportional to ϵ^{-2}. The probability of success of the attack increases with the number of plaintext considered.

Extensions of differential and linear cryptanalysis Since the publication of these two techniques several variants and improvements have been brought by the cryptographic community:

- Differential-linear cryptanalysis: a chosen plaintext attack where the linear cryptanalysis is used to provide a differential characteristic. The goal is to reduce the amount of plaintexts required to mount the attack.

- Non-linear cryptanalysis: an improvement of linear cryptanalysis that decreases the number of plaintexts required. The goal is to find non-linear approximations of Sboxes, which give better probabilities for the key search.
- Chosen-plaintext linear cryptanalysis: an improvement of linear cryptanalysis based on the choice of a particular plaintext to recude the number of active Sboxes.
- Partial or truncated differential: it is possible to build truncated differentials with significant probabilities for some ciphers that are secure against differential cryptanalysis. These differentials predict that some parts of the output difference is 0, while other parts are non-0.
- Higher order differentials: an improvement of differential cryptanalysis that studies the propagation of a set of differences between a larger set of plaintexts. The goal is to calculate higher-order derivatives of the round function to access better probabilities for key search.

4.8 Side-channel analysis

4.8.1 Definition

A side-channel analysis is an attack based on the information that can be gathered from the implementation and the processing of sensitive data on a device. Any device produces emanations that can be measured to retrieve the sensitive data. The means can be a study of the time, power consumption, electromagnetic radiation, light emission or even sound. This attack can be mounted on computers, smart cards, FPGAs and any device that will compute a cryptographic operation.
Attacks can be divided into two categories :

- Passive attacks that will consist only in observing the target and any information it can leak. The attacker can possibly modify the target to execute a specific behavior to observe or to repeat some operation several hundreds or thousands of times.
- Active attacks that will consist in manipulating the target or its environment outside of its normal behavior. Some examples are fault injections, clock glitching or changing the program flow.

4.8.2 Initial discovery.

The U.S. Army and Navy were using secure teletypewriter communications during WWII, the Bell Telephone mixing device 131-B2. Friedman claims that in 1943, engineers from Bell Telephone were testing the 131-B2 when they realized that each time the machine was actioned, a spike appeared on an oscilloscope in a remote place in the lab. Further explorations revealed that it was possible to recover the plaintext being encrypted by the machine using the remote oscilloscope. The engineers set up a demonstration for military officials to prove that this effect was exploitable in the field. An experiment was realized from a building about 25 meters away from a Signal Corps' cryptocenter. After 1 hour of data capture and 3 hours of analysis, 75% of the plaintext data that was processed was recovered.

Bell Labs was appointed to study the causes of these leakages and how to limit or stop them. They identified three different sources:

- Radiation through space and magnetic fields. The protection suggested is shielding.
- Conducted signals on power or signal lines. The countermeasure hinted is filtering on the lines.
- Space-radiated or conducted signals. The fix proposed is masking. In this context, masking consists in creating a lot of ambient electrical noise to override, or hide the signals.

Because of the limited time and the difficulty to implement these solutions, the US being at war and having cryptoprocessors used in many different war zones abroad, the idea was left unused. At the end of the war, most of the machines and papers describing their operations were destroyed. A declassified document from the NSA [124] explains that the first work undertaken by the US government goes back to 1951, when CIA rediscovered the Bell Labs's findings. They realized that any time a machine was used to process information electrically, the various switches, contacts, relays or other components may emit radio frequency or acoustic energy. These emanations can then be used to trace back either the plaintext or the key used to encipher the messages.
Several attempts were made to try and diminish as much as possible this phenomenon by hiding the signal into a noise, or other operations, by masking the data in such a way that clear data were not directly linked to the emissions, by trying to decorrelate them as much as possible from any emission ,or by shielding or coating the various machines with metal surroundings creating a Faraday cage. All techniques used for either retrieving signals and information from cryptographic devices or protecting a device from side-channel analysis are part of a NATO and NSA certification code-named TEMPEST. This specification is available to agencies and companies that work with American or NATO secrets.

4.8.3 Compromising emanation

The NIST defines compromising emanations as "unintentional signals that, if intercepted and analyzed, would disclose the information transmitted, received, handled, or otherwise processed by telecommunications or information systems equipment". Often, useful information can be extracted from the

emanations of any electrical device in use.
These emanations can be of different types and nature. It can be electromagnetic emanations directed from the switching of transistor during an operation, acoustic energy that can be linked to a press on a keyboard or even the sound made by internal components of a computer. Information about a critical source can even be picked up using optical emanations such as the light emitted by a display or the optical signal from LED, as shown in [172].
The electromagnetic emanations of a device can be eavesdropped and saved for analysis as explained in [225] to retrieve the image of a CRT display through the EM field emitted. The historical background of electromagnetic compromising emanations, is well detailed in the PhD thesis of Markus Kuhn [167].

4.8.4 Timing attacks

Side-channel attacks were introduced by Paul Kocher at CRYPTO 1996 [165]. The idea is that an adversary can recover the secrets involved in a computation by measuring the time a cryptographic device needs to execute the algorithm. Kocher presented key-recovery attacks on implementations of RSA, Diffie-Helmann, or digital signature schemes and other algorithms.
Suppose the operation is a modular exponentiation with known base and secret exponent. Assume the modular exponentiation is implemented as a sequence of modular multiplications and squarings. The sequence of operations depends on the secret exponent. In addition, it is reasonable to assume that the modular multiplication takes different time for different inputs. Take the case of the RSA encryption: $C = M^e \pmod{n}$ with M the message, C the encrypted message, e a natural integer, prime with the Euler's totient function of two prime numbers and n an integer such as $e < n$.
The attacker collects enough measurements of the execution time for the exponentiation. Suppose the implementation handles first the least significant bit of the secret exponent e_0. The attacker can model the execution time based on the known inputs and a guess for e_0. This predictions are contrasted with the measurements, and if they match, the guess is deemed correct and the attack continues in a divide-and-conquer fashion against the next secret bit e_1 carrying over the information learned on e_0.

Quick example Let's show a quick example with a very simple attack on a PIN code verifying. Let's suppose the PIN is four digits, each between 1 and 9.

```
boolean verifyPIN (byte[] inputPIN)
{
        for (int i = 0; i < correctPIN.length; i++)
                if (inputPIN[i] != correctPIN[i])
                        return false ;
        return true ;
}
```

While an exhaustive search would need to test every 10000 possibilities, it is possible to mount an attack with only 40 tests at maximum with a timing attack. Actually, the test will stop if the first number is wrong. An attacker then just need to test 0xxx, 1xxx, ..., 9xxx and for one of the ten possibilities, the code will take longer to execute as the first *if* call will not return false. It is then sufficient to do the same with the three other digits to find the correct PIN.

4.8.5 Power analysis attacks

Power analysis attacks were introduced by Paul Kocher, Joshua Jaffe and Benjamin Jun in a technical report in 1998 [164], introducing a whole new area of research. Power analysis studies the instantaneous power consumption of a device performing cryptographic operations to find cryptographic keys. Power analysis can be easy to implement if no countermeasures are present, it does not require expensive equipment and the cost per device is low. This discovery had a huge impact on all the industry that sold cryptographic solutions.

Simple power analysis Simple power analysis (SPA) represents several analysis techniques including gathering and analyzing one or several traces, and inspecting patterns. If successful, SPA can completely break a cryptographic implementation using very few traces. It is mainly used in the

domain of public-key implementations but secret-key implementations can also be vulnerable.
SPA attacks are highly dependent on the details of the implementation which can make them difficult to apply.
A simple example with a RSA signing operation. This operation uses a public input m that is raised to a private exponent d producing the signature $m^d \pmod N$. If the exponentiation uses a naive square-and-multiply algorithm, the sequence of performed square and multiply operations reveals the secret exponent d. This sequence of operations may be distinguishable in the power trace, and thus be susceptible to a SPA attack. In the classical consumption model, each square operation will appear as a 0 and each multiply as a 1, hence leaking the secret exponent.

Differential Power Analysis Differential power analysis (DPA) analyzes several hundreds of traces (up to millions) to recover cryptographic secrets. It exploits the fact that the power consumption is correlated with the data value being handled by the device, the power dissipation to manipulate one bit to 1 is different from the power dissipation to manipulate it to 0.
It first needs to model the power consumption when the device is handling a concrete intermediate value. This modeling involves a guess of a part of the key (usually a byte). Then, the different key-dependent models are compared against measurements. The key guess that leads to a better fit is deemed as the correct key candidate.
DPA has many attractive properties: it is resilient to noise (the success rate can be amplified by using more traces) and does not require extensive computational resources. There are many variations on SPA and DPA. A very popular variant is Correlation Power Analysis [73].
Templates attacks were introduced at CHES 2002 by Suresh Chari, Josyula R. Rao and Pankaj Rohatgi [84]. These attacks are variants of DPA where the adversary can perform a prior profiling with a study device under his control. Another variant is collision attacks: internal collisions are detected during the executions to recover key material. The idea was presented at FSE 2013 by Kai Schramm, Thomas Wollinger and Christof Paar [213].

4.8.6 Fault attacks

The first paper on faults attacks was published by Dan Boneh, Richard A. DeMillo and Richard J. Lipton in [69]. They presented a theoretical model for breaking various cryptographic schemes such as the Fiat-Shamir protocol or certain RSA implementations, by taking advantage of random hardware faults. This attack was described only to target public key cryptosystems. Biham and Shamir then extended this attack to some secret key cryptosystems such as DES [61]. They called this attack Differential Fault Analysis. In this paper, they described how it can be feasible to access not only cryptographic keys of known ciphers but also the key stored in tamper-resistant cryptographic devices even when nothing is known about the structure and operation of the cryptosystem. Fault attacks can be of various type, depending on the platform that is attacked. The most used attacks are power or clock glitching that can be used to bypass some instruction (for example a checking operation that happens on a given clock cycle) or fault injections by laser, that can be used to flipping bits on SRAM to create faulty bytes on a cryptographic computation. In 2003, Dusart et al. described a DFA on AES [190] needing only eight successful faults injected to recover a 128 bits key in just a few seconds.

4.8.7 Countermeasures

In cryptography it is never said that there is a way to create a solution that is 100% secure, but the goal is to make an attack as difficult to realize as possible. It is the same for side-channel countermeasures. There are several possibilities to make an implementation less prone to side-channel analysis, we can cite for example:

- Solutions at algorithm level, such as masking.
- Solutions at protocol level, if an attack is known to be able to retrieve the key in a thousand computations, one can make the key change every 500 computations.
- Solutions at implementation level, unroll an implementation, make it less iterative to make power analysis more difficult.

- Hiding, by generating a noise or adding dummy operations to hide the timing or the power usage of a single operation.
- Solutions at mechanical level, such as shielding with a metal coating, use mesh cover to detect and/ or prevent intrusion.

In this work, we will only present countermeasure based on masking because it is a technique that can be used on every algorithm, in any application, contrary to the other solutions.

4.8.8 Masking

Masking is a countermeasure against power analysis attacks. The point is to decorelate the values processed by the device from the intermediate values of a cryptographic algorithm.
Each bit of the original computation is split into several shares, so that each of the shares considered individually does not reveal any information on the original bit. Every computation is carried out by handling only the different shares; the original bit is only reconstructed at the end of the computation.

The first proposal At the same conference where SPA and DPA were introduced, a team from the IBM Thomas J. Watson research center published a sound countermeasure against power analysis attacks [83]. They proposed a masked encoding of each original bit b into k shares

$$b \oplus r_1, r_2, ..., r_{k-1}, r_1 \oplus ... \oplus r_{k-1}$$

where each r_i is a random bit.

In the meantime, they provide a formal security proof arguing that this probabilistic encoding makes power analysis attacks more difficult in terms of traces needed to distinguish between two possible values of an unshared bit. More precisely, the main result is that an adversary willing to distinguish between two leakage distributions of masked bits $b = 0$ and $b = 1$ needs exponentially more traces as the number of shares is increased. They proved that the amount of samples required to distinguish the two distributions (corresponding to the two possible values for the unshared bit) grows as σ^{2k}, where σ is the noise level and k the masking order.

The duplication method A similar concept of masking was introduced at CHES 1999 by Goubin and Patarin with the name "the duplication method" [132], hence, masking is often credited to both the IBM team and Goubin and Patarin. Goubin and Patarin give practical constructions for computing DES and RSA. For DES they give essentially a table-based approach to compute the non-linear functions, and provide some optimizations to reduce RAM usage.

Threshold implementation Threshold implementation were introduced by Nikova et al. in [188] as an alternative to masking to secure implementation in the presence of glitches.
A TI is based on multi-party computation and secret sharing.
The authors identified two key properties: non-completeness (any share of the masked function must be independent of at least one input share) and uniformity (the masked function uses a uniform sharing of the input and transforms this input into a uniform sharing of the output), which together ensure the provable security of TI against first-order DPA in the presence of glitches on any hardware as long as the independent leakage assumption of masking holds. Two other security notions were then added to complete the security proof of threshold implementations : correctness (the masked function should compute a masked representation of the correct unmasked output) and non-interference.
The concept was extended to higher-order security by Bilgin et al. [63].
Threshold implementation is provably secure against any order DPA with $k + 1$ shares needed in input for a k order DPA resistance. Several implementations have been conducted to make those implementations efficient on different cryptographic components, such as SBOX [62].
For an operation S from $3n$ bits to $3n$ bits, that takes as input a vector (x, y, z) and produces (a, b, c)

as outputs, a threshold implementation at order k is such as:

$$\begin{cases} S(x_1, y_1, z_1) &= (a_1, b_1, c_1) \\ S(x_2, y_2, z_2) &= (a_2, b_2, c_2) \\ \dots \\ S(x_k, y_k, z_k) &= (a_k, b_k, c_k) \end{cases}$$

and

$$\begin{cases} (x_1, y_1, z_1) \oplus (x_2, y_2, z_2) \oplus \dots \oplus (x_k, y_k, z_k) &= (x, y, z) \\ (a_1, b_1, c_1) \oplus (a_2, b_2, c_2) \oplus \dots \oplus (a_k, b_k, c_k) &= (a, b, c) \end{cases}$$

Masking comes with a cost Given a masking order k, the principle of masking is to split all the sensitive data manipulated by the original circuit in $k+1$ shares in the masked circuit, and to perform the computations on those shares only. In the masked circuit, each wire of the original circuit is replaced by a set of wires, and each gate is replaced with operations on the shares equivalent to the operation of the gate in the original circuit. This protection makes attacks more difficult: to recover the value of a sensitive variable, the adversary has to know the value of all its shares.
Various types of masking schemes have been considered in the literature including additive/boolean masking [89] ($b_1 = b \oplus m, b_2 = m$), multiplicative masking [131] ($b_1 = b * m, b_2 = m$) , affine masking [231] ($b_1 = b * m_1 \oplus m_2, b_2 = m_1, b_3 = m_2$), Inner Product masking [36]. All these proposals come with significant overheads in execution time and randomness consumption.
Masking techniques are one very important point to take into consideration when implementing a cryptographic algorithm on a given device. A device carrying sensitive data needs protection against side-channel analysis and as for cryptographic ciphering solutions, lightweight masking solution are under the scope of intensive research to secure devices with as little cost overhead and computation time overhead as possible.

4.9 Hardware implementation of cryptographic algorithms

4.9.1 Choosing the implementation method - Software vs hardware cryptography

Implementing cryptography can be as tricky as designing secure schemes. Moreover, a poor implementation can make a provably secure design completely breakable. Therefore, a focus has to be made on implementation matters and first of all on which platforms our cryptographic schemes have to be implemented.
Modern cryptographic algorithms can be implemented using either dedicated cryptographic hardware or software running on general-purpose hardware. Each has its own pros and cons which we will try to describe below.
When it comes to implementation aspects, several factors have to be taken in consideration. One main drawback of having dedicated and accurate answer to each possible security issue a system may face is that it comes with a computational cost or more exactly, costs. Cost is measured in terms of execution time, memory use, power consumption, and die surface. All the history of security company shows the constant struggle between the trades-offs of lowest implementation cost versus the required level of security. We can cite as example the RSA where keys lengths correspond to the level of security; the longer the key is, the stronger the protection, but also the longer the key is, the higher the complexity, i.e. the computation is longer and the required resources are larger. A prudent risk assessment estimates which key length is sufficient for the task and, therefore, what is the minimum level of complexity required for the implementation.

Software-Based Encryption is the process of keeping data safe using software. Software is usually installed in host computers that encrypt and decrypt data. It usually shares processing resources with all other programs or processes on the system that might have an impact on performance of all other functions of the system.
It is usually less costly to develop software solutions as no specific hardware is required. The fundamental cryptographic needs for these less complex secure applications are hash functions and digital signature verification that can be realized with software implementation. It is simpler than embedded protection and does not impact the die size. Moreover, it does not require a specialized hardware block that may not be used by every customer, and, finally, does not raise system costs.
Software encryption programs are more prevalent than hardware solutions as they can be used to protect all devices within an organization. These solutions can be cost effective as well as easy to use, upgrade and update. Software encryption is readily available for all major operating systems and can protect data at rest, in transit, and stored on different devices. Software-based encryption often includes additional security features that complement encryption, which cannot come directly from the hardware.
Moreover, software solutions allow flexibility in the choice of the algorithm as adding a new algorithm to the platform is as simple as a software update.
The protection granted by these solutions, however, is as strong as the level of security of the operating system of the device. A security flaw in the OS can easily compromise the security provided by the encryption code. Encryption software can also be complicated to configure for advanced use and, potentially, could be turned off by users. Performance degradation is a notable problem with this type of encryption. The security level of a software-based cryptographic module is upper-bounded by the security level of the mechanism that protects the secrecy and integrity of the memory space it uses.

Hardware-Based Encryption is a process of keeping data safe using dedicated and separate processor. It is more cost-effective for larger companies because it does not require any additional software installation. It provides greater throughput capacity and speed in large-scale environment.
A hardware-based solution is most advisable for sensitive data protection on portable devices such as laptops or a USB flash drives. It is also effective when protecting data at rest. Drives containing sensitive data such as banking applications, private data or government fields are better protected through hardware keys that can be effective even if the equipment is stolen or installed in another computer.
Hardware-based encryption offers stronger resilience against common attacks. In general, an attacker will not be able to perform brute-force attacks to a hardware-encrypted system because the crypto module will shut down the system and possibly compromise data after a certain number of attempts, while in software-based solutions, hackers might be able to locate and possibly reset the counters or even copy the encrypted file to different systems for parallel cracking attempts. It also includes faster algorithm processing, tamper-proof or tamper-resistant key storage, and protection against unauthorized code.
On the other hand, hardware implementation is relatively fixed and takes a relatively long time to implement. Even a simple implementation of an algorithm on FPGA, is more difficult to design, code, and test than the same effort in software.
Moreover, hardware solutions might be inapplicable due to their cost. A hardware encryption is tied to a particular device and updates can only happen through device substitution.

Implementation platforms There are typically four different settings (or platforms) that can be used to implement and run cryptographic applications.

- The General-purpose processor (Central Processing Unit (CPU) or microcontrollers for example). Usually a classic desktop or laptop CPU, it can also be part of an embedded device. Most of the time it carries only a few computation cores (less than 20) that can be used very fast and it can execute arbitrary instructions (in arbitrary order) from its assembler language. Encryption algorithms running on CPUs are most likely software-implemented because the algorithms (i.e. the instructions) are merely information given to the CPU for execution. CPUs are best at running complex, linear algorithms.

- The Graphics Processing Unit (GPU). It can be used as desktop or laptop graphics processor or a supercomputing computation accelerator. It is characterized by its high number of cores, the low speed of each core and the limited instruction set. GPUs are made to run simple algorithms that are massively parallel. Much like CPUs, their algorithms are implemented in software.

- The Field Programmable Gate Array (FPGA). It is usually found in small embedded devices, running specialized algorithms. Its hardware can be configured after manufacturing, at the expense of lower speeds of operation than with ASICs. With FPGAs you change the hardware layout of your integrated circuit to run your algorithm. Hence algorithms run by FPGAs are said to be hardware implemented, because in its current state, the hardware can run only this exact algorithm, nothing else.

- The Application Specific Integrated Circuit (ASIC).This is an integrated circuit manufactured to run a set of function, created by producing a specific set of transistors. Once the ASIC is created, it cannot be modified; an ASIC is manufactured for a purpose and can only achieve this purpose. ASICs provide high speed for this algorithm and are used in speed or throughput critical applications. Hardware Security Modules (HSMs) are a good example of application using ASICs to accelerate the execution of cryptographic operations (such as the AES or SHA hashes). Crypto processors commonly are simple processors with additional crypto-specific ASICs.

As for the security of each platform, the tendency is that if going down this list, the security will increase. CPUs are usually occupied by many different processes (including the OS), but may be prone to side-channel attacks, GPUs are commonly not used for cryptographic applications, FPGAs should provide more security than CPUs if implemented properly and provide better raw side-channel security and ASICs have the same benefit, while allowing to go further into hardware exploration.
The speed increases similarly going down the list. CPUs must be capable to do many different operations and cannot be too much optimized in one direction, same goes for GPUs although they have

much more computation power if needed. FPGAs are faster because of their inherent parallelism, configurable logic cells and interconnections. ASICs are the most advantageous because besides the inherent parallelism, their architecture can be highly optimized, while being dedicated to the selected algorithm and using fast hardwired interconnections.
The price has a similar order. CPUs are easy to obtain, cheap to program and easy to implement and run. GPUs are also quite easy to obtain, a bit more expensive to effectively program and can be get to run the code efficiently. FPGAs are more expensive by themselves and require device-specific design tools, especially for placement and routing, timing estimation, and configuration, hence more time and expertise and thus money. ASICs are specific due to their very high non-recurring engineering (NRE) cost and long design cycles, but relatively low production cost.
For the process of designing and implementing an algorithm, it will often first be done in software as it may be a lot easier to test. Once the algorithm is defined, the question to answer is what is this algorithm going to be used for, which performances are important, which comes first between efficiency and flexibility? Moreover, how many devices using this algorithm will there be and do these devices need to be protected through hardware obfuscation? Once these questions answered, the choice has to be weighted between the costs of implementing in hardware versus the benefits of speed and market control.
In our application, the choice was made towards FPGA because of required performances, and the number of devices will not be high enough to make ASICs profitable.

4.9.2 FPGA

FPGA stands for Field Programmable Gate Array. An FPGA is an integrated circuit that is designed to be configured after manufacturing, contrary to an ASIC. A classical FPGA can be reconfigured at will.
An FPGA is a component that can be seen as a big grid of digital components such as gates, look-up tables and flip-flops that can be connected through wires. The description code used to program the FPGA will be translated into physical connections with wires to perform the functions needed. This translation is realized by the vendor programming tool. Similarly to ASICs, FPGAs are very good at performing a large number of operations in parallel. Hence, they are used in high-speed, high-performance tasks such as image processing, telecommunications, digital signal processing or high-frequency stock market trading.

Several technologies exist for configuring FPGA

- Static RAM: the connections are made by turning a transistor on. This technology relies on a standard memory cell loaded at initialization, it is based on the CMOS technology. Its main advantage is that it allows a very quick reconfiguration of the circuit, at the cost of an important surface needed.

- EEPROM: Electrically Erasable Programmable Read-Only Memory technology. Individual bits can be re-programmed or erased electrically (UV light is not needed).

- Flash EPROM technology. Individual bits can be re-programmed or erased electrically, more quickly than EEPROM using address sectors. Some but not all flash devices can be in-system programmed. Usually, a flash cell is smaller than an equivalent EEPROM cell and is therefore less expensive to manufacture.

- Anti-fuse: Once programmed, the FPGA is locked: a high voltage is applied across the terminals, creating a low-resistance link. This technology is less expensive than SRAM, achieves higher speed and needs less surface. However, the FPGA can only be programmed once, and any error in the configuration bitstream or during programming means that the FPGA becomes useless.

The anti-fuse technology is the most efficient in terms of area occupation but cannot be reprogrammed. EEPROM and Flash EPROM-based FPGAs are the second most area efficient, namely because they do not need additional configuration memory, but they necessitate additional hardware for reprogramming (if the reprogramming is needed) and the reprogramming speed is quite low compared to SRAM technologies.

Major manufacturers Two industry rivals Xilinx (now AMD) and Altera (now an Intel subsidiary) are the FPGA market leaders, controlling nearly 90 percent of the market. Each provides a proprietary electronic design automation software for Windows and Linux (ISE/Vivado and Quartus) which enables engineers to design, analyze, simulate, and synthesize their designs.
Other manufacturers include:

- Microchip:
  - Microsemi (previously Actel), producing antifuse, flash-based, mixed-signal FPGAs; acquired by Microchip in 2018
  - Atmel, a second source of Altera-compatible devices; acquired by Microchip in 2016
- Lattice Semiconductor, which manufactures low-power SRAM-based FPGAs featuring integrated configuration flash, instant-on and live reconfiguration
- SiliconBlue Technologies,
- QuickLogic,
- Achronix.

Internal structure of an FPGA FPGA circuits are composed of a matrix of configurable logic block, surrounded by configurable Input/Output (I/Os) blocks. The whole design is interconnected by a configurable interconnection network. The interconnections fill approximately 80% of the whole surface. The internal structure of FPGAs differs depending on the manufacturers. Commercially, four major structures can be found :

- Symmetrical Array
- Row-based
- Sea-of-Gates
- Hierarchical PLD

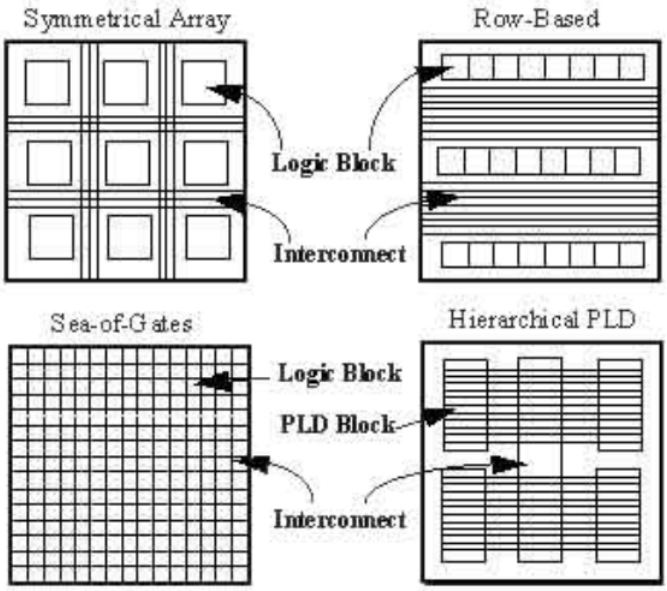


Figure 4.5: The different FPGA structures

Xilinx for example uses symmetrical array while Microsemi FPGAs are Row-based.

Resources of an FPGA FPGA resources are used to perform logic functions. Xilinx FPGA resources are grouped in slices to create configurable logic blocks. A slice contains a set number of LUTs, flip-flops and multiplexers. A LUT is a collection of logic gates hard-wired on the FPGA. It is a memory representing (containing) a truth table of the implemented combinatorial function with up to 6 inputs and provide a fast way to retrieve the output of a logic operation. A flip-flop is a circuit featuring two stable states. It stores a single bit value. A multiplexer, (also called a MUX), is a circuit that selects between two or more inputs and outputs the selected input.
LUTs are grouped to slices (Xilinx) or logic array blocks - LABS (Altera). Different FPGA families implement slices, LABs and LUTs differently. For example, a slice on a Virtex-II FPGA has two LUTs and two flip-flops but a slice on a Virtex-5 FPGA has four LUTs and four flip-flops. In addition, the number of LUT inputs, commonly two to six, depends on the FPGA family.

- Area occupation When comparing ASICs implementation, the area is expressed in Gate Equivalents (GEs). For FPGA, depending on the technology, the area occupation is given in slices or logic cells (LCs), LUTs, FFs or Block RAMs.
  - Slice
  - LUT
  - FF
  - Register
  - Block RAM
- Frequency
- Throughput
- Throughput/area ratio
- Energy
- Power

A LUT (Look-Up Table) is a small asynchronous SRAM that is used to implement combinational logic, while FF (Flip-Flop) is a single-bit memory cell used to hold state. LUTs are usually read-only and their content can only be changed during FPGA configuration. But in Xilinx FPGAs usually half of the LUTs can actually be written to, so they can be used to implement many small RAMs (so-called "distributed RAM").
Flip-Flops are components that can have two stable states, representing one bit. A Flip-Flop is the smallest storage resource on FPGAs. Each one is a binary register used to store the logical state between each clock cycle.
A flip-flop stores a single bit of data; one of its two states represents a "one" and the other represents a "zero". Such data storage can be used for storage of state, and such a circuit is described as sequential logic in electronics. When used in a finite-state machine, the output and the next state depend not only on the current input, but also on the current state. It can also be used for counting pulses and synchronizing variably-timed input signals to some reference timing signal.
Multiplexers (that can be viewed as data selectors) are devices that select between several input signals and forward the selected input to a single output line. The selection is directed by a separate set of digital inputs called the select lines. A multiplexer of 2^n inputs has n select lines to select which input line to output.
A register is a group of flip-flops that stores a bit pattern. A register on the FPGA has a clock, input data, output data, and an enable signal port. At every clock cycle, the input data is latched, stored internally, and the output data is updated to match the internally stored data. Registers can be used to perform functions such as:

- Holding state between iterations of a loop
- I/O synchronization
- Handshaking data between clock domains

- Pipelining

Block RAM, or block memory, is RAM that is embedded throughout the FPGA for storing data. BRAM memory can be used for the following tasks :

- Transfer data between different clock domains using local FIFOs (First In First Out structures).
- Transfer data between the FPGA and a host processor using a Direct Memory Access (DMA) FIFO.
- Transfer data between two FPGAs using a peer-to-peer FIFO.
- Store huge loads of data on FPGAs. This is more efficient than using RAM based on LUT tables.

Designing an FPGA-based application The digital circuitry which should be implemented in the FPGA can be described using schematic diagrams or hardware description languages (HDL). The HDL code is fundamentally different from a software code - it describes the hardware architecture instead of the algorithm. The HDL code is compiled and transformed to netlists containing logic elements available in the selected FPGA family: LUTs, FFs, RAMs, DSP components and other dedicated blocks. The netlist is mapped to the device during the placement and routing process.
Logic blocks can be configured to perform complex combinational functions, or simple logic gates such as AND and XOR. In most FPGAs, logic blocks also include memory elements, which may be simple flip-flops or more complete blocks of memory.
Many FPGAs can be reconfigured online to implement temporarily different logic functions, allowing flexible reconfigurable computing as performed in computer software.

4.9.3 Implementation methods

The implementation method used by the designer of a cryptographic algorithm on any kind of platform depends largely on the implementation methodology, so it is of course the case for FPGA. Many different strategies exist to optimize different resources and the implementation decisions have a significant impact on the area occupation and performance of the targeted architectures. Let's take the example of the AES [1]. This algorithm can be divided into two parts: the key expansion part and the cipher module. The algorithm is also divided into different iterating rounds, 10 for a 128-bit key, 12 for a 192-bit key and 14 for a 256-bit key. Each round iterates the same four operations in the same order: SubByte, ShiftRows, MixColumns and AddRoundKey, except for the last round which does not compute the MixColumns operation. A naive and straightforward implementation of the AES would be to implement each operation linearly, one after the other, in the same way a C code could be implemented naively. The key schedule would be executed first, computing each subkey for each round, and then the cipher part using these subkeys. This is usually not the preferred methodology as there exist different possibilities to improve the implementation performances and occupation.

Unrolling the implementation The cipher module essentially is an iterative looping structure; the first rounds are identical but the last one only differs in one operation. Classic loop optimization technique like loop unrolling can then be applied to optimize the implementation in terms of speed. Loop unrolling is a technique that replaces a looping structure with N copies of that looping body, hence reducing the total loop iterations by N times (i.e. inter-round optimization). Loop unrolling of the cipher module can be combined with key expansion routine to achieve fully parallel implementation of AES algorithm. Key expansion routine can be split into smaller key expansion modules that would be placed along unrolled round operations of cipher module (online key expansion). Unrolling the cipher module combined with split key expansion modules result in fully parallel implementation of the AES algorithm on FPGA. This parallel implementation leads to an improved throughput of AES algorithm compared to a straightforward implementation. However, unrolling the code increases the area occupation of the resulting algorithm.

Storing memory in BRAM Another important way of optimizing the performances of an implementation is to use or not the BRAM blocks present in most of current FPGAs to store part of the code. Using BRAM blocks to store data is an efficient way of minimizing the area occupation but it increases the delay, as the FPGAs takes more time to access data stored in BRAM than to access it while it is stored in the logic cells (or slices). The work realized by Umer Farooq and M. Faisal Aslam provides comparison of five different implementation of the AES storing the SBOX in the logic cells or in BRAM, and using rolled or unrolled implementation [119]. Each implementation choice has an impact on the area occupation, the delay and the maximum throughput achieved by the implementation.

Hardware API for Lightweight Cryptography The first public competition for cryptographic standards was the NIST call for the Advanced Encryption Standard [1]. During this competition, the main criteria of competition were security and software performances. The hardware performances only came into consideration close to the end of the standardization process. The same happened during the SHA-3 competition and there are still only a few referenced papers on hardware performances of the candidates. The real change came with the CAESAR competition, when a standard hardware API (Application Programming Interface) was developed and validated by the competition committee [141]. Hardware implementations were required for all submissions from round 3, with the development team helping candidates to use and develop their solutions. This API was not mandatory but it helped to define new guidelines for comparison at the second half of the competition. The implementation and linked results of each candidate can be found here [3].
For the NIST Lightweight Cryptography Standardization process, no hardware implementation was mandatory but before the announcement of the round two, some candidates were found with a hardware implementation compliant with the CAESAR Hardware API. To fairly implement and compare the different candidates of the NIST standardization process for Lightweight Cryptography (LWC), a Framework for Benchmarking of Hardware Implementations was introduced and presented by Kaps et al. in [156] and [155]. Based on the framework developed for the CAESAR competition [2], it gives the developers a guideline to implement the different ciphers. This requires a few modifications to adapt the implementations based on the CAESAR API and the overhead from this API is well explained in [157]. With the beginning of the round 2 in September 2019, more implementations, compliant with this API, have been created and can be used as a fair basis for benchmarking hardware performances [185].

Specifications In order to speed up the development process, both CAESAR and LWC API feature a development package and an Implementer's Guide. Both designs use the same modules: the Preprocessor, FIFO, Postprocessor and the cryptographic core. The input data are split into two parts: the public data (pdi) and the secret data (sdi) which are both received by the Preprocessor. After removing the header information, the data are passed to the cryptographic core which will implement the chosen cryptographic primitive (designers only have to modify this part to implement their solution). The Postprocessor will then get the data output by the cryptographic core and add the API specific header data before sending it to the ouput port (do). In addition to some changes, the LWC package also fully supports hash algorithms, which will not be considered in this manuscript as not all candidates specify one.

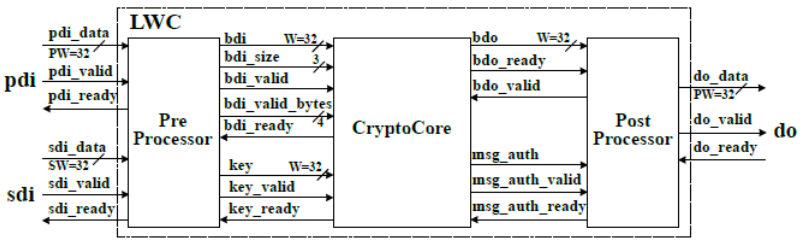


Figure 4.6: The LWC API

Implementation process In order to fairly compare all candidates, one method must be chosen and kept during the development process of each algorithm. We decided to follow the method chosen and explained in [192]. Each algorithm is developed following a basic-iterative construction, meaning that each step or round of an algorithm will be executed in one clock cycle. If the permutation of an algorithm is made of steps and rounds, such as ACE [13], one permutation will be executed in number of steps * number of rounds clock cycles (in this example $16 * 8 = 128$ clock cycles). Moreover, all implementations presented in 8 are fully compliant with the LWC Hardware API.

4.9.4 Comparing the implementation results

Type of FPGA based on applications The FPGA can be classified into three types based on their application: low-end, mid-range and high-end FPGAs.
Low-end FPGAs are designed for low power consumption, low logic density and low complexity per chip. In the works presenting algorithms or applications targeting lightweight implementations, the results are often targeting the Spartan family from Xilinx or the Intel Cyclone families.
Mid-range FPGAs consist in a trade-off between low-end and high-end applications and are developed as a balance between performance and cost. FPGA results when no particular application is targeted are sometimes presented on Artix FPGAs from Xilinx.
High-end FPGAs are developed for logic density and high performance, where the most important resource is either throughput or throughput on area ratio. Comparisons targeting high performance applications are often studied using the Virtex FPGAs from Xilinx, as it is the case for the LWC standardization process or the CAESAR competition.
When comparing different algorithms, schemes or implementation methods, several different resources are to look upon. Depending on the targeted application, the most important ones can differ. For example, lightweight applications target a low area occupation and a good energy and power efficiency (lightweight components only have a few gates dedicated to security and have little or no place for battery and need solutions with a reduced energy consumption).

Chapter 5

Block ciphers

5.1 Introduction

Block ciphers are arguably one of the best understood primitives in the field of symmetric cryptography, if not the best. Over the last decades, solid theoretical foundations have been developed and several design principles have been established to allow cryptographers to construct block ciphers with solid basis against cryptanalysis.
From the first standard in 1976 [186], many improvements have been brought, not only in term of creativity, or security, but also performances. The current challenges in designing new block ciphers tend to follow the development of new products which all need security and require as compact hardware as possible.
This brought the need for more compact design, improving performances, while at the same time taking no compromises regarding to security. Research in block ciphers is also oriented towards development of new, more efficient attacks to prove the robustness of existing ciphers and to propose new cipher designs or design methodologies.

In 1883 Auguste Kerckhoffs published two papers in La Cryptographie Militaire [159] in which he stated six axioms of cryptography. In modern words, these axioms can be summarized as:

- The system must be mathematically and physically indecipherable
- It must not rely on secret algorithms or parameters
- It must be easy and/or efficient to transmit the key and change it
- It must be applicable to telecommunications
- It must allow lightweight implementations
- It must be user friendly and its use must not require technical prowess.

The second axiom, now known as Kerckhoffs' Principle states "Il faut qu'il n'exige pas le secret, et qu'il puisse sans inconvénient tomber entre les mains de l'ennemi" which, in English, means that the method should not be required to be secret, and it can fall into the hands of the enemy without inconvenience.

This implies that the security shall only rely on the secrecy of the key. This relates to Claude Shannon's maxim: "the enemy knows the system." This may be the most important design principle for any cipher: avoid secret designs or design principles. Indeed, scrutiny leads to better ciphers and understanding of possible attacks, but obscurity may give a false sense of security, leading to design mistakes or shortcuts. Thus, the design of cryptographic primitives shall not only be described, but also explained and detailed in its every choices.

5.2 Definitions and design principles

A block cipher is an algorithm that applies a permutation to the set of all blocks of a fixed length, selected by a key. The permutation is called encryption and converts a plaintext into a ciphertext.

The inverse permutation is called decryption and converts a ciphertext into a plaintext using the same key used for encryption. Usually, the cost to compute encryption and decryption are identical or fairly similar.
A cipher is classically deemed secure if it is not feasible to determine the key, even with a large set of plaintext/ciphertext pairs available.

Modern concepts can be defined to prove the security of a block cipher:
Indistinguishability: two systems are said to be indistinguishable if no efficient algorithm is able to decide which one of the two systems it is interacting with. A block cipher $E(K, \cdot)$ and its inverse $D(K, \cdot)$ with an uniformly random key K are secure if E and D are indistinguishable from a truly random permutation π and its inverse π^{-1}.
Indifferentiability: two systems are said to be indifferentiable if and only if the security of any cryptosystem using one as a component is not affected when substituted by the other one. In other words, a block cipher is secure if it is indifferentiable from an ideal one.
Any attack against a block cipher should not run faster than brute force on the key space. Moreover, a block cipher should not be approximable by any easily computable function of the input with a non negligible bias, implying that a block cipher shall be non linear.
Another desirable property is the avalanche effect, which means that any change in either the key or the plaintext should induce an important change in the ciphertext. This was formalized by Webster and Tavares as the Strict Avalanche Criterion (SAC) [229]: "Complementing a single bit in the input or in the key should change any bit in the output with probability 1/2, for any input or key bit and for any output bit."
Claude Shannon first formalized the idea of product cipher by defining it as a sequence of rounds. The input to the first round is the plaintext, then the output of each round is used as input for the next one. Finally, the output of the last round is the ciphertext. Each round is a simple cipher itself, using a round key, derived from the main one. Block ciphers then need a third operation, namely the key schedule (or key expansion) that will generate round keys from the encryption key.
In order to simplify the construction, Shannon suggested to build a product cipher by repeating the same steps over and over until the desired robustness is achieved. Intuitively, increasing the number of rounds improves the security of a cipher at the expense of speed, while reducing the number of rounds can improve performance at the expense of security. This trade-off is often considered during design of new block ciphers.

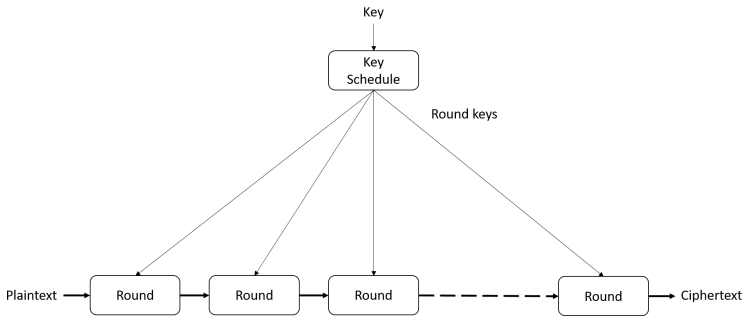


Figure 5.1: An iterated product block cipher

The operations used to build each round shall of course provide both confusion and diffusion.

5.3 Substitution-Permutation Networks

To achieve both confusion and diffusion, a classical way to operate is to use a substitution function or substitution layer that will provide confusion and a diffusion layer for diffusion. The separation of these two operations allows to construct the rounds from simpler operations that can be more easily mathematically analyzed.

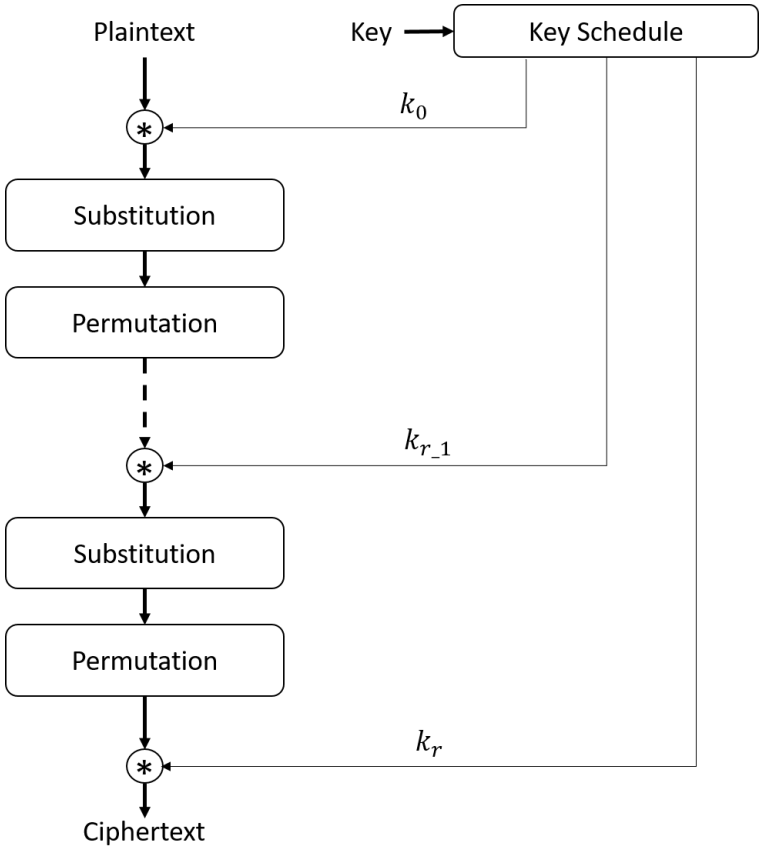


Figure 5.2: Substitution-Permutation Network

The substitution layer can be represented as a table mapping the state value to another value, either in one block or by dividing the state into several words. This function is sometimes called a Sbox. The term Sbox can also be applied to a simple non-linear operation that modifies just a part of the state. In this case, the substitution layer consists of several applications in parallel of Sboxes to the whole or part of the state.
The permutation layer or diffusion layer is sometimes called a Pbox. It can be realized by a fixed permutation of the bits of the state, or more general invertible linear transformation of the state as a vector.
A combination of these two operation types together with a key mixing constitutes a round of a Substitution-Permutation Network (SPN). Key mixing is usually a simple operation, in most cases a XOR or a modular addition.
Most of the block cipher designs and almost all those that are in use today are SPNs. Some of the designs that are often presented as belonging to a different family, such as Feistel networks or Lai-Massey designs, are in fact just particular types of SPNs. Nowadays, Feistel Networks and SPNs are treated as separate design families: when a cipher is clearly a Feistel or Lai-Massey design, it is denoted only as such and rarely referred to as an instance of a SPN, whereas SPN usually denotes a narrower class of ciphers such as bricklayer designs following the Wide Trail strategy.
Classical examples of SPN ciphers are Rijndael [97], and PRESENT [66].

5.4 Feistel scheme

In the early 1970's, IBM created a crypto-group to create a form of encryption to secure their customers data, this group was lead by Horst Feistel. They designed a cipher called Lucifer. Lucifer is a family of block ciphers whith different block and key size. The most known member of this family is the one which later was tweaked by the NSA to become the Data Encryption Standard. This member present a new design, named after its author: the Feistel scheme or Feistel Network.
This Feistel scheme is a design to construct block ciphers with almost similar encryption and decryption process. Actually, the only difference between the two operations is the order in which the subkeys are fed into the cipher.

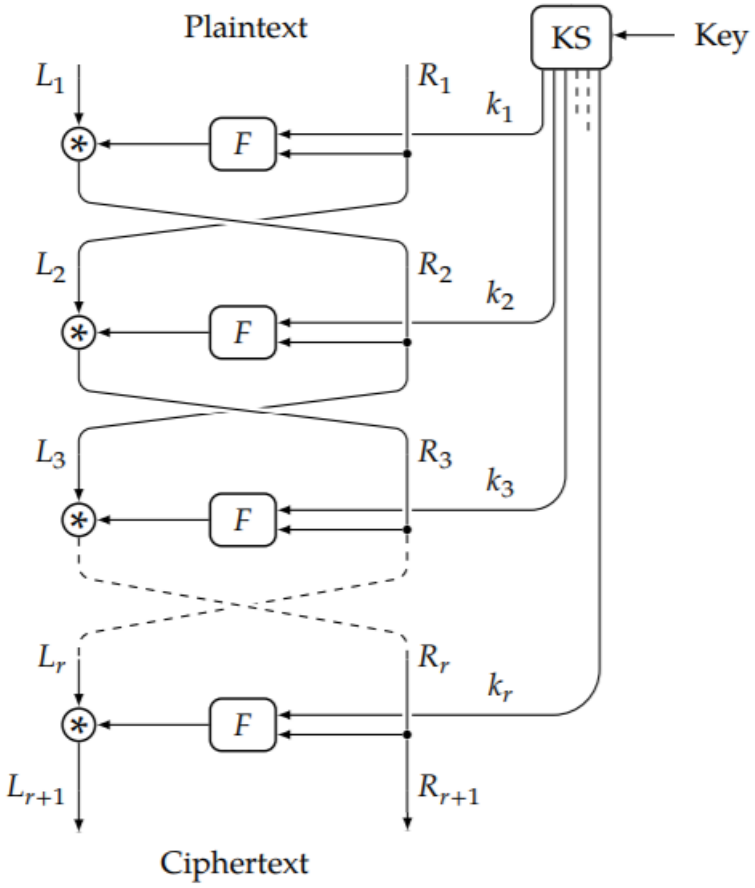


Figure 5.3: Feistel Network

The state is split into two halves, called branches. A function, called the F-function, is computed on the first half, called the source branch. The result of this operation is composed with the other half of the state, called the target, in a reversible way (the source branch remains unchanged). Finally, the two branches are swapped. For the composition of the transform of the source to the target, the XOR operation is usually used because XORing with some value is reversed by XORing again with the same value, but other composition operations may be used.
Let the state be written as $L||R$, as the concatenation of the Left and Right branches. The plaintext is the initial state $L_1||R_1$.
A round computes state $L_{i+1}||R_{i+1}$ from $L_i||R_i$ as follows:

$$\begin{cases} R_{i+1} &= L_i \circledast F(R_i, k_i) \\ L_{i+1} &= R_i \end{cases}$$

The ciphertext is the state $L_{r+1}||R_{r+1}$ after r rounds. The round keys $k_1, \ldots, k_r$ are derived from the key through a key schedule.
Feistel networks are a special case of SPNs: only half of the state is modified at each round. However, the state is a function of the whole previous state, and the permutation function is very simple. This also means that confusion and diffusion must be applied to the source branch. Since confusion and diffusion are applied only to part of the state, a higher number of rounds is necessary, usually twice as much. But these rounds can be less expensive since they have to operate only on half of the state.
Moreover, the steps are easy to revert when the composition is the XOR, since only the round keys have to be fed in the opposite order to decrypt. This means a minor augmentation of hardware surface needed to implement both encryption and decryption, compared to classical SPN.
Finally, the F-function can be a simple SPN itself, so all the theory developed for SPNs is recycled. This was the case for the development of the Lucifer family of block ciphers whose first instances were SPNs, and then the experience of the design of these SPNs was exploited to design the round function of the final Lucifer.
In fact, the F-function itself can be an existing cipher, and this construction can therefore be seen as a way to double the block size of a cipher: the original cipher is used as the F-function, so it operates on half of the state of the new cipher, and the key-schedule is extended to generate different round keys for the various instances of the original cipher. For example, DEAL [163] uses DES as the round function in this way.
One main advantage of this construction comes from the fact that the F-function does not needs to be bijective, with serious positive implications for performance, flexibility of design, and also for cryptographic theory. Michael Luby and Charles Rackoff analyzed the Feistel cipher construction [174], and proved that if the round function is a cryptographically secure pseudo-random function then three rounds are sufficient to make the block cipher a pseudo-random permutation, while four rounds are sufficient to make it a "strong" pseudo-random permutation (the cipher remains pseudo-random even to an adversary who gets oracle access to its inverse permutation). Since these results, Feistel ciphers are also called Luby-Rackoff ciphers.

Feistel types There are many types of Feistel networks and generalizations. Please refer to Figure 5.4 for the different notations used below:

1. Classic Feistel network, also called a balanced Type 1 Feistel network, the two branches have the same size. Examples of such designs are DES, Blowfish [212], Camellia [26], or Simon [44].

2. Mitsuru Matsui's variation as used in various components of the ciphers MISTY-1 and MISTY-2 [178], is here given as the "L-network," i.e. the version where the F-function is on the target branch. There is also another version, the "R-network", where the F function is on the source branch. In these constructions the F-function must be invertible.

3. Unbalanced Feistel network. The bit length of in_1 and out_0 is different from the bit length of in_0 and out_1. The output of a round can then be repartitioned before being input to the next round. But variants exist where there is no repartitioning, two different F-functions are used, with one "expanding" its input to be applied to the smaller branch and the other one compressing the input to apply the output to the larger branch, such as BEAR, LION and LIONESS [24]. The most extreme example of repartitioning unbalanced Feistel network is the Thorp shuffle, where the n bit state is divided into a 1 bit part and a $n-1$ bit part. The function F takes the $n-1$ bit part and outputs a single bit that is XORed to the 1 bit part. NLFSR-based ciphers such as KATAN [100] can be described in terms of unbalanced Feistel networks.

4. Generalized Feistel Network (GFN) Type 1. It is one of the many ways to describe unbalanced networks: instead of having two branches of different width, The state is split into more than two branches. The generalized Type 1 network depicted here has four branches, but there can be even more. CAST-256 [14] is a classic example of this design. Other variants are possible, for instance where the output of the F-function is combined to the in_3 branch. In this case the source branch gets permuted onto the target, as represented in the subfigure 4(b), and this variant is actually the inverse of the previous design, up to numbering of the branches. The Skipjack [15] rounds combine ideas from the Type 4 and Matsui networks.

5. Type 2, notable examples being RC6 [196], CLEFIA [219] encryption and key schedule.

6. Nyberg's Type 2 [189]. It is a variation of Type 2 with a different permutation of the branches. The subfigure 6(a) follows Kaisa Nyberg's original depiction, whereas the subfigure 6(b) shows how the permutation is different from the usual Feistel cyclic rotation of branches. The significative difference between Type 2 and Nyberg's Type is that in Type 2 source and target branches are always swapped, whereas in the Nyberg's Type some source and target branches are permuted to branches of the same type.

7. Zheng-Matsumoto-Imai Type 3.

8. MARS Type 3, also called target-heavy, which is used in MARS [74].

9. Source-heavy, as used in RC2. This is a dual of the target-heavy topology.

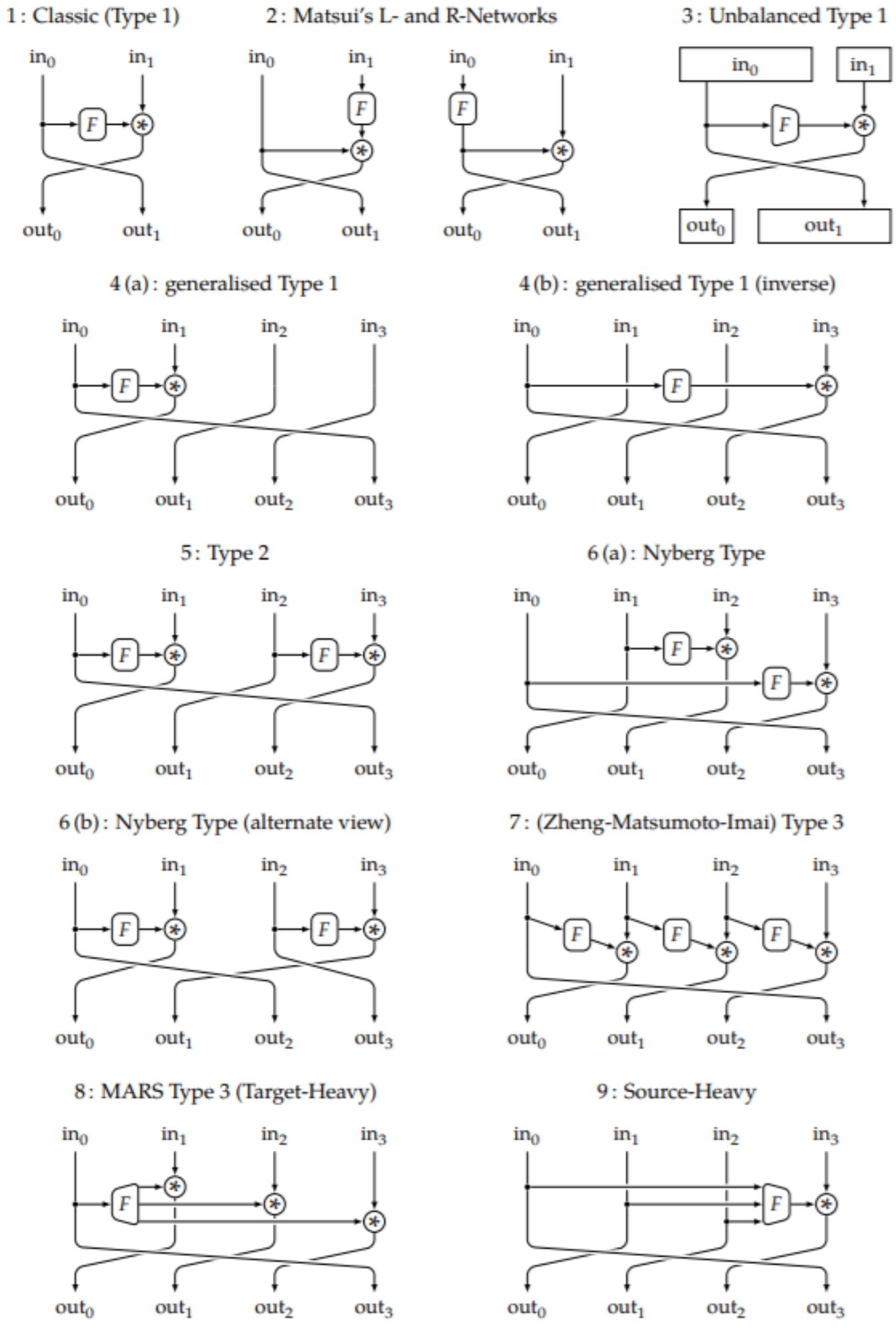


Figure 5.4: Different Feistel Types

5.5 Lai-Massey Design

In Feistel networks, part of the state is left unchanged implying that confusion is not applied to the whole state in each round. On the other hand, Feistel networks allow more freedom in the choice of the round functions than more general SPNs, and make it easier to design ciphers where the data obfuscation path is the same for encryption and decryption. IDEA [168] attempts to combine the advantages of both Feistel networks and more general SPNs. Despite being still, in principle, a SPN, its design is original enough to warrant its own terminology: some authors indeed call IDEA and similar ciphers a Lai-Massey Design, after the names of its inventors. There are two main differences with Feistel Networks: the two halves of the state are first combined before being fed to the F-function, and then the output of the F-function is combined with both halves of the state. The purpose is to accelerate both diffusion and confusion. In order to be able to reverse this scheme the input to F must be reconstructed from the outputs.
It is necessary that $L \circledast R = (L \circledcirc \Delta) \circledast (R \odot \Delta)$ is satisfied for all L, R and Δ. This is clearly satisfied if all three operations $\circledast, \circledcirc$ and $\odot$ are the bitwise XOR, but other combinations of operations are possible. For instance we can take $\circledcirc$ and $\circledast$ to be integer addition, while $\odot$ is integer subtraction.

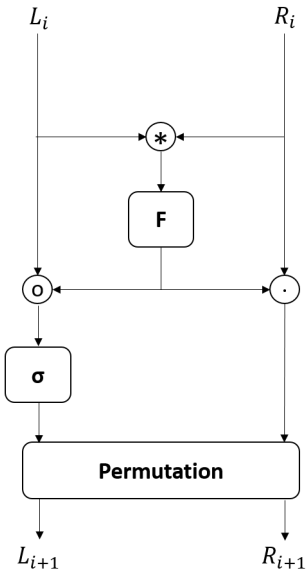


Figure 5.5: Substitution-Permutation Network

$L \oplus R$ is a constant, therefore a permutation layer is added after the F-function, and often an additional operation σ (called an orthomorphism) is necessary to avoid $\circledast$-differentials that an attacker may be able to exploit. The permutation layer alone may be sufficient, if it can guarantee good diffusion over sufficiently many layers. Some designs include σ but no permutation. Serge Vaudenay [226] studies security properties of Lai-Massey schemes and proves the same security bounds for Lai-Massey schemes with an orthomorphism as for Luby-Rackoff constructions, i.e. if the F-function is a random function, then three rounds are sufficient to make the block cipher a pseudo-random permutation, while four rounds is sufficient to make it a strong pseudo-random permutation.

5.6 The Even-Mansour Schemes and the FX construction

In 1991, Shimon Even and Yishay Mansour considered the question What is the simplest possible construction of a block cipher which is provably secure in some formal sense? To answer this question they came up with a minimalist design [117], inspired by Ron Rivest's FX construction. The DESX was proposed as a way to enhance the security of the DES by XORing two masked keys before and after the DES call. In general, the name of the construction refers to "F", i.e. any function, and "X", meaning that the function F is eXtended.

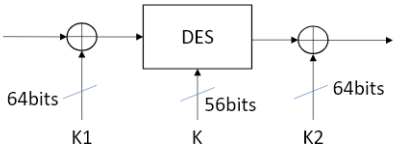


Figure 5.6: The DESX construction

The Even-Mansour (EM) scheme instead uses whitening keys, each as large as the plaintext block. The resulting scheme is extremely simple: to encrypt an n-bit plaintext, XOR it with a n-bit key, apply a publicly known permutation, and XOR the result with a second n-bit key. The main difficulty is to construct the permutation that must be a pseudo-random function.

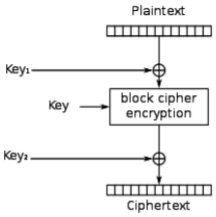


Figure 5.7: The Even Mansour scheme

Joan Daemen attacked this scheme [96] with a chosen ciphertext attack of complexity $O(2^{n/2})$, while the scheme needs $2n$ key bits. Orr Dunkelman, Nathan Keller and Adi Shamir proved in [110] a bound $\Omega(2^n/s)$ on attacks on the Even-Mansour scheme where s is the number of known plaintexts, matching Daemen's upper bound. They also showed that one key can be used in place of two (but both ends must be whitened), meaning that only n key bits are needed to achieve the same security bounds.
Thus, even assuming the central permutation is ideal, the best attainable security is $2^n/2$.
Better security bounds can be attained by alternating more key additions and state permutations. With these designs, not all rounds need to be keyed. For example MARS, Threefish or LED [148] apply key mixing only every four rounds.
The differences between the EM-scheme and the FX-construction are fundamental: the internal cipher of the FX-construction does not need to be a pseudo-random function (it is often a cipher for which the existence of distinguishers has been proved), and it is keyed. But the attacks on the EM scheme can still be mounted, for instance by fixing the internal l-bit key and mounting a generic attack on EM schemes on the resulting function. As a result, Kilian and Rogaway showed that if the core cipher E is ideal, the FX construction achieves $(l + n - 1 - log(T)$-bit security where T is the number of pairs of inputs and outputs for E known by the attacker.

The FX-construction is an easy way to strengthen an existing cipher (against brute force attacks, differential and linear cryptanalysis) with a very low additional implementation cost. But it has also become a design criterion on its own, as several ciphers are now designed including a key whitening step. The construction gives a trade-off security: in order to reach a level of $l + n$ bits of security (meaning that no attacks taking less time than 2^{l+n} can be mounted) a different design strategy is required. The

FEAL cipher [218] introduces an interesting form of key whitening that uses keys derived from the initial key material. In a Feistel cipher, this can increase security by concealing the specific inputs to the first and last round functions. This version of key whitening offers no additional protection from brute force attacks, but can make other attacks more difficult.
Some examples of cipher with key whitening: FEAL, MARS, Twofish, RC6, Camellia, HIGHT [142], CLEFIA, PRINCE [71] and Piccolo [217].
Pierre-Alain Fouque and Pierre Karpman described a stronger version of the FX construction where instead of simple key whitening more complex keyed functions are used [122].

The Key schedule can be used to specify the subkeys that are mixed to the state at each round in a product cipher (it is the case for the DES and most SPNs including AES). It can also be used to initialize some fixed elements of a cryptographic transform, such as substitution tables (for example in Blowfish and Twofish).
It can also be used to dynamically select functions from fixed families depending on its outputs. These families of functions can be Sboxes from a list, or rotations. This was the approach used in some early ciphers such as the DES, or CAST. In the case of FROG, the key schedule expands the key into a program describing the data obfuscation path.
The principle of confusion states that the key bits must be combined with the plaintext bits in a way as complex as possible. In an iterative block cipher with round key mixing between the rounds, the use of diffusion and confusion to get rid of any linear relation between plaintext and ciphertext contributes to the mixing of the key bits with the plaintext, such as the non-linearity between plaintext and ciphertext.
A first study of how complex the key schedule must be in order to achieve a sufficient diffusion of the key bits in combination with the round functions was done by Jialin Huang and Xuejia Lai [143]. They specified a criterion to evaluate key schedule diffusion that takes into account the round diffusion.
Most proofs of security for block ciphers rely on strong assumptions about the independence of the round keys, or the pseudo randomness of the output of the key schedule process.
There are several ways to construct a key schedule, such as using a linear key schedule, combining bit permutations and extractions and other linear operations (for example the DES, IDEA, or Skipjack). Masking with fixed constants to a linear key schedule can be added, such as in SIMON and Piccolo. PRINCE is an extreme example because each round key is just the master key XORed with a different round constant. The AES, PRESENT, or SPECK add a non-linear component to the linear operations. Some algorithms encrypt the master key using a block or stream cipher with fixed keys, and use the intermediate states of the block cipher or the output of the stream cipher as the subkeys. The same algorithm as the data obfuscation path can be reused to generate some or all of the subkeys, as in Camellia, or a different one, possibly reusing some of the components of the the data obfuscation path, as in Twofish, or CLEFIA. Blowfish repeatedly uses the encryption routine to generate the key expansion while SHARK's [193] key schedule is based on cipher itself in CFB mode.

5.7 State of the art

5.7.1 Norms and Standards

In this section we try to gather information about the norms and standards from different standardization organizations, namely the American one: the NIST (National Institute for Standard and Technologies), the French one: the ANSSI (Agence Nationale de la Sécurité des Systèmes d'Information) and a more global one: ISO (International Organization for Standardization). For each of these three organizations, we present the standards on block cipher algorithms, key size and the standardized mode of operation.

International Organization for Standardization ISO/IEC 18033 specifies encryption systems (ciphers) for the purpose of data confidentiality:

- for 64-bit block ciphers: TDEA, MISTY1 [178], CAST-128 [14], HIGHT [142],
- for 128-bit block ciphers: AES [97], Camellia [26], SEED.

ISO/IEC 29192-2:2019 specifies three block ciphers suitable for applications requiring lightweight cryptographic implementations:

- PRESENT: a lightweight block cipher with a block size of 64 bits and a key size of 80 or 128 bits.
- CLEFIA: a lightweight block cipher with a block size of 128 bits and a key size of 128, 192 or 256 bits.
- LEA: a lightweight block cipher with a block size of 128 bits and a key size of 128, 192 or 256 bits.

For the cryptographic mode of operation, the fourth edition of ISO/IEC 10116 specifies five modes for any n-bit block ciphers:

- Electronic Codebook (ECB);
- Cipher Block Chaining (CBC);
- Cipher Feedback (CFB);
- Output Feedback (OFB);
- Counter (CTR).

National Institute for Standards and Technologies Currently, there are two approved block cipher algorithms that can be used for encryption and/or decryption: AES and Triple DES. The Triple DES (or TDEA for Triple Data Encryption Algorithm) use has been deprecated through 2023. From 2023 and up to 2030, only the AES is certified for secure uses with a key size from 128 to 256 bits.
In SP800-38A [111] five confidentiality modes are specified for use with any approved block cipher, such as the AES algorithm. The modes in SP 800-38A are updated versions of the ECB, CBC, CFB, and OFB modes that are specified in FIPS 81 and the CTR mode.

ANSSI Up to 2020, the minimum recommended key size was 100 bits, but since 2020, the minimum one is 128 bits. Similarly, up to 2020, the minimum block size was 64 bits, and is now set up to 128 bits at minima.
The ANSSI does not recommend a specific algorithm, but provides a set of recommendations based on the resistance of an algorithm towards classical cryptanalysis attacks. To simplify the study, they specify three quantities: N_{op} the number of operations needed to mount the attack, N_{block} the number of blocks needed to cipher or decipher to mount the attack and N_{mem} the quantity of data to store to mount the attack. A simplified way of considering an algorithm as secure was to define wheither there was any known attack needing less than $N_{op} = 2^{128}$ operations. The other two quantities are

not mentioned to not complexify the rules. The current rule to even simplify the comprehension is "Il est recommandé d'employer des algorithmes de chiffrement par bloc largement éprouvés dans le milieu académique" which can be translated as it is recommended to use the block cipher algorithm that have been widely studied by the cryptographic community. Nevertheless, they give the AES as an example of a block cipher satisfying these recommendations.
As for the mode of operations, the recommendations are to use a non-deterministic one, jointly with an integrity mechanism and a mode of operations which has been described with security proofs.

5.7.2 DES

This algorithm is designed to encipher and decipher blocks of data consisting of 64 bits with a 56-bit key.
A block to encrypt is subjected to an initial permutation IP, then to a complex key-dependent computation and finally to a permutation which is the inverse of the initial permutation IP^{-1}. The key-dependent computation can be simply defined in terms of a function f, called the cipher function, and a function KS, called the key schedule.
The DES is designed following the Feistel construction, which means that at each round, the state will be divided into two 32-bit words L and R for respectively left and right part. The right part will be processed through the f function and the result is the XORed with the left part. Finally, the two branches are inverted. For the i^{th} round, we have $L_i = R_{i-1}$ and $R_i = L_{i-1} \oplus f(R_{i-1}, K_i)$, with K_i the subkey of round i.

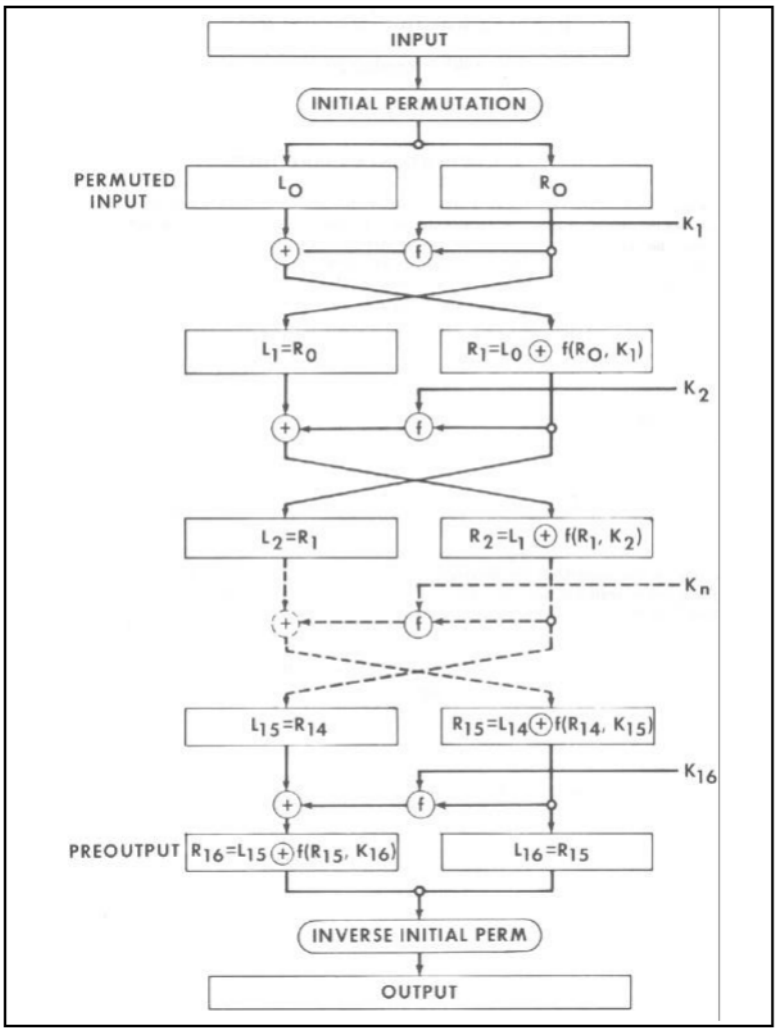


Figure 5.8: The DES

The IP and IP^{-1} are a permutation and its inverse, each of the 64 bits of the state will get scrambled to a different position according to the following tables:

58	50	42	34	26	18	10	2
60	52	44	36	28	20	12	4
62	54	46	38	30	22	14	6
64	56	48	40	32	24	16	8
57	49	41	33	25	17	9	1
59	51	43	35	27	19	11	3
61	53	45	37	29	21	13	5
63	55	47	39	31	23	15	7

Figure 5.9: The IP permutation table

40	8	48	16	56	24	64	32
39	7	47	15	55	23	63	31
38	6	46	14	54	22	62	30
37	5	45	13	53	21	61	29
36	4	44	12	52	20	60	28
35	3	43	11	51	19	59	27
34	2	42	10	50	18	58	26
33	1	41	9	49	17	57	25

Figure 5.10: The IP^{-1} permutation table

The f function is described in the figure below. It takes as input the half-state of 32 bits R_i and the subkey of 48 bits K_i. It first processes the 32-bit word through the expansion function E which expands it into a 48-bit word. The result is then XORed to the subkey. The resulting 48-bit word is then split into 8 6-bit words, each processed by a substitution box S_1 to S_8. Finally the result goes through a permutation table P.

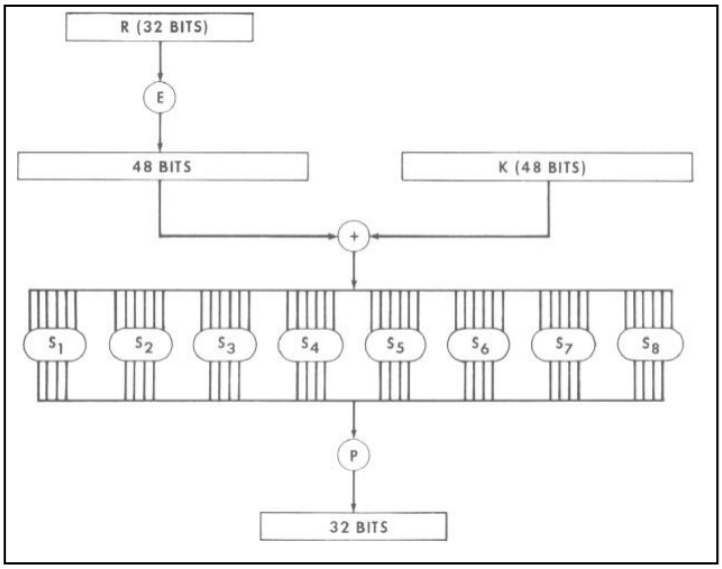


Figure 5.11: DES f function

The DES key schedule is a simplified Feistel form that is inspired by the encryption function. The key goes through a first permutation table then it is split in two halves. Each half undergoes a left shift. This operation is iterated 16 times, and at each step a round key is extracted.

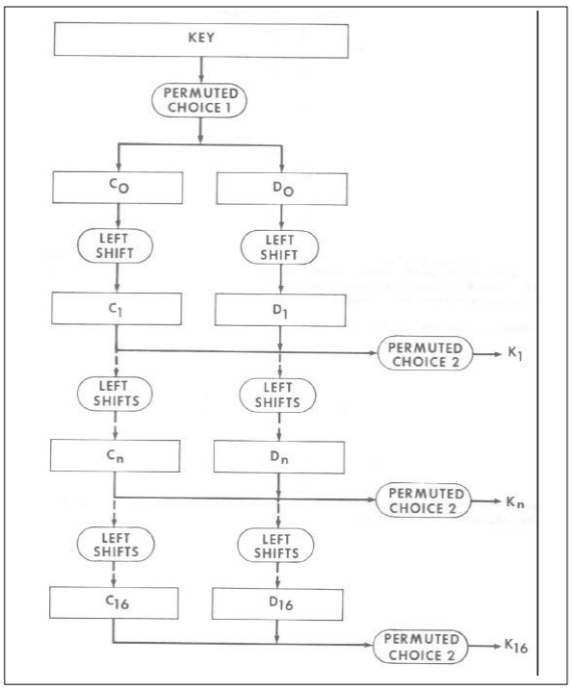


Figure 5.12: DES key schedule

5.7.3 AES

The AES is the universal standard block cipher. Rijndael was first published in 1998 and selected as the US AES in 2001 after a public selection process by the NIST. Rijndael is designed by Vincent Rijmen and Joan Daemen.

The first publication presented a block cipher with a block size and a key size between 128 and 256 bits with a specification for any multiple of 32 bits in between. AES fixes the block length to 128 bits, and supports key lengths of 128, 192 or 256 bits only.

The AES is a Substitution Permutation Network using 10 rounds for 128-bit keys, 12 for 192-bit keys and 14 for 256-bit keys. Each round of the AES is the composition of four different operations: SubBytes, ShiftRows, MixColumns and AddRoundKey. The state is represented as a 4×4 matrix of bytes. The state is initialized by the plaintext, then XORed with the first round key. The state is then updated the appropriate number of times according to the key size. The last round does not use the MixColumn operation.

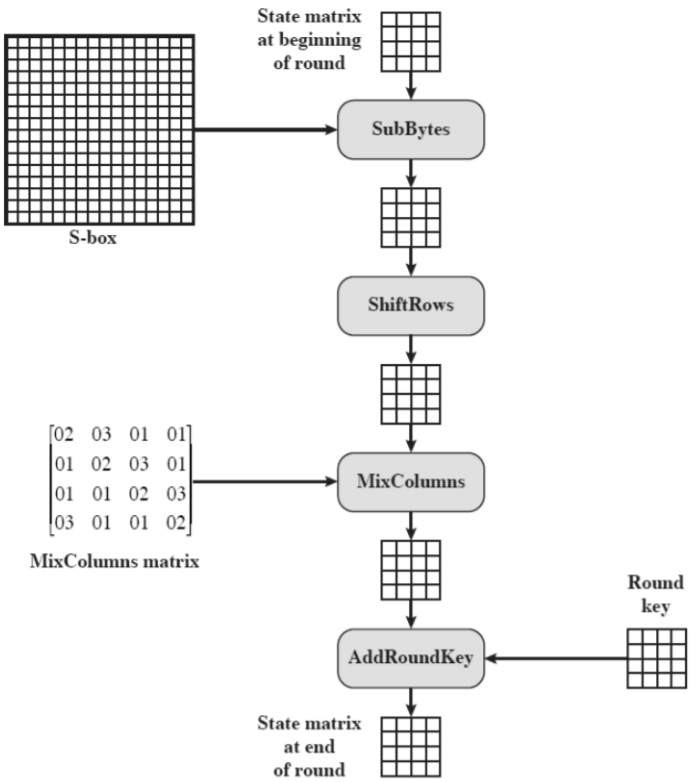


Figure 5.13: One AES round

The SubBytes operation is a non-linear substitution step where each byte $a_{i,j}$ is replaced with $S(a_{i,j})$ using a 8-bit Sbox, that can be seen as a lookup table, the Rijndael Sbox. This operation provides the non-linearity in the cipher. The Sbox used is derived from the multiplicative inverse over $GF(2^8)$, known to have good non-linearity properties. The Sbox is constructed by combining the inverse function with an invertible affine transformation. This is made to avoid attacks based on simple algebraic properties. The Sbox is also chosen to avoid any fixed point, i.e., $S(a_{i,j}) \neq a_{i,j}$, and also any opposite fixed point, i.e., $S(a_{i,j}) \oplus a_{i,j} \neq$ 0xFF. For the decryption, the Inverse SubBytes step is used, which requires first taking the affine transformation and then finding the multiplicative inverse (reversing the steps used in SubBytes step).

$$\begin{bmatrix} b_0' \\ b_1' \\ b_2' \\ b_3' \\ b_4' \\ b_5' \\ b_6' \\ b_7' \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$

Figure 5.14: The AES Sbox representation

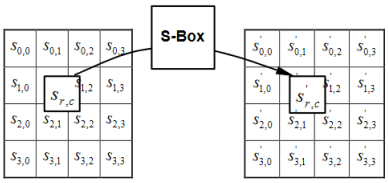


Figure 5.15: The SubByte operation

The ShiftRows operation is a transposition step where the last three rows of the state are cyclically shifted a certain number of steps.
It operates on the rows of the state by cyclically shifting the bytes in each row by an offset corresponding in the number of the row minus 1. The first row is left unchanged. Each byte of the second row is shifted one to the left. The third and fourth rows are shifted by two and three respectively. This way, each column of the output state of the ShiftRows step is composed of bytes from each column of the input state. (Rijndael variants with a larger block size have different offsets). The goal of this step is to avoid the columns being linearly independent, in which case, AES degenerates into four independent block ciphers.

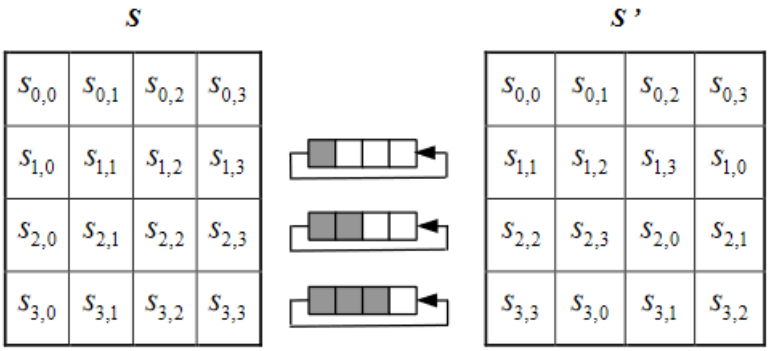


Figure 5.16: The ShiftRows operation

The MixColumns operation is a mixing operation that operates on the columns of the state, combining the four bytes in each column using an invertible linear transformation. The MixColumns function takes four bytes as input and outputs four bytes, where each input byte affects all four output bytes. This step increases the diffusion of the cipher.

During this operation, each column is transformed using a fixed matrix:

$$\begin{bmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{bmatrix}$$

Matrix multiplication is a set of multiplication and addition on words of 8 bits treated as coefficients of polynomial of order x^7 in $GF(2^8)$. Addition is a simple XOR while multiplication is modulo the irreducible polynomial $x^8 + x^4 + x^3 + x + 1$. The columns are considered as polynomials over $GF(2^8)$ and multiplied modulo $x^4 + 1$ with a fixed polynomial $a(x)$, given by $a(x) = 03x^3 + 01x^2 + 01x + 02$. The coefficients are displayed in their hexadecimal equivalent of the binary representation of bit polynomials from $GF(2)[x]$.

The AddRoundKey operation consists in XORing the round key to the state, byte by byte using a simple bitwise XOR.

The key expansion is used to generate the round keys from the main key. The AES needs $N_r + 1$ subkeys, with N_r the number of rounds, each of size N_b. The resulting key schedule consists of a linear array of 4-byte words, denoted $[w_i]$, with i in the range $0 \leq i < N_b(N_r + 1)$.
The expansion of the input key into the key schedule proceeds according to the pseudo code in figure below.

```
KeyExpansion(byte key[4*Nk], word w[Nb*(Nr+1)], Nk)
begin
   word  temp

   i = 0

   while (i < Nk)
      w[i] = word(key[4*i], key[4*i+1], key[4*i+2], key[4*i+3])
      i = i+1
   end while

   i = Nk

   while (i < Nb * (Nr+1)]
      temp = w[i-1]
      if (i mod Nk = 0)
         temp = SubWord(RotWord(temp)) xor Rcon[i/Nk]
      else if (Nk > 6 and i mod Nk = 4)
         temp = SubWord(temp)
      end if
      w[i] = w[i-Nk] xor temp
      i = i + 1
   end while
end
```

Figure 5.17: The AES key expansion pseudo-code

SubWord() is a function that takes a four-byte input word and applies the Sbox to each of the four bytes to produce an output word. The function RotWord() takes a word $[a_0, a_1, a_2, a_3]$ as input, performs a cyclic permutation, and returns the word $[a_1, a_2, a_3, a_0]$. The round constant word array, Rcon[i], contains the values given by $[x^{i-1}, \{00\}, \{00\}, \{00\}]$, with x^{i-1} being powers of x.

5.7.4 Lightweight algorithms

Existing cryptographic algorithms were, for the most part, designed to meet the needs of the desktop computing era. Such cryptography is not very well-suited for pervasive computing, in which many highly constrained hardware- and software-based devices will need to communicate wirelessly with one another. Security is important for these devices and the cost of ciphers, as well as the cryptographic protocols are important matters while more and more of these devices can carry private information. Moreover, lightweight ciphers need to be easily secured against side-channel analysis because many lightweight devices are easily accessible in huge amount so that they can be more easily analyzed and attacked. Moreover, because of the cost, they are often unprotected.

The relatively new field of lightweight cryptography addresses the security issues for highly constrained devices. A lot of work has been done in this area, much of it aiming specifically at developing block ciphers suitable for lightweight cryptographic applications.

The obvious first question for developers of lightweight applications is "Why not build protocols around AES?" Indeed, it has been suggested for lightweight use, and should be used whenever appropriate. However, for the most constrained environments, AES is not always the right choice: in hardware, for example, the consensus in the academic literature is that area should not exceed 2000 gate equivalents, while the smallest available implementation of AES requires 2400.

Among the block ciphers intended for use on constrained devices, some have been designed specifically to perform well on dedicated Application-Specific Integrated Circuits (ASICs), and thus can be realized by small circuits with minimal power requirements. Others are meant to perform well on low-cost microcontrollers with limited flash, SRAM, and/or power availability. Unfortunately, design choices meant to optimize performance on one platform often are not compatible with another one.

PRESENT PRESENT is a 64-bit block cipher developed by Andrey Bogdanov et al. and introduced at CHES 2007 [66]. The algorithm is described with two versions, with keys of 80 and 128 bits. It was designed for low-cost devices like RFID-tags after realizing that the AES was not suitable for

extremely constrained environments due to its implementation cost.
PRESENT is a Substitution-Permutation Network with 31 rounds. Each round consists of: a round key addition, a substitution layer, and a permutation layer. A final key addition is computed before outputting the ciphertext.

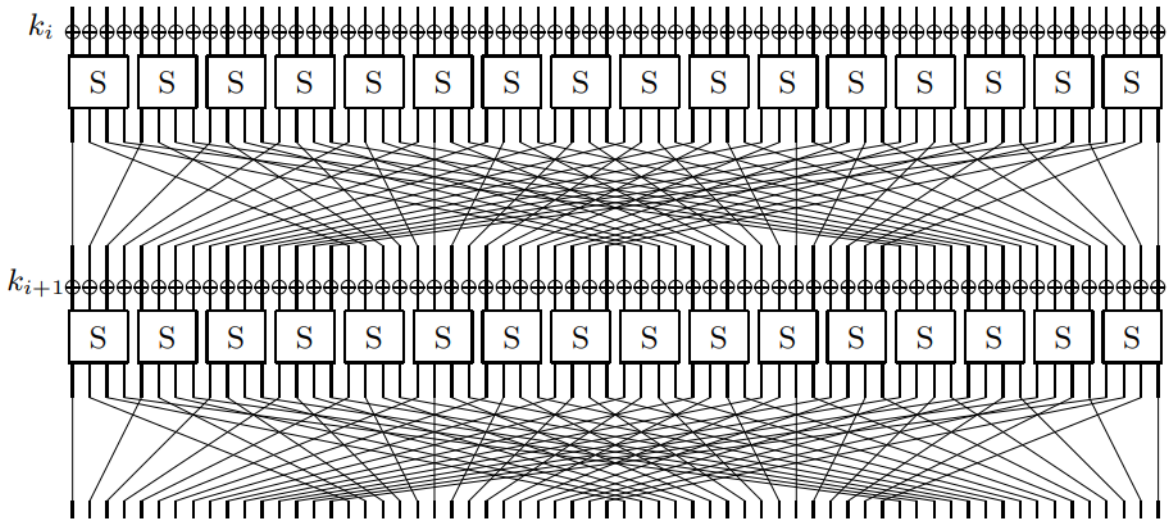


Figure 5.18: A PRESENT round

The substitution layer is composed of 4-bit Sbox applied to each 16 nibbles of the 64-bit state. This Sbox was designed with cryptanalytic resistance in mind. Conditions were posed on the Sbox to improve the avalanche of change:

- For any output difference, there are at most four inputs differences.
- There are no single bit input differences that give single bit output differences.
- For all non-zero a, $b \in \mathbb{F}_2^4$, $|S_b^\omega(a)| \leq 8$, with $S_b^\omega(a)$ the Walsh-Fourier coefficient of S.
- For all a, $b \in \mathbb{F}_2^4$, both with Hamming weight equal to 1, $|S_b^\omega(a)| = 4$.

The chosen Sbox was sorted among the set of all 4-bit Sbox fulfilling these requirements, the chosen one being the one with the lowest hardware footprint.

x	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
S(x)	C	5	6	B	9	0	A	D	3	E	F	8	4	7	1	2

Table 5.1: The PRESENT Sbox

This Sbox has been reused in several cryptographic applications, such as SPONGENT [67] or PHOTON [136] due to its good properties on both security and hardware efficiency.
The permutation layer is a pure bit permutation layer, associating each input bit to one output. This construction allows a good hardware efficiency by minimizing the number of processing elements. This design gives the propriety that any five-round differential characteristic of PRESENT has a minimum of 10 active Sboxes.
For the key schedule: in the case of 80-bit keys, a 80 bit register containing the secret key is rotated by 61 positions to the left, the least significant nibble is passed through the Sbox, and then the value of a counter is XORed to some of the bits of the register. The round key is the leftmost 64 bits of this register.
In the case of 128-bit keys, the round key register is 128 bits wide and two nibbles are passed through the Sbox.

SIMON, SPECK and Simeck In 2013, a team from the NSA published a paper presenting two new cipher algorithms dedicated to lightweight cryptography [44]. SIMON and SPECK are designed to be secure, flexible, and analyzable lightweight block ciphers offering good performances on hardware and software platforms, while being flexible enough to admit a variety of implementations on a given platform. They are not given with any security proofs but are said to be easily analyzable.
They both perform well across numerous lightweight applications, but SIMON is tuned for optimal performance in hardware, and SPECK for optimal performance in software.
SIMON and SPECK are designed to be block cipher families: Each supports block sizes of 32, 48, 64, 96, and 128 bits, with up to three key sizes to go along with each block size. Each family provides ten algorithms in all.
The SIMON block cipher with a n-bit word (and hence a $2n$-bit block) is denoted SIMON2n, where n is required to be 16, 24, 32, 48, or 64. SIMON2n with an m-word (mn-bit) key will be referred to as SIMON2n/mn. For example, SIMON64/128 refers to the version of SIMON acting on 64-bit plaintext blocks and using a 128-bit key. Each instance of SIMON is based on a Feistel Network.
SIMON2n encryption and decryption use only three operations on n-bit words: bitwise XOR ($\oplus$), bitwise AND (&), and left circular shift (S^j), by j bits.
The SPECK2n computation makes use of the following operations on n-bit words: bitwise XOR ($\oplus$), addition modulo 2^n ($+$), and left and right circular shifts, S^j and S^{-j}, respectively, by j bits.

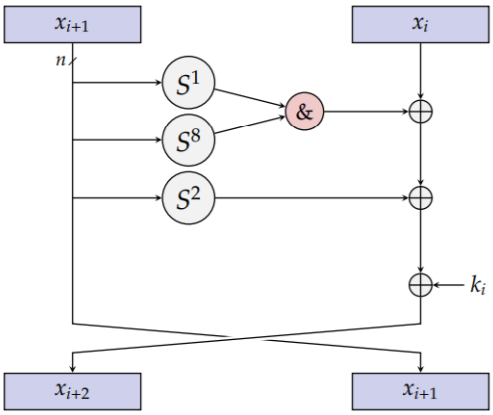


Figure 5.19: Simon round

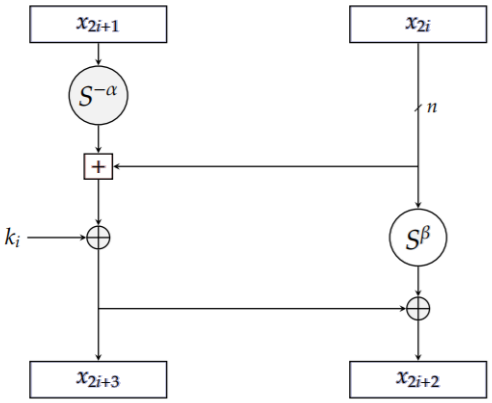


Figure 5.20: SPECK round

Inspired by the designs of SIMON and SPECK, Simeck was designed, combining their good components in order to get a new design of block cipher family [237]. The encryption process is a modified version of SIMON's round function, and it reuses it in the key schedule like SPECK does. Moreover, the key schedule uses Linear Feedback Shift Register (LFSR) based constants in order to further reduce hardware implementation footprints. The new family of lightweight block ciphers Simeck aims to have comparable security levels as Simon and Speck but more efficient hardware implementations.
The family proposes three members with block/key size of 32/64, 48/96 and 64/128 bits.

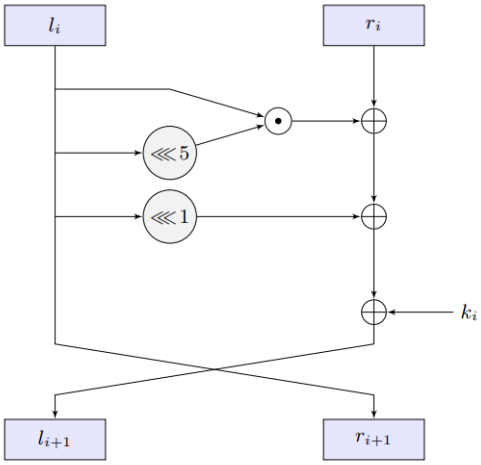


Figure 5.21: A Simeck round

To generate the round key k_i from a given master key K, the master key is first segmented into four words and loaded as the initial states (t_2, t_1, t_0, k_0) of the feedback shift registers. The n least significant bits of K are loaded into k_0 and the n most significant bits are put into t_2. To update the registers and generate round keys, the round function with a round constant $C \oplus (z_j)_i$ acting as the round key is used, i.e.

$$\begin{cases} k_{i+1} &= t_i \\ t_{i+3} &= k_i \oplus f(t_i) \oplus C \oplus (z_j)_i \end{cases}$$

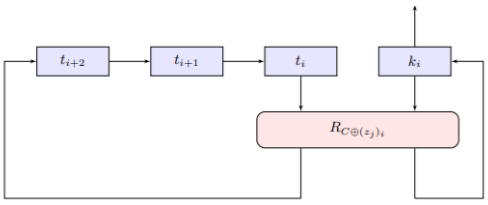


Figure 5.22: The Simeck key schedule

5.8 Tweakable block ciphers

5.8.1 Definition

The first cipher that could be called a tweakable block cipher is the Hasty Pudding Cipher [214], designed by Schroepel for the AES competition. However, the concept of Tweakable Block Cipher (TBC) was introduced by Moses Liskov, Ronald L. Rivest, and David Wagner in their article [171]. A tweakable block cipher accepts as input the usual block of plaintext and the key plus an additional tweak. The tweak serves the same purpose as an initialization vector or a nonce (number only used once) does for some mode of operation. The authors claim three properties characterizing a "good" tweakable block cipher:

- tweakable block ciphers are easy to design,
- the extra cost of making a block cipher tweakable is small,
- and it is easier to design and prove modes of operation based on tweakable block ciphers.

The first construction they described was $\tilde{E}_K(T, M) = E_K(T \oplus E_K(M))$ with $\tilde{E}_K$ the tweakable block cipher based on the block cipher E_K using the key K, a message M and a tweak T. They prove that if E_K is a secure block cipher then $\tilde{E}_K$ is a secure tweakable block cipher, meaning that any adversary can only access a small advantage in distinguishing the encryption algorithm from a random permutation.
The second one has the form $\tilde{E}_{K,h}(T, M) = E_K(M \oplus h(T)) \oplus h(T)$ with h a hash function from $\{0,1\}^t \to \{0,1\}^n$. If the set of function h is ϵ-almost 2-xor-universal ($Pr_h[h(x) \oplus h(y) = z] \leq \epsilon$ for all (x, y, z) with $\epsilon \leq 1/2^n$ then $\tilde{E}$ is a strong tweakable block cipher, it is chosen-ciphertext secure.

Building tweakable block ciphers There are mainly three approaches to build a tweakable block cipher. The first one is to start from scratch which was the case for Hasty Pudding Cipher [214], but also for Mercy [91] and Threefish (used in the SKEIN Hash function) [175]. The second one is to introduce the additional tweak to generic constructions of block cipher. Methodologies exist for different structure of block ciphers such as Feistel-based ciphers with [130], generalized Feistel Networks with [184] or key alternating ciphers (iterated Even-Mansour) with for example [88]. These designs are provably secure with a security up to $2/n/2$, n being the size of the message space. The last approach is also the most common one and consists in starting from a classical block cipher and using it as a black block to build a TCB. This is the case for the LRW1 and LRW2 constructions [171].

5.8.2 The TWEAKEY Framework

In 2014, Jean, Nikolic and Peyrin introduced the TWEAKEY framework [147] to unify the design of tweakable block ciphers and of block ciphers resistant to related-key attacks. The design goal is to extend the key-alternating construction, and allow the construction of a primitive with arbitrary tweak and key sizes, given the public round permutation such as the AES round, i.e to blend the tweak with the key in the key scheduling algorithm.
The design of the TWEAKEY framework came from the two points below:

- From an efficiency point of view, updating the tweak input of a TBC should be doable very efficiently which means that the tweak schedule should be lighter than the key schedule.
- From a security point of view, the tweak is fully known and controllable, unlike the key which means that the tweak schedule should be stronger than the key schedule.

This leads to the point that the tweak and the key should be treated equivalently.
The regular key schedule is replaced by a TWEAKEY schedule that generates subtweakeys. A TK-2 primitive (or TWEAKEY of order 2) is a TBC with n-bit key, n-bit tweak that has $2n$-bit TWEAKEY and a function g that compresses $2n$ to n bits (which can also be seen as a $2n$-bit key cipher with no tweak).

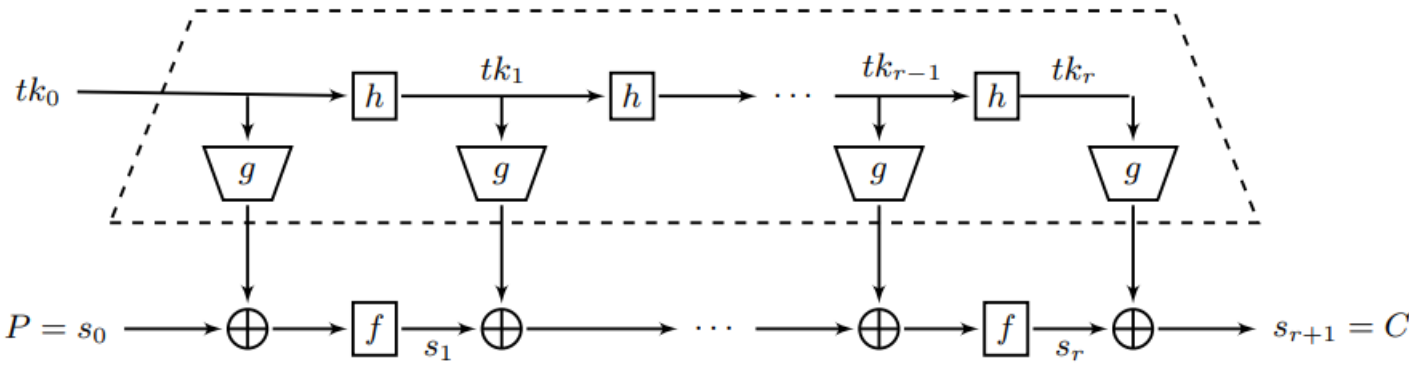
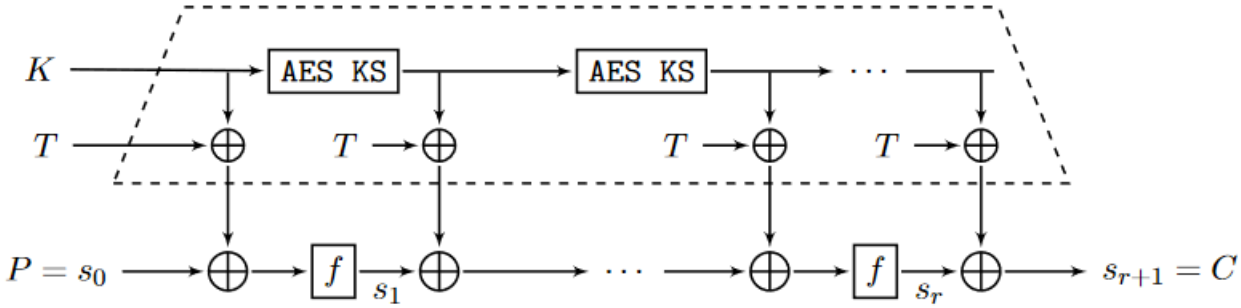


Figure 5.23: The TWEAKEY framework

This design gave birth to the KIASU cipher which is based on the AES with a fixed 64-bit tweak value XORed to each subkey. The f function in the figure below is the AES round and AES KS represents the AES key schedule.



$$T = \begin{array}{|c|c|c|c|} \hline T_0 & T_2 & T_4 & T_6 \\ \hline T_1 & T_3 & T_5 & T_7 \\ \hline 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 \\ \hline \end{array}$$

Figure 5.24: The KIASU block cipher

The TWEAKEY framework unifies the tweak and key input for a tweakable block cipher, but does not provide real instantiation of this construction, i.e. which functions f, g and h (and number of rounds r) one should choose. Hence, the designers introduced the STK construction (Superposition TWEAKEY) in order to simplify the analysis of the concrete security of such scheme and give the ability to reach security bounds given the input parameters.
The STK construction is a subclass of the TWEAKEY framework for AES-like ciphers defined over a finite field $GF(2^c)$

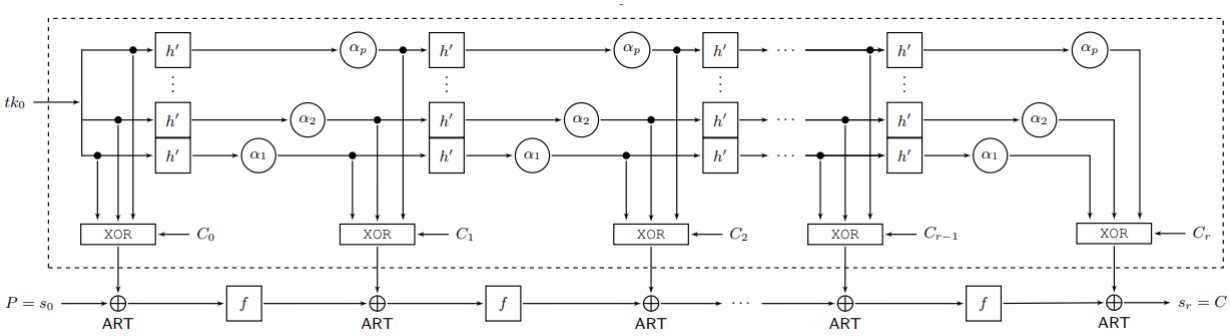


Figure 5.25: The STK framework

$p = (t+k)/n$ denotes the number of n-bit words in the TWEAKEY state composed of t-bit tweak and k-bit key. Assuming that the AES-like Sbox operates on c bits, the STK construction further specifies the f, g and h functions:

- the function g XORs all the p n-bit words of the TWEAKEY state to the internal state (AddRoundTweakey, denoted ART), and then XORs a round-dependent constant C_i.
- the function h first applies the same nibble position substitution function h_0 to each of the p n-bit words of the TWEAKEY state, and then multiplies each c-bit cell of the j-th n-bit word by a nonzero coefficient α_j in the finite field $GF(2^c)$.
- the f function is an AES-like round.

This construction was used to design the block ciphers Deoxys and Joltik.

5.8.3 Targeting optimally secure tweakable block ciphers

In 2015, Bart Mennink published a new work on designing Tweakable Block Cipher with said optimal security [182]. This work tries to answer the following question: Can we design an optimally secure tweakable block cipher $\tilde{E}$ with n-bit in- and outputs using only a block cipher E with n-bit in- and outputs?
The generic design to achieve this uses a classical block cipher E, called ρ times, with arbitrary mixing functions to generate the inputs to primitive calls and generate the final output. The result is called $\tilde{E}[\rho]$ with $\rho \geq 1$.
With one block cipher call and using polynomial mixing function (involving multiplication), the $\tilde{F}[1]$ construction stands with the form :

$$\begin{aligned} \tilde{F}[1] &: \{0,1\}^n \times \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^n \\ \tilde{F}[1](k,t,m) &= E(k \oplus t, m \oplus z) \oplus z, \text{where } z = k \otimes t \end{aligned} \tag{5.1}$$

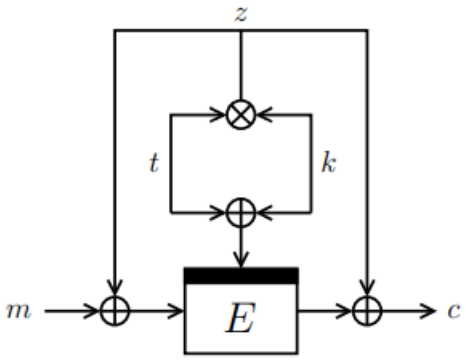


Figure 5.26: Tweakable Block Cipher $\tilde{F}[1]$

This construction is proven indistinguishable from an ideal tweakable cipher as long as the distinguisher's complexity is at most $2^{2n/3}$.
With two block cipher calls and linear mixing functions the designers also introduced the $\tilde{F}[2]$ construction:

$$\begin{aligned} \tilde{F}[2] &: \{0,1\}^n \times \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^n \\ \tilde{F}[2](k,t,m) &= E(k \oplus t, m \oplus z) \oplus z, \text{where } z = E(2k,t) \end{aligned} \tag{5.2}$$

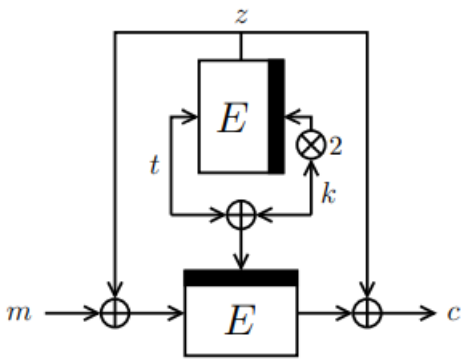


Figure 5.27: Tweakable Block Cipher $\tilde{F}[2]$

Using slightly more computations than for $\tilde{F}[1]$, $\tilde{F}[2]$ is proven to be an optimally secure tweakable cipher up to about 2^n queries.

In 2016, Wang and al. published a new work on the construction of secure tweakable block ciphers [227] aiming at designs achieving full 2^n security. They searched for every possible instances of tweakable block ciphers with only two calls to the inner block cipher and found 32 efficient solutions that provide up to 2^n provable security. Every solution follows the following form :

$$\tilde{E}[b_p, b_c](k,t,p) := \tilde{E}(k,t,p \oplus (b_p.t)) \oplus (b_c.t)$$

for all $k \in \mathcal{K}$, $t \in \mathcal{T}$ and $p \in \mathcal{M}$, with $b_p, b_c \in \{0,1\}$.

5.8.4 Use of tweakable block ciphers

Tweakable block ciphers and tweakable modes of operation have seen different uses since the first construction in 2020. One application of tweakable block ciphers is disk encryption, where each block is encrypted with the same key, plus a tweak that corresponds to the block index. Disk encryption can also be achieved with tweakable mode of operation with for example the XTS mode [198]. Tweakable block ciphers can also be used to generate authenticated encryption, with several examples at the CAESAR competition [2] and the NIST LWC standardization process [8] and the ciphers AEGIS-128 [235] or Deoxys-II [146] which both ranked first at the CAESAR competition for their respective use case.

Chapter 6

Authenticated encryption

6.1 Concept and definitions

For a communication to be considered secure, it needs to achieve two characteristics, namely confidentiality and integrity.
Confidentiality ensures that the information transmitted between a set of correspondents is only accessible to those whose access is granted to this information. In cryptography, this means that a secure communication is encrypted in a way that only the correspondent knowing both the encryption algorithm and the key to encrypt/decrypt the data can access it.
The integrity means that any data stays accurate and unchanged during its treatment, transmission, or saving.
A system that should guarantee data integrity has to possess a way to check whether the data has been changed during its life cycle (either a deletion, addition, or modification of some bits of the message) or not. Usually, confidentiality is achieved through encryption and integrity through authentication, using a secure MAC. Authenticated encryption tries to achieve both security goals in a single algorithm or protocol.

The need for authenticated encryption emerged from the observation that securely combining separate confidentiality and authentication block cipher operation modes could be error prone and difficult. This was confirmed by a number of practical attacks introduced into production protocols and applications by incorrect implementation, or lack of authentication.

Authenticated encryption can be generically constructed by combining an encryption scheme and a MAC, provided that the encryption scheme is secure under chosen plaintext attacks and the MAC function is unforgeable under chosen message attacks.

A typical programming interface for an AE implementation provides the following functions:

- Encryption
  - Input: plaintext, key, and optionally a header in plaintext that will not be encrypted, but will be covered by authenticity protection.
  - Output: ciphertext and authentication tag (message authentication code).
- Decryption
  - Input: ciphertext, key, authentication tag, and optionally a header (if used during the encryption).
  - Output: plaintext, or an error if the authentication tag does not match the supplied ciphertext or header. The header part is intended to provide authenticity and integrity protection for networking or storage metadata for which confidentiality is unnecessary, but authenticity is desired.

In addition to protecting message integrity and confidentiality, authenticated encryption can provide security against chosen ciphertext attack. Authenticated encryption schemes can recognize improperly-constructed ciphertexts and deny them. This prevents the attacker from requesting the decryption of

any ciphertext unless it was generated correctly using the encryption algorithm, thus implying that the plaintext is already known. Implemented correctly, authenticated encryption removes the usefulness of the decryption oracle, by preventing an attacker from gaining useful information that he does not already possess.

6.1.1 Variants of Authenticated Encryption

Encrypt-then-MAC (EtM) The plaintext is first encrypted using for example a block cipher and a key K_1, then a MAC is produced based on the resulting ciphertext using for example a cryptographic hash function and a key K_2. The ciphertext and the MAC are sent together, concatenated. The keys used to encrypt and to produce the MAC need to be different to maintain the expected level of security.

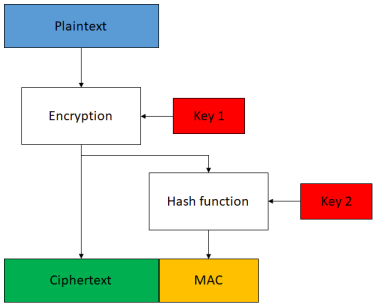


Figure 6.1: Encrypt-then-MAC

This method is used for example in the IPSec protocol, it is the standard method according to ISO/IEC 19772:2009. This is the best composition method (the method that can mathematically reach the highest definition of security in AE), but this needs a strongly unforgeable MAC.

Encrypt-and-Mac (E&M) The MAC and the encryption process are done simultaneously, both using the plaintext. The Ciphertext and the MAC are sent together. The same key can be used to generate both cryptographic applications.

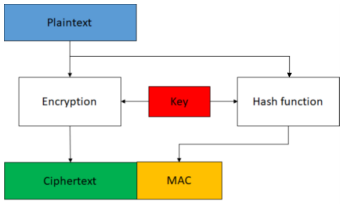


Figure 6.2: Encrypt-and-MAC

This method is used in SSH. Even though the E&M approach has not been proved to be strongly unforgeable in itself, indeed, their is no integrity on the ciphertext, so it is possible to apply some minor modifications to SSH to make it strongly unforgeable.

MAC-then-Encrypt (MtE) The plaintext is first used to generate a MAC. The resulting MAC is then concatenated to the plaintext and encrypted to produce the ciphertext. Only the ciphertext is sent.

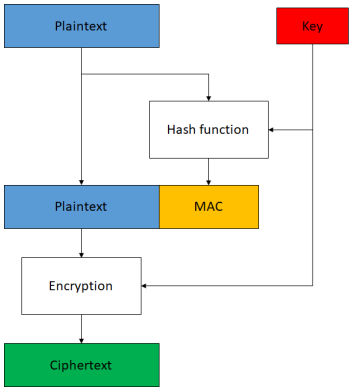


Figure 6.3: MAC-then-Encrypt

This method is used for example in SSL/TLS. Even though the MtE approach has not been proven to be strongly unforgeable in itself (it is susceptible to padding oracle attacks), the SSL/TLS implementation has been proven to be strongly unforgeable thanks to the encoding used alongside the MtE mechanism. SSL/TLS can be modeled as a MAC-then-pad-then-encrypt mechanism, i.e. the plaintext is first padded to the block size of the encryption function.

6.2 Mode of operation

The NIST specifies five modes of operation to achieve both confidentiality and authentication:

- Two authenticated encryption modes
  - An Authenticated Encryption Mode: counter with cipher block chaining message authentication code; counter with CBC-MAC (CCM). CCM combines the counter mode for confidentiality with the cipher block chaining technique for authentication (CBC-MAC).
  - A High-Throughput Authenticated Encryption Mode : Galois/Counter Mode (GCM). GCM combines the counter mode for confidentiality with an authentication mechanism that is based on an universal hash function. GCM was designed to facilitate high-throughput hardware implementations.
- Three key wrapping methods
  - AES Key Wrap (KW).
  - AES Key Wrap with Padding (KWP).
  - TDEA Key Wrap (TKW).

Three other modes of operation to achieve authenticated encryption have been submitted to the NIST for consideration:

- Galois/Counter Mode-Synthetic Initialization Vector (AES-GCM-SIV).
- Offset Codebook (OCB).
- Encrypt-then-Authenticate-then-Translate (EAX).

All of these modes of operations are defined with a block cipher which is not modified, either the AES or TDEA.

6.2.1 Galois/Counter Mode

The Galois/Counter Mode is a cryptographic mode of operation using block ciphers to achieve authenticated encryption. This mode of operation is widely used because of its performances.

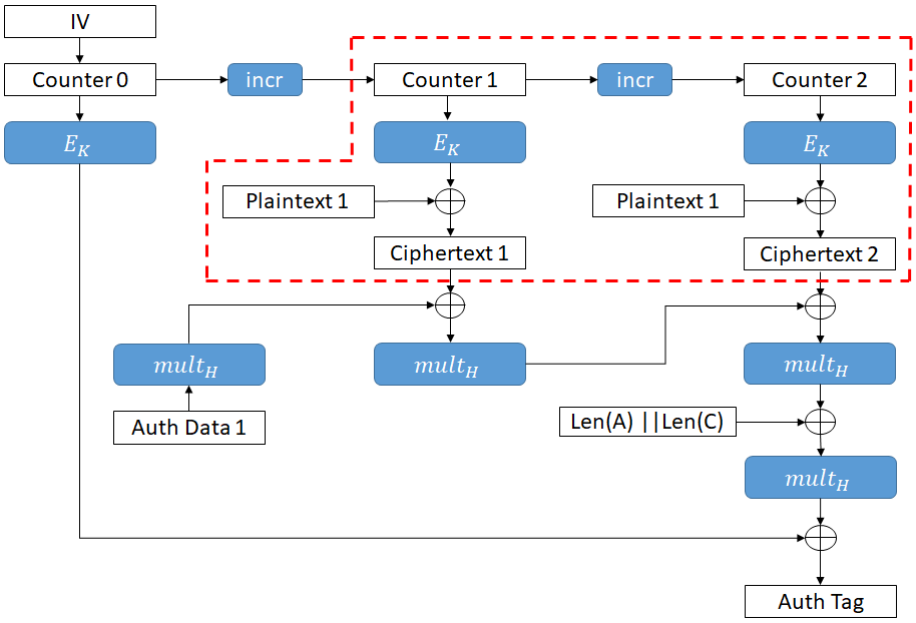


Figure 6.4: The Galois/Counter Mode

The red part of the scheme is the CTR mode of operation, the rest of the scheme is the Galois Hash (GHASH) function, generating the authentication tag from the ciphertext and additional data. The $mult_H$ block in the scheme represents the multiplication in the Galois field GF128 of the Additional Authenticated Data and H. The value H is generated by computing one encryption using the block cipher taking as input a message made of 128 zeros.
E_k represents the encryption using the block cipher E (usually the AES) with the key K. $Len(A)||Len(C)$ is the concatenation of the bit vector representation of the length of the last block of authenticated data and the last block of ciphertext.
Such as in the CTR mode, blocks are numbered sequentially. It is defined for block ciphers with a block size of 128 bits. The mode uses an initialization vector of arbitrary size as input, which will feed a counter, incremented for each block of 128 bits. Each counter obtained is then computed through the block cipher E_K and the result is XORed to the plaintext to obtain the resulting ciphertext. Like all counter modes, this is essentially a stream cipher, forcing to use a different IV for each data that is encrypted.

Security GCM is proven secure in the concrete security model when used with a block cipher indistinguishable from a random permutation. This security depends on choosing a unique initialization vector for every encryption performed with the same key. For any given key and initialization vector combination, GCM is limited to encrypting $2^{39} - 256$ bits of plain text (68 GB). Moreover, GCM is neither well-suited for very short tag-lengths nor very long messages.
As for all symmetric message authentication codes, the authentication strength depends on the length of the tag. T is recommended to be any one of the following five values: 128, 120, 112, 104, or 96. The bit-length of the tag, denoted T, is a security parameter. Like with any message authentication code, if the adversary chooses a T-bit tag at random, it is expected to be correct for given data with probability 2^{-T}. With GCM, however, an adversary can increase their likelihood of success by choosing tags with words of size n, i.e the total length of the ciphertext plus any additional authenticated data (AD), described as $Len(A) + Len(C)$ in our scheme. This leads to a probability of finding the tag in $1 - n.2^{-T}$ for arbitrarily large T. In 2005, Ferguson and Saarinen independently described how an attacker can perform optimal attacks against GCM authentication [120] [205] (the attack meets the lower bound on its security). Ferguson showed that there is a method of constructing a targeted

ciphertext forgery that is expected to succeed with a probability of approximately $n.2^{-T}$, with n the number of blocks in the encoding (the input to the GHASH function). If the tag length t is shorter than 128, then each successful forgery in this attack increases the probability that subsequent targeted forgeries will succeed, and leak information about H. Eventually, H may be compromised entirely and the authentication assurance completely lost.
GCM is not nonce misuse resistant, which means that an attacker, given the ability of using the same nonce to encrypt different messages can practically break the AES-GCM [43].
Moreover, the GCM mode of operation does not achieve Random Key Robustness (two different plaintext, used with two different keys can not produce the same ciphertext + tag), which is a required property for tools targeting Password Authenticated Key Exchange.

6.3 Context of research

In July 2012, a workshop focused on authenticated encryption took place in Stockholm: DIAC - Directions in Authenticated Ciphers, organized by the virtual labs SymLab [6] and VAMPIRE [10]. The goal was to take stock on the state of the art on authenticated encryption. The CAESAR Competition had just been announced and the objective was to shape the future competition with research axis in the same way that SASC 2004 and the ECRYPT Hash Workshop 2007 shaped the ECRYPT Stream Cipher Project and the SHA-3 competition: "Users, starting with a shared secret key, need to protect messages against espionage and against forgery. Dissatisfaction with the security and performance of current approaches has led to calls for a new competition for authenticated ciphers. The purpose of this workshop is to evaluate the state of the art in authenticated encryption and gather community input regarding desired future directions."

During DIAC 2013 [4], the cryptographic community started a common effort to update the state of the art in authenticated encryption which continued with the launch of the Competition for Authenticated Encryption: Security, Applicability and Robustness (CAESAR) [2].

6.3.1 CAESAR Competition

CAESAR (Competition for Authenticated Encryption: Security, Applicability, and Robustness) is a crypto competition aiming to identify a portfolio of authenticated ciphers that offer advantages over AES-GCM and are suitable for a widespread adoption. The first submission deadline met 57 candidates, with some offering more than one version, leading to a total of 61 proposals. The competition went on from March 2014 up to February 2019 with the announcement of the final portfolio.
Among all candidates, four categories can be distinguished:

- Block Cipher-based: mode of operation using a block cipher as basis. Several of the candidates were based on the AES [97], using a mode of operation using it as its main cryptographic core to achieve authenticated encryption, this is the case for OCB. Some candidates proposed their own block cipher, such as Deoxys. Some candidates were designed using parts of known block cipher, such as AEGIS.

- Stream-cipher-based: like block ciphers, stream ciphers can be used as a core primitive in authenticated encryption scheme to achieve both confidentiality and integrity as long as the cipher is secure. Stream ciphers are designed to be fast and small in size, and are mainly targeting applications with low resources such as embedded devices. ACORN, the second recommendation for lightweight applications is stream-cipher-based.

- Compression Function-based: a one-way compression function transforms two fixed-length inputs into one fixed-length output. OMD is a candidate based on the Merkle-Damgård construction, which is widely used to create hash functions.

- Permutation-based: permutation-based candidates are algorithms using several calls to a permutation taking as inputs every elements needed to perform the authenticated encryption (plaintext, additional data, nonce, key) to produce both the ciphertext and the tag. This is the case for Minalpher [208] for example. A large part of permutation-based candidates are designed using the sponge construction, like ASCON.

During this competition, three rounds were organized, each one focusing on one or a few points, in order : security and usability for the first round, software performances for the second and hardware performances for the third one. The CAESAR competition was the first one with a hardware dedicated API proposed to guide cryptographers to implement their candidate on FPGA or ASIC technology. Several candidates proposed interesting properties and eventually, no proposal was better than all the others, depending on the focus undertaken by the designers.

The final portfolio is organized into three use cases, each one providing two choices :

Use case	Candidate	Designers
Lightweight Applications	ASCON [106] ACORN [234]	C. Dobraunig, M. Eichlseder, F. Mendel,et al. H. Wu
High-performances Applications	AEGIS-128 [235] OCB [201]	H. Wu, B. Preneel T. Krovetz, P. Rogaway
Defense in depth	Deoxys-II [146] COLM [25]	J. Jean, I. Nikolić, T. Peyrin, et al. E. Andreeva, A. Bogdanov, N. Datta, et al.

6.3.2 NIST LWC standardization process

In parallel, the NIST initiated several workshops on lightweight cryptography since 2015 that ended on the launch of a lightweight cryptography standardization process aiming to solicit, evaluate and standardize lightweight cryptography algorithms suitable for use in constrained environments in which the current standards are not acceptable. The point is to either develop new cryptographic solutions, enhance the existing ones or improve the implementations of current algorithms dedicated to authenticated encryption and optionally hash functions.
The proposed cryptographic solutions are authenticated encryption with associated data (AEAD) algorithms. This cryptographic function takes four byte-string as inputs and one byte-string as output. The four inputs are a variable-length plaintext, variable-length associated data, a fixed-length nonce, and a fixed-length key. The output is a variable-length ciphertext. The algorithms shall provide both authenticated encryption and decryption-verification. The cryptographic solutions should also provide confidentiality under adaptive chosen-plaintext attacks and integrity of the ciphertexts under adaptive forgery attempts. They are expected to maintain security when the uniqueness of the nonce is respected (any proof of security under nonce-reuse can be advertised as a feature). The minima in terms of size and security are a key of at least 128 bits, resisting attacks of at least 2^{112} computations on a classical computer in a single-key setting, and a key of 256 bits resisting attacks of at least 2^{224} computations. The recommended minimum sizes for nonce and tag are respectively 96 bits and 64 bits and the limit on the input sizes shall not be smaller than $2^{50}-1$ bytes. The entirety of the requirements are stated in [9].

57 candidates were proposed and 56 selected as Round 1 candidates. The focus during the first round was set on the cryptographic security of the submissions leading to two major selection criteria: maturation of the candidates and cryptanalysis of the candidates [224]. In September 2019, the NIST announced the candidates which made it to the round 2. For this second round, performance will play a larger role in the selection process, the goal being to obtain optimized versions fairly benchmarked on both software and hardware platforms.

Round 2 candidates			
ACE	ASCON	COMET	DryGASCON
Elephant	ESTATE	ForkAE	GIFT-COFB
Gimli	Grain-128AEAD	HYENA	ISAP
KNOT	LOTUS-AEAD / LOCUS-AEAD	mixFeed	ORANGE
Oribatia	PHOTON-Beetle	Pyjamask	Romulus
SAEAES	Saturnin	SKINNY-AEAD/HASH	SPARKLE
SPIX	SpoC	Spook	Subterranean 2.0
SUNDAE-GIFT	TinyJambu	WAGE	Xoodyak

6.4 Authentication through the sponge construction

6.4.1 The Sponge construction

The sponge construction was introduced by Bertoni et al. in [57] in 2007. It is based on a fixed-length permutation or transformation building a function mapping a variable-length input to a variable-length output. The construction was first used to build strong hash functions such as Keccak, the winner of the SHA-3 competition [12]. The sponge construction operates on a state of $b = r + c$ bits where b is called the width, r the rate and c the capacity. The data is padded and divided into words of r bits. The b bits of the state are initialized to zero. Then the construction proceeds in two phases. During the absorption, r bits are added to the state which goes through the f function. Once the whole data is absorbed, r bits are squeezed at a time with a call to the f function at each time. The last c bits of the state are never directly affected by the input block and are never output during the squeezing phase.

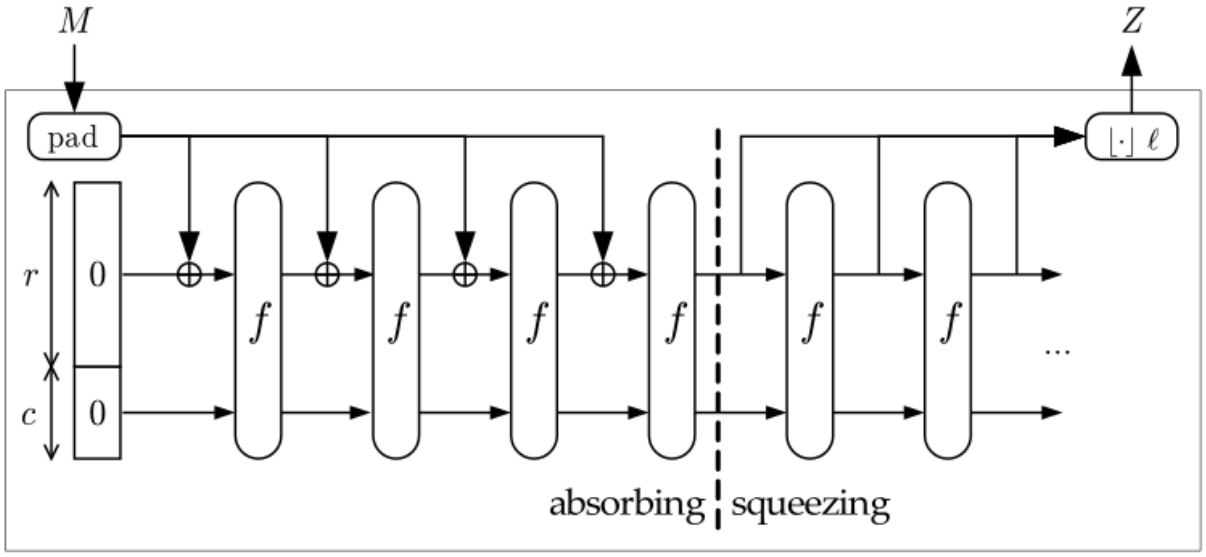


Figure 6.5: The sponge construction

From the sponge construction, keyed variants have been developed and have become a very popular mode of operation for a wide spectrum of cryptographic functions including message authentication codes computation, key derivation, pseudo-random functions, stream ciphers, Extendable-Output Functions (XOFs)[12] and authenticated encryption modes.
The sponge construction allows the possibility to build lightweight design for hash functions. Since the introduction of the design, numerous lightweight hash algorithms have indeed been proposed, such as SPONGENT[67], Quark [31] or PHOTON[136]. One main advantage when compared to classical Merkle-Damgård hash functions is the size of the state which is fairly smaller.
The keyed sponge principle also got adopted in Spritz, a new RC4-like stream cipher [197], and in 10 out of 57 submissions to the CAESAR[2] competition on authenticated encryption, with ASCON [106] as one of its best representatives for this competition.

The Random Oracle distinguishing advantage of the sponge construction when calling a random transformation f is upper bounded by:

$$1 - e^{\frac{-N(N+1)}{2^{c+1}}}$$

The Random Oracle distinguishing advantage of the sponge construction when calling a random permutation f is upper bounded by:

$$1 - e^{\frac{-N(N+1)}{2^{c+1}} - \frac{N(N-1)}{2^{r+c+1}}}$$

With N the cost of the computation, r the rate and c the capacity. If N is significantly smaller than 2^c, then these bounds become :

$$Adv(\mathcal{A}) < \frac{N(N+1)}{2^{c+1}}$$

and

$$Adv(\mathcal{A}) < \frac{N(N+1)}{2^{c+1}} - \frac{N(N-1)}{2^{r+c+1}}$$

6.4.2 Sponge-based authenticated encryption schemes

The Duplex construction Encrypting using the Sponge construction is usually done via the Duplex construction[59]. It is a stateful construction made of two interfaces: one for initialization and one for duplexing. The initialization interface is used to initialize the state (which can be all-zero) while the duplexing interface absorbs a message of at most r bits and squeezes $\leq r$ bits of the outer part.
The duplex construction also uses a fixed-length transformation or permutation f, a padding rule, and a bitrate r to build a cryptographic scheme.

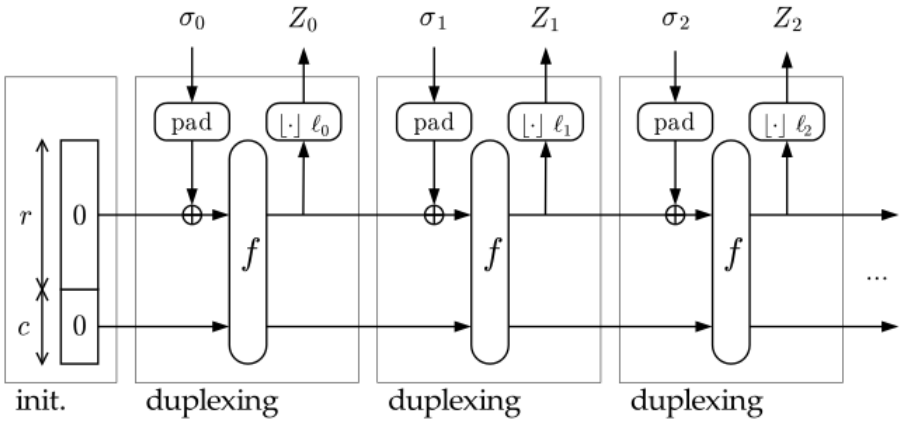


Figure 6.6: The Duplex construction

Existing security bounds The security of the Duplex mode is directly linked to the indifferentiability of the classical Sponge, leading to a $O(2^{c/2})$ security bound. Bertoni et al.[59] showed that the Duplex construction can be used for authenticated encryption in the form of SpongeWrap. This mode stands as the basis of the majority of Sponge-based submissions in the CAESAR competition and further work on improving sponge-based designs.
In the indifferentiability framework, Bertoni et al. [57] proved that the Sponge construction is secure up to the $\mathcal{O}(2^{c/2})$ birthday-type bound. The capacity part is left untouched throughout the evaluation of the Sponge construction: a violation of this paradigm would make the indifferentiability security result void.

A thorough analysis of the full-state message absorption keyed Sponge had to wait for Gazi, Pietrzak and Tessaro [126], who proved nearly tight security up to $\mathcal{O}(lq(q+N)/2^b + q(q+l+N)/2^c)$, where the adversary makes q queries of maximal length l, and makes N primitive calls. However, their analysis only applies to the fixed-output-length variant. The dubbed Full-state Keyed Sponge (FKS) implies the security of Donkey Sponge in the ideal permutation setting, and proves that it is secure up to approximately

$$\frac{2(ql)^2}{2^b} + \frac{2q^2l}{2^c} + \frac{\mu N}{2^k}$$

where k is the size of the key, and μ is a parameter called the multiplicity. This observation only works for $k \leq c$.

In the above mentioned result, for the integrity security the authors have assumed that the number of forgery blocks is limited. To be more specific, total number of forgery attempted blocks σ_v is restricted to satisfy the following: $q_p + \sigma_e + \sigma_v \leq 2^c/\sigma_v$, where σ_e is the total number of encryption query blocks and q_p is the number of permutation queries. The above equation clearly suggests that the number of decryption blocks should be at most $2^c/2$.
Jovanovic et al. [153] claimed that Sponge-based constructions for authenticated encryption can achieve a significantly higher bound with a security of $min\{2^{b/2}, 2^c, 2^k\}$, with b the permutation size, c the capacity and k the key size.

Beetle The authors of the Beetle Sponge mode wanted to develop a sponge-based mode of operation that could allow a better ratio rate capacity than already existing sponge-based mode of operations. They started from noticing from the CAESAR competition that even if sponge modes were known to provide lightweight solutions for authenticated encryption, the lightest candidate was COFB [81], which is block cipher-based. Consequently, the motivation to further understand the security bounds for sponge modes and trying to achieve even more lightweight designs. The Beetle mode of operation is combining a feedback function with a sponge to create a difference between the cipher text block and the rate part of the next feedback.

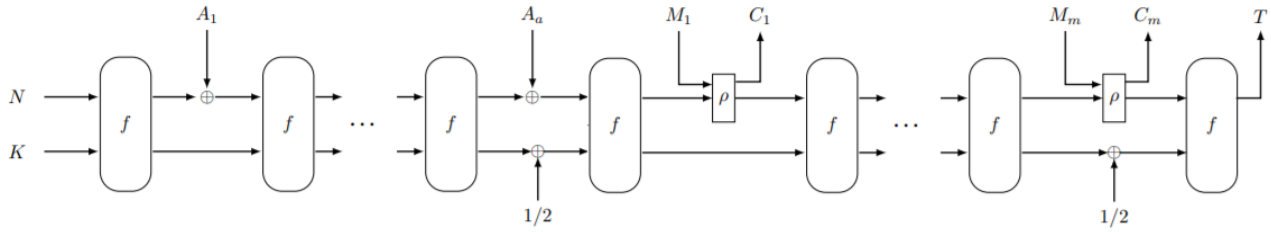


Figure 6.7: The Beetle mode of operation

For I_1 and I_2 two blocks of r bits, the feedback function called ρ is defined such as:

$$\rho(I_1, I_2) = (O_1, O_2), \text{where } O_1 = shuffle(I_1) \oplus I_2, O_2 = I_1 \oplus I_2.$$

The inverse function ρ' is

$$\rho'(I_1, O_2) = (O_1, I_2), \text{where } O_1 = shuffle(I_1) \oplus I_1 \oplus O_2, I_2 = I_1 \oplus O_2.$$

shuffle is defined as $shuffle(W) = (W[2], W[2] \oplus W[1])$ with $W \in \{0,1\}^r$ and $W[1]$ and $W[2] \in \{0,1\}^{r/2}$ such as $W = (W[1], W[2])$.
The authors claim a provable security up to $min\{c - log(r), b/2, r\}$ which offers a security of 121 bits for a 256-bit permutation with a capacity and a rate of 128 bits, thus allowing good performances and reducing hardware cost.

SpoC SpoC, Sponge with masked Capacity, is a permutation based mode of operation for authenticated encryption with associated data functionality. The high level design is inspired by the Beetle mode of operation.
Capacity is masked with data blocks instead of rate which improves the security and allows larger rate value per permutation call. It offers higher security guarantee with smaller states as compared to some of the previous AEAD designs based on the Sponge construction.
A 4-bit control signal which distinguishes different phases and does not require an extra call of permutation in the case of empty and partial data blocks.
The authors claim 128-bit security with state sizes 192 and 256, and corresponding rate values 64 and 128 bits, respectively.

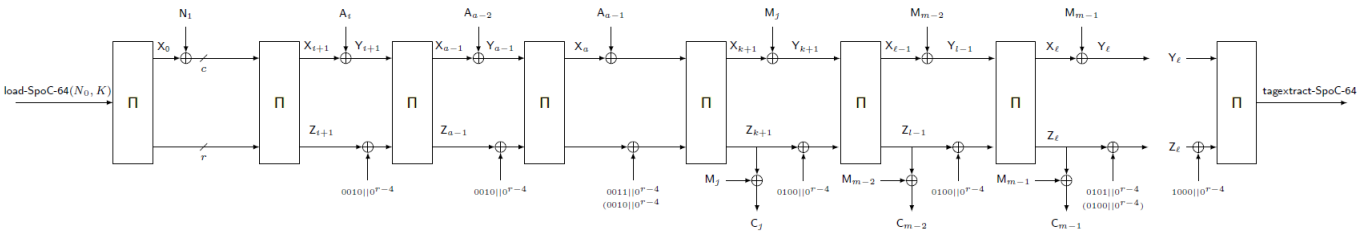


Figure 6.8: Sponge with masked Capacity

6.4.3 Side-channel resistant or resilient sponge mode by design

One main threat for lightweight platforms such as IoT, RFID tag or smartcard are physical attacks (or side-channel attacks) that can break the encryption and retrieve the key without mathematically

cryptanalysing the algorithm.
In order to face this threat some designers decided to look into the sponge resilience to leakages and physical resistance to known attacks.
In a presentation made at a workshop from the NIST LWC standardization process, François-Xavier Standaert presented his analysis on the leakage resistance of some candidates that made it to the round 2 [221].
The sponge modes can be divided into four steps, namely initialization, computation, finalization and tag verification.

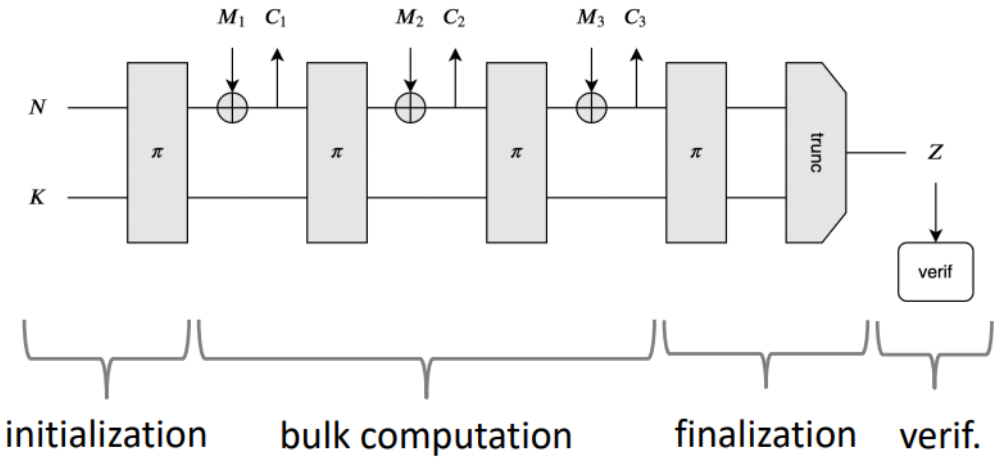


Figure 6.9: The four steps of AE sponge modes

Standaert divided the study on sponge modes in three cases : re-keyed modes, re-keyed modes with a strengthened initialization and finalization and two-passes re-keyed modes with a strengthened initialization and finalization.
Modes such as PHOTON-Beetle or Gimli can be seen as AE mode with a re-keying part, as the key is scrambled into the state along with the nonce through a permutation. Ascon or Spix for example have a strengthened initialization and finalization by XORing the key to the state. Spook and the S1P mode of operation also have this extra characteristics by using a tweakable block cipher as re-keying modules. Lastly, ISAP and the S2P mode of operation use a two-passes design in addition to the other properties to achieve even better security at the cost of an heavier and slower design.
The security targets are confidentiality, integrity for both plaintext and ciphertext, reduced leakage during encryption and/or decryption, nonce-misuse resistance or resilience, leakage-resistance or resilience and beyond birthday bond security in the single or multi-user paradigm.

ISAP Isap [105] is a family of permutation-based authenticated encryption schemes. The Isap instances are parameterized by the security parameter k, which defines the cryptographic security level of k bits, as well as a set of permutations with different round numbers. The authenticated encryption algorithm E gets as input a key $K \in \{0,1\}^k$, a nonce $N \in \{0,1\}^k$, associated data $A \in \{0,1\}^*$, and a message $M \in \{0,1\}^*$. It outputs a ciphertext $C \in \{0,1\}^{|M|}$ and a tag $T \in \{0,1\}^k$, where $|M|$ denotes the bitlength of M. The decryption algorithm D takes K, N, A, C, and T, and returns either the message M or an error $\perp$.
Isap is an encrypt-then-MAC design, where the same k-bit key is used for encryption and message authentication. Three different operations are used : the encryption IsapEnc, the message authentication IsapMAC and the re-keying function IsapRK used internally by both functions.
In these algorithms, p_H, p_B, p_E, p_K denote permutations updating an n-bit state S.
Rate r_H is applied for states in the unkeyed sponge and in the keyed sponge that are unlikely to be evaluated more than once for different outer parts with a fixed inner part, which means that r_H may be reasonably large. Rate r_B is applied for states in the keyed sponge that may be evaluated more than once, which means that we must bound the amount of leakage by limiting the total number of evaluations that may be made for that state. In each of the members of Isap, we set $r_H = n - 2k, c_H = 2k$ and $r_B = 1, c_B = n - 1$.
The rate r defines how the state S is split into an r-bit outer part S_r and a c-bit inner part S_c as $S = S_r||S_c = [S]^r[S]_c$, where $c = n - r$.

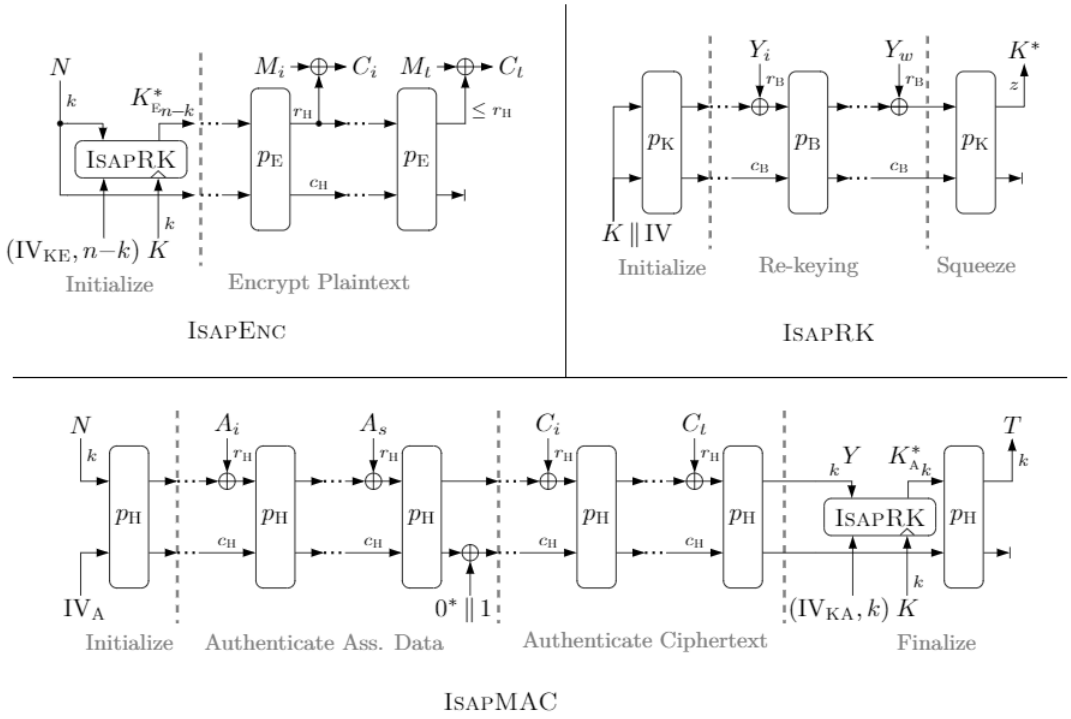


Figure 6.10: The ISAP mode of operation

The re-keying function IsapRK is called by IsapEnc and IsapMAC to generate session keys K_E^* and K_A^* to perform encryption and authentication, respectively. The function gets as input a k-bit key K, a k-bit string Y, a constant IV, and an output size z, where

$$(IV, z) = \begin{cases} (IV_{KE}, n-k), & \text{if called byIsapEnc} \\ (IV_{KA}, k), & \text{if called byIsapMAC} \end{cases}$$

and transforms these into a subkey K^* of size z bits. It is instantiated using permutations p_K, called in the beginning to process the master key K and p_B for all intermediate duplexes using a very small rate r_B.
Encryption is performed with the keyed sponge construction in the streaming mode. First, IsapRK is called to generate a subkey K_E^*. IsapEnc gets as input a k-bit key K, a k-bit nonce N, and an arbitrarily

large message M, and generates a ciphertext C of size $|M|$. It first calls IsapRK for encryption using the constant initial value $IV = IV_{KE}$ and $z = n - k$ in order to derive a $(n-k)$-bit subkey K_E^*. Once this subkey is generated, a regular sponge-based streaming mode using permutation p_E is evaluated at high rate r_H.
For message authentication, we use a sponge-based hash function to build a suffix-MAC. IsapMAC gets as input a k-bit key K, a k-bit nonce N, arbitrarily large associated data A, and arbitrarily large ciphertext C, and it outputs a tag T of size k bits. It starts by initializing the state as $N||IV_A$ and absorbing the non-secret inputs (A, C) in plain sponge mode using permutation p_H with high rate r_H. Note that domain separation between A and C is performed using the XOR of a single bit '1' to the inner part of the state. The resulting state S is then split into a k-bit value $[S]^k$ and a $(n-k)$-bit value $[S]^{n-k}$. The value $[S]^k$ is fed as input string to IsapRK to generate a subkey K_A^*, and a final call to the permutation p_H is made on input $K_A||[S]^{n-k}$ to obtain the k-bit tag T. For verification, the tag T' is re-computed in the same way from the received nonce N, associated data A, and ciphertext C, and compared with the received tag T.
ISAP is defined with two different permutations: Keccak-p[400] and Ascon-p.

The TETSponge/S1P The TETSponge/S1P scheme is a one-pass sponge-based mode for AEAD [135].

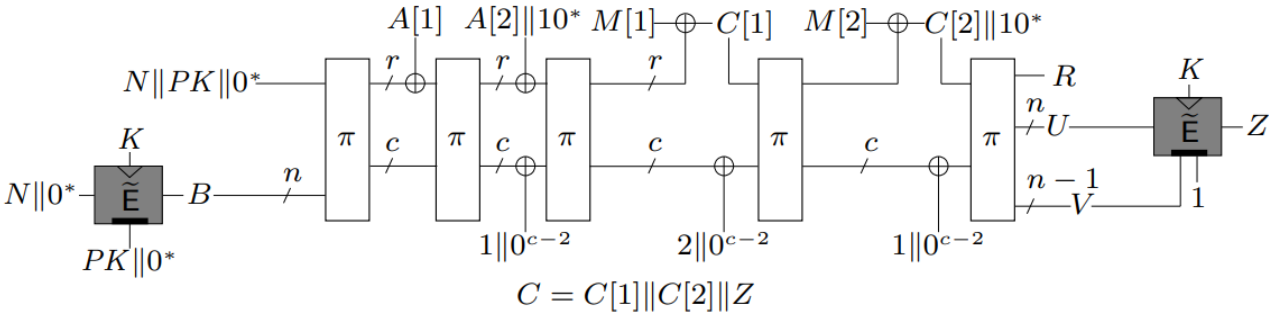


Figure 6.11: The TETSponge/S1P scheme

$\tilde{\mathrm{E}}$ is an (n, n, n)-TBC and Π is an $l = r + c$ bit keyless permutation. The key of S1P is $K||PK$, where $|K| = n$ and $|PK| = n_p$. We stress that only K has to be kept secret, but PK can be public. The secret key K is picked uniformly at random in $\{0,1\}^n$. The public key PK only needs to be distinct for each session. For simplicity, the focus is only put on uniform $PK \in \{0,1\}^{n_p}$. Let $n = |N|$ be the fixed length of the nonce. Is is only required that $n_p \leq r$, $n_N + n_p + n \leq r + c$, and $2n \leq r + c + 1$. Yet, it is recommended that $n_p \approx n$ and $c \approx 2n$ and it was actually chosen by the authors $n_p = n - 1$ and $c = 2n$ as this leads to a security up to $2^n/n^2$ complexity. There is no recommendation for n_N , but when $n = 128$ one could take $n_N = 96$ which is a standard choice.
Upon encrypting (N, A, M), the mode first derives an n-bit initial seed B from N, using a strongly protected TBC-call to $\tilde{\mathrm{E}}_K^{PK||0^*}(N||0)$. The initial seed B is then used as the key of the inner keyed duplex to process A and $M = M[1]||\ldots|||M[l]$ and produce $c = C[1]\ldots|||C[l]$. Note that 2 bits are used for domain separation, in order to distinguish M from A and mark if the last blocks of A and M are of full r bits or not. Let $U||V$ be the least significant $2n-1$ bits of the final state with $|U| = n$. As discussed, this truncated state is not immediately used as the tag. Instead, another strongly protected TBC-call is made, which generates the n-bit tag $Z = \tilde{\mathrm{E}}^{V||1}(U)$. The final ciphertext is $C[1]||\ldots||C[l]||Z$.
The Decryption. Upon decrypting (N, A, C), $C = C[1]||\ldots||C[l]||Z$, the mode first derives the initial seed B via $B = \tilde{\mathrm{E}}_K^{PK||0*}(N||0)$, and then runs the inner keyed duplex on A and $C[1]||\ldots||C[l]$ to derive M and the $2n-1$ bit truncated state $U||V$. Finally, it makes an inverse TBC call $U^* = (\tilde{\mathrm{E}}_K^{V||1})^{-1}(Z)$, and outputs M if and only if there is a match $U = U^*$. In such a way, invalid decryption only leaks meaningless random values U^* (instead of the correct tag) which is instrumental to achieve CIML2.

The TEDTSponge/S2P The main limitation of TETSponge/S1P pointed by its authors is that it does not achieve CCAmL2 security (which seems to be the price to pay to obtain a one-pass design).

With two passes, an Encrypt-then-MAC composition, the de facto standard choice for leakage-resilient Authenticated Encryption becomes possible.
The TBC-based key-derivation is used to start a keyed duplex for encrypting. Then a hash-then-TBC MAC function generates the tag from the nonce, the associated data, the ciphertext, and the public key. The keyless hash function is built upon the sponge function. The ideas of using hash-then-TBC for beyond $n/2$ security and hashing PK for beyond $n/2$ multi-user security are again inherited from [56]. In summary, S2P$[\Pi, \tilde{\mathrm{E}}]$ can be seen as a (more efficient) sponge-based variant of the TEDT TBC-mode, or an ISAP variant using a TBC to improve leakage-resilience (i.e., CIML2 and CCAmL2) security.
Formally, with the same conventions $n_p \leq r$, $n_N + n_p + n \leq r + c$, and $2n \leq r + c + 1$. The recommended parameters are $n_p = n - 1$ and $c = 2n$.

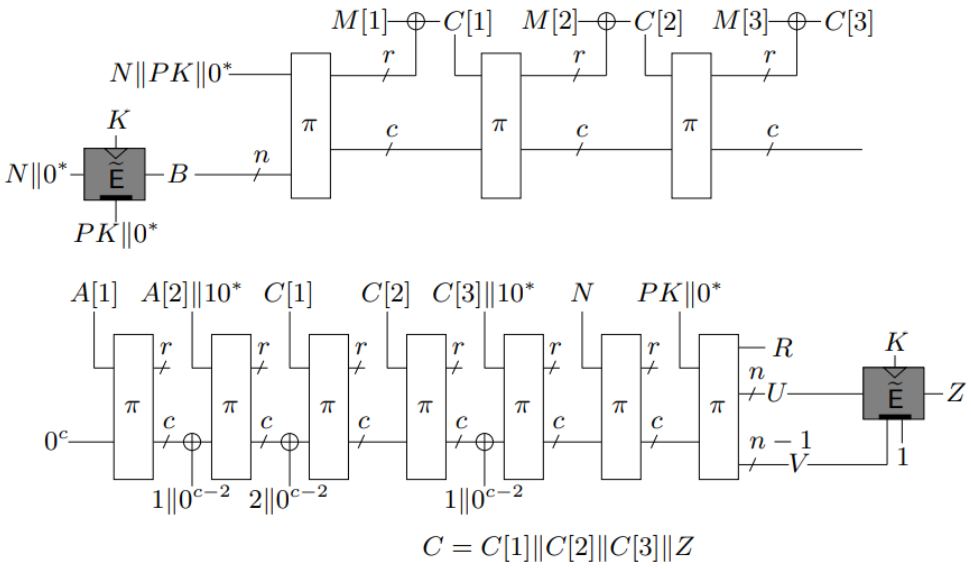


Figure 6.12: The TEDTSponge/S1P scheme

6.4.4 Focus on the permutation

Through the years, several sponge-based schemes have been published, each one presenting one or several enhancements on the mode, being either implementation efficiency, improvement of security bounds, enhancement or applicability of side-channel attacks and solutions to prevent them... But for the best of our knowledge, no work yet has been trying to compile all of these features and compare them. This is what we are trying to do here.

The permutation is the main component of the sponge construction. It is called numerous times during the encryption and MAC part. The majority of sponge-based AEAD schemes only rely on the permutation plus a few XOR for constants adding, the padding of messages and additional data.

Different state sizes can be found in the literature. From the NIST LWC standardization process, we can gather information on the size of the permutation used. The smallest one is used for SpoC with 192 bits and the largest one is Spook with 512 bits. We also study some interesting sponge-based hash scheme which can quite easily be turned into AEAD schemes.
Considering the security bounds given by the designers of the different sponge modes and emphasizing lightweight hardware implementations, we focus our work on permutations with size $b \in \{192, 256, 259, 320, 384\}$.

WAGE WAGE is an AEAD algorithm design introduced by Aagaard et al. [21]. The state is made of 259 bits and the permutation is based on the Welch-Gong stream cipher[206].

The state is divided into 37 7-bit words S_{36} to S_0. The rate part of the WAGE-AE-algorithm, called Sr is made of the last bit of S_{36} and $S_{35}, S_{34}, S_{28}, S_{27}, S_{18}, S_{16}, S_{15}, S_9, S_8$. The capacity part, of 195 bits is made of the rest of the state. The squeezing and absorbing phases of WAGE are shown in Fig. 8 with the rate part shaded in orange and the green part representing the capacity.

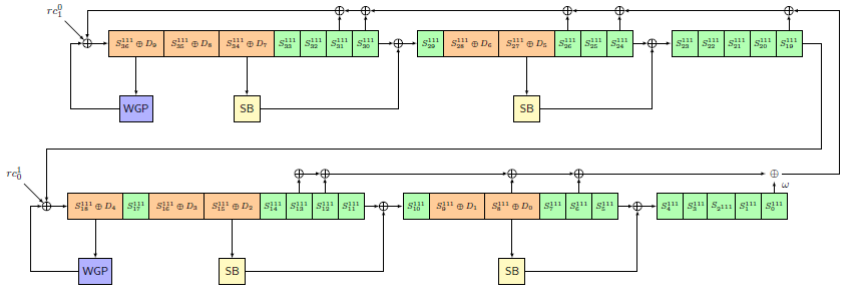


Figure 6.13: The squeezing and absorbing phases of WAGE

The WAGE permutation operates over the finite field $\mathbb{F}_{2^7}$. Its round function is constructed by tweaking the initialization phase of the Welch-Gong cipher with four 7-bit SBoxes added to achieve a faster confusion and diffusion. The core components of the round function are an LFSR, two Welch-Gong Permutations (WGP) and four SBoxes. The state is updated 111 times as described next.
The Welch-Gong permutation is defined by:
$\mathrm{WGP}(x) = x^{13}+(x^{13}+1)^{33}+(x^{13}+1)^{39}+(x^{13}+1)^{41}+(x^{13}+1)^{104}$.
The state is updated as follow:

$$
\begin{aligned}
f_b &= S_{31}^i \oplus S_{30}^i \oplus S_{26}^i \oplus S_{24}^i \oplus S_{19}^i \oplus S_{13}^i \oplus Si_{12} \oplus S_8^i \oplus S_6^i \oplus (\omega \otimes S_{26}^i) \\
S_4^{i+1} &= S_5^i \oplus SB(S_8^i \\
S_{10}^{i+1} &= S_{11}^i \oplus SB(S_{15}^i \\
S_{18}^{i+1} &= S_{19}^i \oplus WGP(S_8^i) \oplus rc_0^i \\
S_{23}^{i+1} &= S_{24}^i \oplus SB(S_8^i) \\
S_{29}^{i+1} &= S_{30}^i \oplus SB(S_{15}^i) \\
S_{36}^{i+1} &= f_b \oplus SB(S_{36}^i) \oplus rc_1^i \\
S_j^{i+1} &= S_{j+1}^i, j \in \{0, ..., 36\} \setminus \{4, 10, 18, 23, 29, 36\}
\end{aligned}
$$

with rc_0^i and rc_1^i round constants calculated with a LFSR of length 7.

The sLiSCP-light mode The sLiSCP family of lightweight cryptographic permutations [22] was proposed at SAC17. It is designed on the unified duplex construction to provide multiple cryptographic functions aiming for the lowest hardware cost as possible. The permutation used is based on the Simeck [237] block cipher. It follows a 4-bit sub-block Type-2 Generalized Feistel-like Structure (GFS) with unkeyed round-reduced Simeck as the round function. The sLiSCP-Light design is designed by tweaking the GFS structure into a Partial Substitution-Permutation Network, thus achieving better security properties while reducing the hardware cost of the design and enhancing its performances.

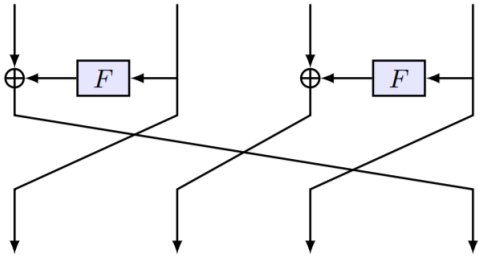


Figure 6.14: The sLiSCP permutation

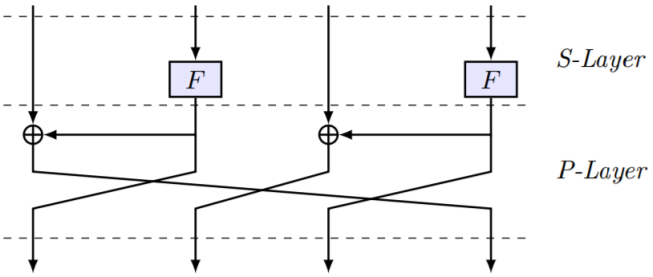


Figure 6.15: The sLiSCP-Light permutation

The sLiSCP-Light permutation operates on s step, computing b bits made of $4 \times m$ sub-blocks $(X_0^i, X_1^i, X_2^i, X_3^i)$, with i the step number (numerated from 0 to s-1).
In each step, the state is updated by a sequence of three transformations: SubstituteSubblocks(SSb), AddStepconstants(ASc), and MixSubblocks(MSb), such as:

$$(X_0^{i+1}, X_1^{i+1}, X_2^{i+1}, X_3^{i+1}) \leftarrow MSb \circ ASc \circ SSb(X_0^i, X_1^i, X_2^i, X_3^i)$$

The SubstituteSubblocks operation is a partial substitution layer of the SPN structure, where the non-linear operation is applied to half of the state, on two different m-bit sub-blocks. It applies the u-round iterated unkeyed Simeck box.

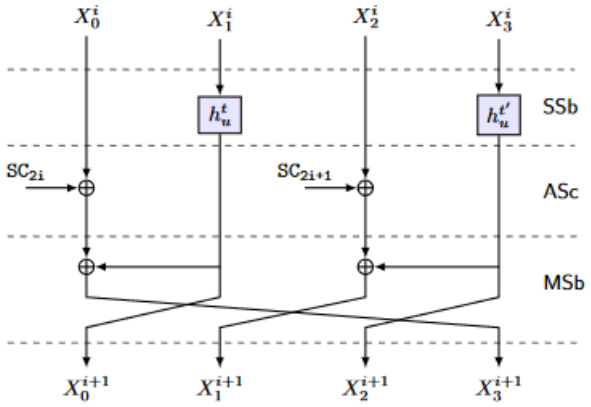


Figure 6.16: One sLiSCP-Light step

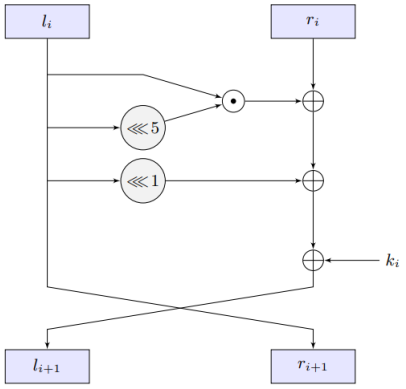


Figure 6.17: Simeck round

The usual round sub-key added at each round is here replaced by round constants.
The AddStepconstants operation consists of adding step constants to the two m-bit sub-blocks that did not go through the Simeck box.
The MixSubblocks layer applies the linear transformation that is used in the Type-2 GFS to the sub-blocks of the state. More precisely, each even indexed sub-block is replaced by the XOR of its initial value with its neighboring odd indexed sub-block. Then a sub-block cyclic left shift is applied.
The permutation is designed to work with 192 or 256 bits with 48 or 64 bit sub-blocks. It is used in the SPIX [20] and SpoC [19] candidates to the NIST LWC standardization process.

ACE ACE [13] is a 320-bit sponge-based algorithm that can be used for both Authenticated Encryption with Associated Data and hashing functions. Its name comes from the strongest card of the deck. Following the designers' idea, ACE is designed to achieve a balance between hardware cost and software efficiency.
In the case of ACE, the 320-bit state is divided into five 64-bit words, A, B, C, D and E. The Simeck box is applied to three of the five words, namely to words A, C and E. The permutations are made in 16 steps.
The sLiSCP-light uses a combination of a Type II Generalized Feistel Structure (GFS) and a round-reduced unkeyed Simeck box (SB). Each step consists of three transformations, namely, Substitute-Subblocks (SSb), AddStepconstants (ASc), and MixSubblocks (MSb). The non-linear operations are applied in the SSb, or SB. SBs consist of XORs, bitwise rotations, and a logical AND.
The three SBs on 64-bits each operate on a total of 192 bits out of 320 bits of state. Round constants are supplied to each SB at the start of each SSb transformation.
An SSb transformation requires 8 rounds, each of which executes in one clock cycle. Local state variables, as well as updated round constants, are stored during SSb transformations. The three round constants (rc_0, rc_1 and rc_2) and three step constants (sc_0, sc_1 and sc_2), each 8 bits, are implemented using look-up tables. The result of the C words through the second SB is then XORed to word B, and the same happens with words E and D. The three state constants are added to words B, D and E. Finally, the MixSubblocks part mixes the 5 words to their corresponding output, as shown in Figure 8.6. One sLiSCP-Light permutation is executed in $16 \times 8 = 128$ clock cycles.

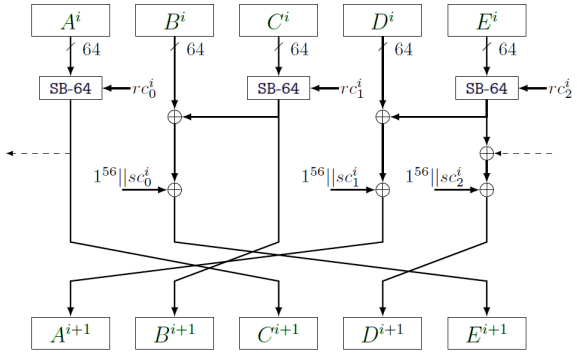


Figure 6.18: The ACE Permutation

The Simeck box shown in the figure below is an unkeyed independently parameterized variant of the round function of the Simon algorithm.

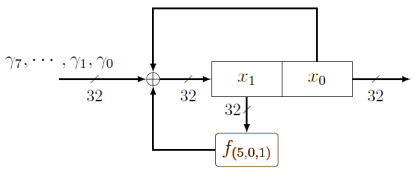
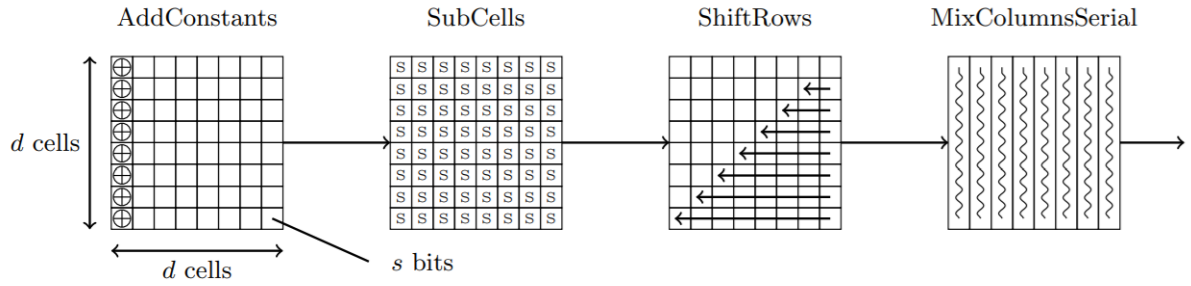


Figure 6.19: The Simeck Box

PHOTON The PHOTON permutation is an AES-like function, applied on an internal state of d^2 elements of s bit each. The state can be represented as a $(d \times d)$ matrix. The permutation is composed of N_r rounds, each made of four layers : AddConstants (AC), SubCells (SC), ShiftRows (ShR), and MixColumnsSerial(MCS). AddConstants consists in adding fixed values to the cells of the internal state, SubCells applies an s-bit Sbox to each of them, ShiftRows rotates the position of the cells in each row and MixColumnsSerial linearly mixes all the columns independently.



AddConstants: The constants have been chosen such that each of the N_r round computations are different, and such that classical symmetry between columns in AES-like designs is destroyed. The AddConstants layer prevents an input with all columns equal to maintain this property through the rounds. Only the first column of the internal state is involved, to limit the number of operations. The round constants can be generated by a combination of very compact Linear Feedback Shift Registers.

SubCells: Two types of Sboxes are used in PHOTON, the 4-bits SBOX from PRESENT [66] and the 8-bits SBOX from the AES. The Sboxes have mainly been chosen for their low hardware footprint. 4-bit Sboxes can be very compact in hardware while the acceptable upper limit on the cell size is $s = 8$. The choice was also made to save time for cryptanalysis by reusing already trusted and well analyzed components.

ShiftRows: Each row i will be rotated by i positions to the left, i counts from 0 to $d - 1$.

MixColumnsSerial: The starting point for choosing the matrix was that the matrix underlying the AES MixColumns function can not be implemented in a very compact way, mainly because the byte-serial implementation of this function is not compact. To maintain as much diffusion as possible and follow the idea used in AES, the matrix A is designed such as A^d is MDS, denoted as $Serial(Z_0, .., Z_{d-1})$ such as:

$$A = \begin{pmatrix} 0 & 1 & 0 & 0 & \cdots & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & \cdots & 0 & 0 & 0 & 0 \\ & \vdots & & & & & & \vdots & \\ 0 & 0 & 0 & 0 & \cdots & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & \cdots & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & \cdots & 0 & 0 & 0 & 1 \\ Z_0 & Z_1 & Z_2 & Z_3 & \cdots & Z_{d-4} & Z_{d-3} & Z_{d-2} & Z_{d-1} \end{pmatrix}$$

The size which interests us for this work is the P_{256} permutation, used in PHOTON-Beetle. This is a permutation on 256 bits, represented as a (8×8) matrix of 4 bits elements, computed 12 rounds. The matrix used for the MixColumnsSerial operation is $Serial(2, 3, 1, 2, 1, 4)$. The PRESENT Sbox in hexadecimal display:

$$SBOX_{PRESENT} = [0xc, 0x5, 0x6, 0xb, 0x9, 0x0, 0xa, 0xd, 0x3, 0xe, 0xf, 0x8, 0x4, 0x7, 0x1, 0x2]$$

The Mixing Matrix:

$$(A_{256})^8 = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 2 & 4 & 2 & 11 & 2 & 8 & 5 & 6 \end{pmatrix}^8 = \begin{pmatrix} 2 & 4 & 2 & 11 & 2 & 8 & 5 & 6 \\ 12 & 9 & 8 & 13 & 7 & 7 & 5 & 2 \\ 4 & 4 & 13 & 13 & 9 & 4 & 13 & 9 \\ 1 & 6 & 5 & 1 & 12 & 13 & 15 & 14 \\ 15 & 12 & 9 & 13 & 14 & 5 & 14 & 13 \\ 9 & 14 & 5 & 15 & 4 & 12 & 9 & 6 \\ 12 & 2 & 2 & 10 & 3 & 1 & 1 & 14 \\ 15 & 1 & 13 & 10 & 5 & 10 & 2 & 3 \end{pmatrix}$$

The PHOTON family members of hash functions proposes different possible parameters, as presented below:

	hash output n	internal state t	capacity c	output bitrate r'	Preimage	2nd-Preimage	Collision
PHOTON-80/20/16	80	100	80	16	2^{64}	2^{40}	2^{40}
PHOTON-128/16/16	128	144	128	16	2^{112}	2^{64}	2^{64}
PHOTON-160/36/36	160	196	160	36	2^{124}	2^{80}	2^{80}
PHOTON-224/32/32	224	256	224	32	2^{192}	2^{112}	2^{112}
PHOTON-256/32/32	256	288	256	32	2^{224}	2^{128}	2^{128}

Figure 6.20: The Photon Family members

LS-design Spook is designed in a mode of operation minimizing side-channel leakages. To achieve its security, it is designed with a tweakable block cipher Clyde-128 and a permutation, Shadow. Shadow is described with two state sizes: 512 bits, the primary candidate, and 384 bits. To reduce at maximum the hardware footprint of the whole algorithm, Clyde and Shadow use the same components.
Shadow is a variant of the LS-designs denoted as mLS-designs (multiple LS-designs) that mixes multiple LS-designs thanks to an additional diffusion layer. Such ciphers work on $n = (m \cdot s \cdot l)$-bit states, where m is the number of LS-designs considered, the size of the Sbox is s and the size of the Lbox is $2l$.
The state is denoted as x, each $(s \cdot l)$-bit substate corresponding to an LS-design as a bundle $x[b, *, *](0 \leq b < m)$, a bundle row as $x[b, i, *](0 \leq i < s)$ and a bundle column as $x[b, *, j](0 \leq j < l)$.
The internal representation of the data is an $(m \cdot s \cdot l)$-bit state but the cipher operates over bitstring inputs and outputs. The mapping between a bitstring B and a state x is $x[b, i, j] = B[b \cdot l \cdot s + i \cdot l + j]$.
In summary, Shadow-512 updates the n-bit state x by iterating N_s steps, each of them made of two rounds (A and B) that respectively apply an L-box to the rows of each bundle independently, and a diffusion layer mixing the rows of different bundles (on top of the S-box layer).

The Shadow permutation is defined as such:

$x \leftarrow \mu$
for $0 \leq \sigma < N_s$ do
  for $0 \leq b < m$ do
    for $0 \leq j < l$ do
      $x[b, *, j] = S(x[b, *, j])$;
    end for
    for $0 \leq i < s \setminus 2$ do
      $(x[2i, *], x[2i + 1, *]) = L(x[2i, *], x[2i + 1, *])$;
    end for
    $x \leftarrow x \oplus W(2 \cdot \sigma)$
  end for
  for $0 \leq b < m$ do
    for $0 \leq j < l$ do
      $x[b, *, j] = S(x[b, *, j])$;
    end for
  end for
  for $0 \leq i < s$ do
    for $0 \leq j < l$ do
      $x[*, i, j] = D(x[*, i, j])$;
    end for
  end for
  $x \leftarrow x \oplus W(2 \cdot \sigma + 1)$
end for
return x

where μ denotes the input, $W(r)$ are round constants, S and L are an s-bit Sbox and a $2l$-bit L-box and D is a m-bit diffusion layer.

The SBox used for the permutation is derived from the one used in Skinny [47]:

x	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$S(x)$	0	8	1	15	2	10	7	9	4	13	5	6	14	3	11	12

Table 6.1: Spook Sbox Table

The Lbox applies jointly to pairs of 32-bit words and has branch number 16 over those pairs. Denoting the two words on which it is applied as x and y:

$$(a, b) = L'(x, y) = \begin{pmatrix} circ(0xec045008) \cdot x^T \oplus circ(0x36000f60) \cdot y^T \\ circ(0x1b0007b0) \cdot x^T \oplus circ(0xec045008) \cdot y^T \end{pmatrix}$$

where *circ* denotes the circulant matrix whose first line is given in hexadecimal notation, so that the number $b = \sum_{i=0}^{31} 2^i b_i$ corresponds to the row vector $(b_0, ..., b_{31})$.
The diffusion layer is represented with the D matrix. It is the diffusion layer of the low-energy cipher Midori [39], which is based on a near-MDS 4×4 matrix defined as follows:

$$(a, b, c, d) = D(w, x, y, z) = \begin{pmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix} \cdot \begin{pmatrix} w \\ x \\ y \\ z \end{pmatrix}$$

ASCON permutation The Ascon permutation operates on a 320-bit state with a SPN-based round transformation that consists of three steps, namely p_c, p_s and p_l. The state is divided into five 64-bit words, $S = x_0||x_1||x_2||x_3||x_4$. The p_c operation is the addition of the round constant to the third 64-bit word of the state. The p_s is the substitution layer, each word being processed through a 5-bit Sbox such as:

x	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f	10	11	12	13	14	15	16	17	18	19	1a	1b	1c	1d	1e	1f
$\mathcal{S}(x)$	4	b	1f	14	1a	15	9	2	1b	5	8	12	1d	3	6	1c	1e	13	7	e	0	d	11	18	10	c	1	19	16	a	f	17

Figure 6.21: ASCON SBOX

The p_l represents the Linear Diffusion layer, defined with the following equation:

$$x_0 \leftarrow \sum_0(x_0) = x_0 \oplus (x_0 \ggg 19) \oplus (x_0 \ggg 28)$$
$$x_1 \leftarrow \sum_1(x_1) = x_1 \oplus (x_1 \ggg 61) \oplus (x_1 \ggg 39)$$
$$x_2 \leftarrow \sum_2(x_2) = x_2 \oplus (x_2 \ggg 1) \oplus (x_2 \ggg 6)$$
$$x_3 \leftarrow \sum_3(x_3) = x_3 \oplus (x_3 \ggg 10) \oplus (x_3 \ggg 17)$$
$$x_4 \leftarrow \sum_4(x_4) = x_4 \oplus (x_4 \ggg 7) \oplus (x_4 \ggg 41)$$

These steps are iterated 12 times when the permutation is used for initialization, finalization or computing of additional data and six or eight times for message computation, depending on the targeted security level.

SHAMASH SHAMASH is an authenticated cipher based on a 320-bit permutation. The permutation divides the state into five 64-bit words and computes them with three layers of operation, namely a round constant addition to the first word of the state, a substitution layer and a diffusion layer.
The round constants are $(11 \times i) \oplus 0xff$ where i is the round number.
The substitution step, such as ASCON, consists in the passage of the state through sixty-four 5-bit to 5-bit Sboxes.

x	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
S[x]	16	14	13	2	11	17	21	30	7	24	18	28	26	1	12	6
x	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
S[x]	31	25	0	23	20	22	8	27	4	3	19	5	9	10	29	15

Table 6.2: The SHAMASH 5-bit Sbox

The diffusion layer consists of a series of rotations and XORs, in three steps. First, each 64-bit word is rotated twice to the right and the results XORed with the original word:

$$w \rightarrow w \oplus Rot(w, a) \oplus Rot(w, b)$$

With a, b, constants, different for each word. For any constants $a, b \neq 0$, $a \neq b$ the above transformation is invertible. The inverse is a function of the form

$$w \rightarrow \bigoplus_{i=0}^{t} Rot(w, a_i)$$

where t is the number of terms of the inverse.
The constants a and b have been chosen for each word so that the inverse of each of the five transformations has the highest value as possible, i.e. 43 and 37:

	a	b	t
W_0	43	62	37
W_1	21	46	37
W_2	58	61	43
W_3	57	63	37
W_4	3	26	37

Then the bits are mixed vertically, following:

$$W_i = \begin{cases} W_i \oplus W_3 \oplus W_4, & \text{if } i = 0, 1 \text{ or } 2 \\ W_i \oplus W_0 \oplus W_1 \oplus W_2, & \text{if } i = 3 \text{ or } 4 \end{cases}$$

Lastly, the 4 first words W_i are rotated from $2i + 1$ bytes to the right, while the last word is left untouched.
This whole process is iterated nine times to compute the permutation.

Sycon Sycon v1.0 [176] was first designed to enter the NIST LWC standardization process but did not make it into the round 2. After noticing that some parts of the design were not optimal (for example, some of the differential trail claimed in the submission document were not correct), the design goal was to design a permutation, lighter as the permutation from Ascon but with the same security and efficiency. Sycon [176] is an AEAD scheme based on the MonkeyDuplex construction. The design of the permutation is based on the Substitution-Permutation Network, applying first a substitution layer (Sboxes in parallel) that induces the confusion property and then a linear diffusion layer to induce diffusion.
The Sycon permutation is an iterative permutation on 320 bits. The state is first divided into sixty-four 5-bit words, each processed trough a 5-bit Sbox, in parallel. This operation is named Sbox (SB).

x	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
S[x]	8	19	30	7	6	25	16	13	22	15	3	24	17	12	4	27
x	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
S[x]	11	0	29	20	1	14	23	26	28	21	9	2	31	18	10	5

Table 6.3: The Sycon 5-bit Sbox

The diffusion layer is a linear transformation that is applied independently on five 64-bit subblocks s_i, $0 \leq i \leq 4$. The linear transformation has the form $\mathcal{L}(x) = (x \oplus (x \lll u) \oplus (x \ll 2u)) \lll t$ where $0 \leq u, t \leq 63$ are positive integers.

$$\begin{aligned} s_0 &\leftarrow (s_0 \oplus (s_0 \lll 59) \oplus (s_0 \lll 54)) \lll 40 \\ s_1 &\leftarrow (s_1 \oplus (s_1 \lll 55) \oplus (s_1 \lll 46)) \lll 32 \\ s_2 &\leftarrow (s_2 \oplus (s_2 \lll 33) \oplus (s_2 \lll 2)) \lll 16 \\ s_3 &\leftarrow (s_3 \oplus (s_3 \lll 21) \oplus (s_3 \lll 42)) \lll 56 \\ s_4 &\leftarrow (s_4 \oplus (s_4 \lll 13) \oplus (s_4 \lll 26)). \end{aligned}$$

The last step of the permutation is the AddRoundConst (RC). The round constants are generated using a 4-bit LFSR, each state of the LFSR serving as a distinct round constant. The four bits thus generated are concatenated with a 60-bit constant value: $0xaaaaaaaaaaaaaaa0$.
These steps are iterated twelve times when the permutation is used for initialization, finalization or computing of additional data and six or seven times for message computation, depending on the security level targeted.

Keccak permutation The Keccak-p permutations are derived from the Keccak-f permutations [12] and have a tunable number of rounds. A Keccak-p permutation is defined by its width $b = 25 \times 2^l$, with $b \in \{25, 50, 100, 200, 400, 800, 1600\}$, and its number of rounds n_r.
Keccak-$p[b, n_r]$ consists in the application of the last n_r rounds of Keccak-$f[b]$. When $n_r = 12 + 2l$,

Keccak-$p[b, n_r]$ =Keccak-$f[b]$.
The permutation Keccak-$p[b, n_r]$ is described as a sequence of operations on a state a that is a three-dimensional array of elements of GF(2), namely $a[5, 5, w]$, with $w = 2^l$.
The expression $a[x, y, z]$ with x, $y \in \mathbb{Z}_5$ and $z \in \mathbb{Z}_w$, denotes the bit at position (x, y, z). Expressions in the x and y coordinates should be taken modulo 5 and expressions in the z coordinate modulo w
Keccak-$p[b, n_r]$ is an iterated permutation, consisting of a sequence of n_r rounds R, indexed with i_r from $12 + 2l - n_r$ to $12 + 2l - 1$. Note that i_r, the round number, does not necessarily start from 0. A round consists of five steps:

$$R = \iota \circ \chi \circ \pi \circ \rho \circ \theta$$

with
$\theta : a[x, y, z] \leftarrow a[x, y, z] + \sum_{y'=0}^{4} a[x - 1, y', z] + \sum_{y'=0}^{4} a[x + 1, y', z - 1]$, $\rho : a[x, y, z] \leftarrow a[x, y, z - (t + 1)(t + 2)/2]$,
with t satisfying $0 \leq t \leq 24$ and $\left(\begin{smallmatrix} 0 & 1 \\ 2 & 3 \end{smallmatrix}\right)^t \left(\begin{smallmatrix} 1 \\ 0 \end{smallmatrix}\right) = \left(\begin{smallmatrix} x \\ y \end{smallmatrix}\right)$ in $GF(5)^{2\times 2}$, or $t = -1$ if $x = y = 0$, $\pi : a[x, y] \leftarrow a[x', y']$ with $\left(\begin{smallmatrix} x \\ y \end{smallmatrix}\right) = \left(\begin{smallmatrix} 0 & 1 \\ 2 & 3 \end{smallmatrix}\right) \left(\begin{smallmatrix} x' \\ y' \end{smallmatrix}\right)$,
$\chi : a[x] \leftarrow a[x] + (a[x + 1] + 1)a[x + 2]$,
$\iota : a \leftarrow a + RC[i_r]$.

Spongent Spongent is a sponge construction based on a PRESENT-type permutation [66]. It produces a n-bit hash value for any input of a finite number of bits using a permutation π_b operating on a state of b bits. The permutation is a N-round iteration of the transformation of the b-bit state. It is similar to the PRESENT round function but in a wider version. Moreover, instead of the key addition a round constant is XORed to the state. The round constants are computed using a LFSR. The substitution and permutation layers of spongent are the same as in PRESENT but defined for a wider state.

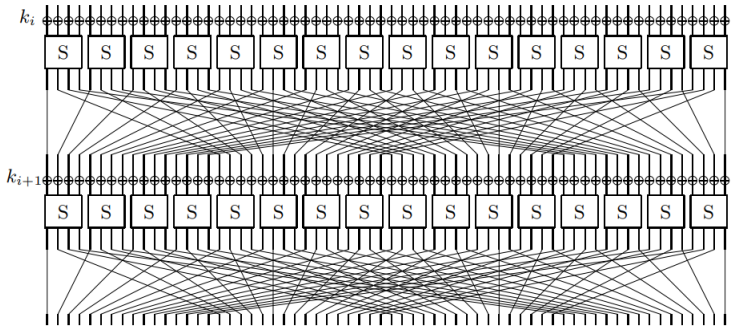


Figure 6.22: PRESENT round

The substitution layer uses a 4-bit Sbox which is applied $b/4$ times in parallel:

x	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
S(x)	E	D	B	0	2	1	4	F	7	A	8	5	9	C	3	6

Table 6.4: The PRESENT Sbox

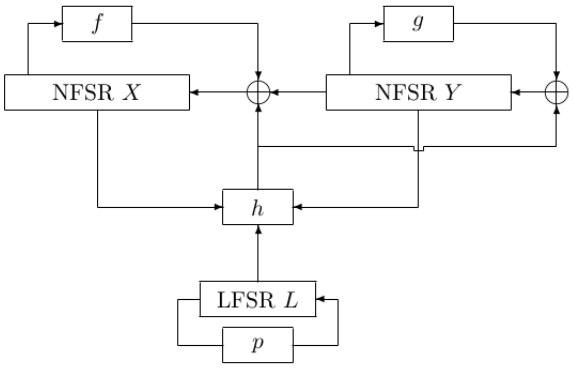
The permutation layer is an extension of the inverse PRESENT bit-permutation and moves bit j of the state to position $P_b(j)$ following:

$$P_b(j) = \begin{cases} j.b/4 \pmod{b-1} & \text{if } j \in \{0, ..., b-2\} \\ b-1 & \text{if } j = b-1 \end{cases}$$

Spongent is proposed with 5 sets of parameters, each one resulting on a different security level:

	n (bit)	b (bit)	c (bit)	r (bit)	R number of rounds	security(bit) preimage	2nd preimage	collision
SPONGENT-88	88	88	80	8	45	80	40	40
SPONGENT-128	128	136	128	8	70	120	64	64
SPONGENT-160	160	176	160	16	90	144	80	80
SPONGENT-224	224	240	224	16	120	208	112	112
SPONGENT-256	256	272	256	16	140	240	128	128

QUARK QUARK is a hash function with a hardware oriented permutation inspired by the lightweight block ciphers KTANTAN and KATAN and the hardware oriented stream cipher Grain. The smallest version, U-QUARK produces a 136 bits long digest, D-QUARK produces a 176 bits long digest and the longest version, S-QUARK a 256 bits long digest.
The update function computes b bits by loading each half of the state into a distinct NLSFR of length $b/2$ and then computing it $4\times b$ times. The NLSFRs are connected to each other and to a small LFSR of length $log(4b)$. The functions f, g and h are boolean functions chosen for their non-linearity, resilience, algebraic degree and density. f and g are identical for all versions and borrowed from Grain-v1 while h depends on the instance.



The Quark instances can easily be modified to operate in the duplex construction to allow the realization of functions as authenticated encryption or reseedable pseudo-random generators. C-QUARK is such a version, dedicated to authenticated encryption.

GIMLI GIMLI [54] is a 384-bit permutation, designed for cross-platform performance. It also aims to be energy-efficient in hardware, and it provides side-channel-protected implementations for hardware platforms and for micro-controllers.
Gimli applies a sequence of rounds to a 384-bit state. The state is represented as a parallelepiped with dimensions $3 \times 4 \times 32$ or, a 3×4 matrix of 32-bit words. A column j is a sequence of 96 bits and a row i is a sequence of 128 bits.

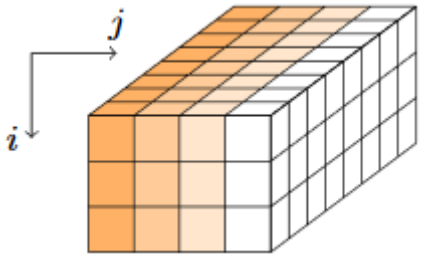


Figure 6.23: Representation of the GIMLI state

Each round is a sequence of three operations: a non-linear layer (96-bit SP-box applied to each column) (1), in every second round, a linear mixing layer (2) ; and in every fourth round, a constant addition (3). The SP-box consists of three sub-operations: rotations of the first and second words, a 3-input nonlinear T-function and a swap of the first and third words. The linear layer consists of two swap operations: Small-Swap and Big-Swap. Small-Swap occurs every four rounds starting from the first round, Big-Swap every four rounds starting from the third round. There are 24 rounds in Gimli, and constants are added to the state on rounds 24, 20, 16, 12, 8 and 4. XOR the round constant $0x9e377900 \oplus r$ to the first state words.

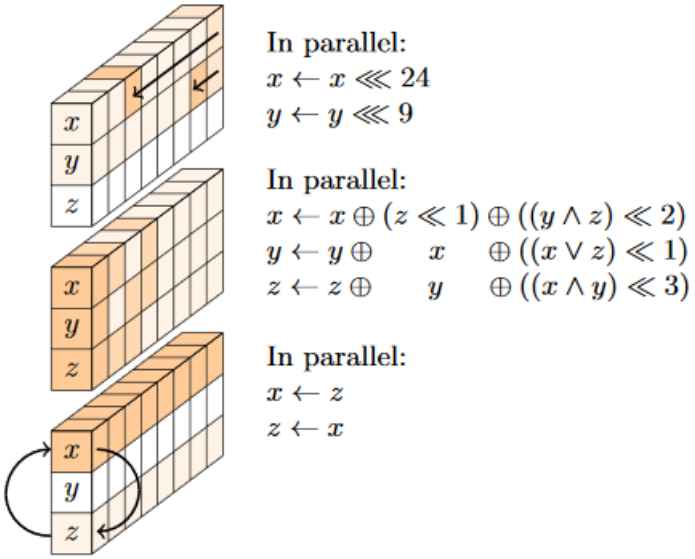


Figure 6.24: Gimli SP-box

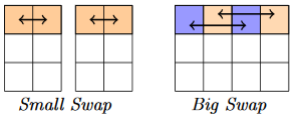


Figure 6.25: Gimli linear layer

Part III

My contributions

For the duration of this thesis, I have been working on several topics upon which I could bring my contribution. Three of them are quite related to the topics I was working on for HENSOLDT France, which I will present here. The last one is about a new protocol for blockchain, trying to initiate a cryptocurrency rewarding useful works, that I will present in appendix of this thesis.

The first paper is entitled Non-Triangular Self-Synchronizing Stream Ciphers, and was written in collaboration with Julien Francq, Paul Huynh, Philippe Guillot, Gilles Millerioux and Marine Minier. It was published in IEEE Transactions on Computers [123]

Abstract We propose here an instantiation, called Stanislas, of a dedicated Self Synchronizing Stream Ciphers (SSSC) involving an automaton with finite input memory using non-triangular state transition functions. Previous existing SSSC are based on automata with shifts or triangular functions (T–functions) as state transition functions. Our algorithm Stanislas admits a matrix representation deduced from a general ans systematic methodology called Linear Parameter Varying (LPV). This particular representation comes from the automatic theory and from a special property of dynamical systems called flatness.
Hardware implementations and comparisons with some state-of-the-art stream ciphers on Xilinx FPGAs are presented. It turns out that Stanislas provides bigger throughput than the considered stream ciphers (synchronous and self-synchronizing) when straightforward implementations are considered. Moreover, its synchronization delay is much smaller than the SSSC Moustique (40 clock cycles instead of 105) and the standard approach CFB1-AES128 (40 clock cycles instead of 128).

My main contribution for this work was the implementation of the Stanislas algorithm on FPGA and some others stream ciphers for comparisons. The goal was rather to have an implementation developed in the same spirit as those from the other algorithms found in the literature than to provide a fully optimized implementation, in order to get a fair comparison. An optimized version could be developed later if the algorithm finds its use in a practical application. In addition, we studied some solutions for masking the algorithm but the results were not conclusive so we decided not to disclose it in this paper.

The second work followed the NIST LWC standardization process that aims to define the new standards for lightweight authenticated encryption. For this purpose, I implemented and made a comparative study of some sponge-based candidates that made it into the second round of the competition. This work was made in collaboration with Louis Goubin, who helped me analyze the proper conclusions and perspectives. This work was realized in parallel to the NIST LWC standardization process and similar results have been published by others teams of cryptographers, including one from the CERG (Cryptographic Engineering Research Group) that is the current reference in cryptographic implementations on FPGA. In order to be resubmitted, further improvements will be required for an optimized FPGA implementation of PHOTON-Beetle, which has been accepted in the third round of the LWC standardization process.

Abstract Achieving security while maintaining a low hardware footprint is one of the new challenges in cryptography. Small objects like IoT (Internet of Things) have seen their use skyrocket those past years and they need secure solutions adapted to their size, thus the need to develop or adapt lightweight yet robust cryptographic solutions. This calls for improved designs and implementations of constructs such as Authenticated Encryption with Associated Data (AEAD) which can ensure confidentiality, integrity, and authenticity, all in one algorithm.
The CAESAR competition[2] with ASCON [106] as first recommendation for lightweight use cases for AEAD have shown that sponge-based design could be very practical to design lightweight yet robust solutions for those functions.
The U.S. National Institute of Standards and Technology (NIST) has begun a years-long effort called the Lightweight Cryptography (LWC) standardization process in order to evaluate lightweight AEAD and hash algorithms to update the U.S. federal standards accordingly. Several sponge-based candidates were presented.
The goal here is to implement and compare some of these candidates to distinguish which are the

most interesting bricks to construct lightweight sponge-based candidates depending on the performance mainly targeted. We therefore implemented three sponge-based AEAD candidates on FPGA: ACE[13], WAGE[21] and PHOTON-BEETLE[40]. These implementations are compliant with the newly-released hardware (HW) applications programming interface (API) for lightweight cryptography and come to complet the work initiated in [192].

The third and last work I present in this part is an attack on the SIV mode of operation. The goal of this attack was to introduce an attack which is feasible if the protocol is not well instantiated, i.e., the number of allowed messages while using the same key is not defined. This work was realized in collaboration with Louis Goubin. While we were working on a new mode of operation, we found interesting to test the security of existing an one and found this attack. This paper has not been submitted yet at the time this thesis is written, but is intended to be soon.

Abstract SIV mode of operation is a mode of operation described to obtain deterministic authenticated encryption. This means that the mode of operation reaches security while not using any initialization vector. Algorithms designed to achieve this can be viewed as misuse-resistant modes of operations. In this work we try to show the characteristics that need to be given for a proper use of this mode of operation for secure protocol, i.e. no more than $2^{64} - 1$ messages per key. We describe an attack on the mode of operation that can be realized if this characteristic is not maintained.

Lastly I have been able to contribute to a paper on a new protocol for blockchain, a Useful-Work protocol, that I introduce in Appendix A. This work is not related to the main topic of this thesis but was a very interesting starting point for me in the vast universe of the blockchain.

At the time this thesis is written, the paper concerning the Useful-Work protocol have been submitted to acceptance for publication.

Chapter 7

A Self-Synchronizing Stream Cipher : Stanislas

7.1 Introduction

Self-Synchronizing Stream Ciphers (SSSC) were patented in 1946. The basic principle of such ciphers is to encrypt every plaintext symbol with a transformation that only involves a fixed number of previous ciphertext symbols. Therefore, every ciphertext symbol is correctly deciphered provided that previous symbols have been properly received. This self-synchronization property has many advantages and is especially relevant to group communications. In this respect, since 1960, specific SSSC have been designed and are still used to provide bulk encryption (for Hertzian line, RNIS link, etc.) in military applications or governmental radio mobile networks [179, 203].
The canonical form of the SSSC combines a shift register, which acts as a state register with the ciphertext as input, together with a filtering function that provides the running key stream. The cryptographic complexity of the canonical form of the SSSC lies in the filtering function.
In the early 90s, studies have been performed [179, 93] to propose secure designs of SSSCs. These works have been followed by effective constructions ([137, 207, 95]). What motivates the present proposal for a new SSSC is that, till now, all of these SSSC schemes have been broken ([150, 151, 152, 158, 162]). And up to our knowledge, since 10 years, no other proposals of SSSCs has been made. Clearly, SSSC can be naturally built using a block cipher by applying the Cipher Feedback (CFB) mode. However, the computational cost of CFB is one full block cipher operation per digit. So for single-bit digits, it is n times less efficient than synchronous stream encryption modes such as Output Feedback (OFB) or Counter (CTR) Mode, with n the block length. For example, AES in single-bit CFB mode (as defined as "CFB1-AES128" in NIST SP 800-38a [111]) is 128 times less efficient than AES in counter mode. As a consequence, it seems interesting to propose dedicated SSSC and consider them as a category of primitives of their own.
The aim of the present chapter is to propose a new framework, along with an instantiation called Stanislas, to design SSSC. The design approach is based on both the special feature of Finite State Machines admitting a matrix representation, called LPV (Linear Parameter Varying) automata and on a property of dynamical systems named flatness coming from control theory. Flatness is interpreted in terms of the structure of a graph associated to the automaton from which self-synchronization can be easily characterized. The matrix representation is a generalization of Rational Linear Finite State Machines or Feedback with Carry Shift Registers proposed in [28, 29]. The use of flatness for the sake of cryptography has been first proposed in [108]. It has been shown that flatness characterizes the self-synchronization property and that it allows to design automata which can be more general than T–functions as it was the case over the past years (see [94, 95] as examples). One of the benefits of this approach is that we could introduce nonlinearities with proved properties in the matrix representation, what a shift register does not always permit. Due to this peculiarity, the class of admissible automata to act as SSSCx is thereby enlarged.
In the present chapter, a complete cipher, called Stanislas (for Secure Transmission Algorithm with Non triangular Iterative Structure Looking After Self-synchronization) and designed from the LPV framework, is described. The key schedule, the design rationale and the security analysis are provided. Next, hardware implementation results on Xilinx FPGA platforms of Stanislas are performed

and compared with some state-of-the art competitors: some Synchronous Stream Ciphers coming from eSTREAM portfolio (Trivium [101] and Grain [138]), one SSSC (Moustique) and the only known feedback mode (the NIST-standardized CFB1-AES128). Interestingly, Stanislas provides the highest throughput on Xilinx Spartan-6 XC6SLX75T FPGAs compared to its stream ciphers and SSSC competitors, implemented in a straightforward manner. Moreover, with the same comparison conditions, its synchronization delay is much smaller than the SSSC Moustique (40 clock cycles instead of 105) and the standard approach CFB1-AES128 (40 clock cycles instead of 128), which provides a decisive advantage for applications when low-latency synchronization is required (e.g., Telecom).
The chapter is organized as follows. Section 7.2 recalls the theoretical results linking together flatness and SSSC. Section 7.3 presents the new SSSC Stanislas. The design rationale and the security analysis are detailed in Section 7.4. Finally, Section 7.5 provides hardware implementation results and comparisons. Section 7.6 concludes this chapter.

7.2 Theoretical Foundations and Flatness

After few generalities on stream ciphers, it is recalled in this section the main results of [108] concerning the design of SSSC based on the property of flatness.

7.2.1 Generalities on Stream Ciphers

For a stream cipher, it must be given an alphabet A, that is, a finite set of basic elements named symbols. The set A stands in this paragraph as a general notation without any specific alphabet. Typically, A could be composed of 1 or several bits elements. Hereafter, the index $t \in \mathbb{N}$ will stand for the discrete-time. On the transmitter part, the plaintext (also called information or message) $m \in \mathcal{M}$ ($\mathcal{M}$ is the message space) is a string of plaintext symbols $m_t \in A$. Each plaintext symbol is encrypted, by means of an encryption (or ciphering) function e, according to:

$$c_{t+r} = e(z_{t+r}, m_t), \tag{7.1}$$

where $z_t \in A$ is a so-called keystream (or running key) symbol delivered by a keystream generator. The function e is invertible for any prescribed z_t. The resulting symbol $c_t \in A$ is the ciphertext symbol. The integer $r \geq 0$ stands for a potential delay between the plaintext m_t and the corresponding ciphertext c_{t+r}. This is explained by computational or implementation reasons, for instance pipelining (see [94] for example). Consequently, for stream ciphers, the way how to encrypt each plaintext symbol changes on each iteration. The resulting ciphertext $c \in \mathcal{C}$ ($\mathcal{C}$ is called the ciphertext space), that is the string of symbols c_t, is conveyed to the receiver through a public channel.
At the receiver side, the ciphertext c_t is deciphered according to a decryption function d which depends on a running key $\widehat{z}_t \in A$ delivered, similarly to the cipher part, by a keystream generator. The decryption function d obeys the following rule. For any two keystream symbols $\widehat{z}_{t+r}, z_{t+r} \in A$, it holds that

$$\widehat{m}_{t+r} := d(c_{t+r}, \widehat{z}_{t+r}) = m_t \text{ whenever } \widehat{z}_{t+r} = z_{t+r}. \tag{7.2}$$

Equation (7.2) means that the running keys z_t and $\widehat{z}_t$ must be synchronized for a proper decryption. The generators delivering the keystreams are parametrized by a secret key denoted by $K \in \mathcal{K}$ ($\mathcal{K}$ is the secret key space). The distinct classes of stream ciphers (synchronous or self-synchronizing) differ each other by the way on how the keystreams are generated and synchronized. Next, we detail the special class of stream ciphers called Self-Synchronizing Stream Ciphers.

7.2.2 Keystream Generators for Self-Synchronizing Stream Ciphers

A well-admitted approach to generate the keystreams has been first suggested in [179]. It is based on the use of so-called finite state automata with finite input memory as described below. This is typically the case in the cipher Moustique [158]. At the ciphering side, the automaton delivering the keystream takes the form:

$$\begin{cases} x_{t+1} = f_K(x_t, m_t), \\ z_{t+r} = h_K(x_t) \end{cases} \tag{7.3}$$

where $x_t \in A$ is the internal state, f is the next-state transition function parametrized by $K \in \mathcal{K}$. As previously stressed, the delay r is introduced to cope with special situations, in particular when the computation of the output (also called filtering) delivered by the function h involves r successive operations processed at time instants $t, \ldots, t+r$. Those operations will be here matrix multiplications as detailed later in Equation (7.14). Substituting m_t by its expression (7.2) yields an automaton described by

$$\begin{cases} x_{t+1} = g_K(x_t, c_{t+r}), \\ z_{t+r} = h_K(x_t) \end{cases} \tag{7.4}$$

If such an automaton has finite input memory, it means that, by iterating (7.4) a finite number of times, there exists a function ℓ_K and a finite integer M such that

$$x_t = \ell_K(c_{t+r-1}, \ldots, c_{t+r-M}), \tag{7.5}$$

and thus,

$$z_{t+r} = h_K(\ell_K(c_{t+r-1}, \ldots, c_{t+r-M})). \tag{7.6}$$

Actually, the fact that the keystream symbol can be written in the general form

$$z_{t+r} \quad = \quad \sigma_K(c_{t-\ell}, \ldots, c_{t-\ell'}), \tag{7.7}$$

with σ_K a function involving a finite number of past ciphertexts from time $t-\ell$ to $t-\ell'$ ($\ell, \ell' \in \mathbb{Z}$), is a common feature of the SSSC. Equation (7.7) is called the canonical equation.

Remark 1 The benefits of implementing the recursive forms (7.3) or (7.4) instead of directly implementing the canonical form (7.7) is that we can obtain nonlinear functions σ_K of high complexity by implementing simpler nonlinear functions f_K or g_K. The complexity results from the successive iterations which act as composition operations.

At the deciphering side, the automaton takes the form

$$\begin{cases} \widehat{x}_{t+1} = g_K(\widehat{x}_t, c_{t+r}), \\ \widehat{z}_{t+r} = h_K(\widehat{x}_t) \end{cases} \tag{7.8}$$

where $\widehat{x}_t$ is the internal state. Similarly to the cipher part, the automaton having a finite input memory, it means that, by iterating Equation (7.8) a finite number of times, one also obtains

$$\hat{x}_t = \ell_K(c_{t+r-1}, \ldots, c_{t+r-M}),$$

and thus,

$$\hat{z}_{t+r} = h_K(\ell_K(c_{t+r-1}, \ldots, c_{t+r-M})).$$

Hence, it is clear that after a transient time of maximal length equal to M, it holds that, for $t \geq M$,

$$\widehat{x}_t = x_t \text{ and } \widehat{z}_{t+r} = z_{t+r}. \tag{7.9}$$

In other words, the generators synchronize automatically after at most M iterations. Hence, the decryption is automatically and properly achieved after at most M iterations too. No specific synchronizing protocol between the cipher and the decipher is needed. This explains the terminology Self-Synchronizing Stream Ciphers. The integer M is called the synchronization delay.

Hereafter, the considered automata will be assumed to operate on the q elements finite field $\mathbb{F} = \mathbb{F}_q$ where q is a prime power.

7.2.3 Flat LPV Automata and SSSC

For the automaton described by (7.3) (or the equivalent automaton described by (7.4) after substitution) to get a finite input memory feature (see (7.5)), the solutions proposed in the open literature call for state transition functions g_K in the form of shifts or more generally T–functions (T for Triangle). It is recalled that T–functions are functions that propagate dependencies in one direction only. Till now, none of the proposed SSSCs has involved non-triangular state transition functions although T–functions are known to suffer from weakness [152]. Indeed, T–functions induce a propagation of differential properties which make them easier to cryptanalyse.
It is explained by the fact that no systematic methodology for constructing finite automata with finite input memory and involving general non triangular state transition functions was proposed so far. Actually, in [108] a particular class of automata called flat LPV (Linear Parameter-Varying) has been introduced and it has been shown that this class allows to define such an expected systematic methodology. Let us recall what is a flat automaton.

Definition 1 An automaton described by the dynamics f verifying

$$x_{t+1} = f(x_t, m_t) \tag{7.10}$$

where $x_t \in \mathbb{F}^n$ is the state, $m_t \in \mathbb{F}$ is the input, is said to be flat, if there exists a function

$$\begin{aligned} h: \quad & \mathbb{F}^n \times \mathbb{F} \to \mathbb{F} \\ & c_t = h(x_t, m_t) \end{aligned}$$

such that all system variables can be expressed as a function of c_t and a finite number of its backward and forward shifts.

The output c_t is called the flat output. Hence, by definition, there exists a function $\mathcal{F}$ such that

$$x_t \quad = \quad \mathcal{F}\big(c_{t+t_0}, \ldots, c_{t+t_1}\big) \tag{7.11}$$

where t_0 and t_1 are $\mathbb{Z}$-valued integers. A central remark is that (7.11) is nothing but the canonical equation of an SSSC (compare with (7.5)). As a direct consequence, a flat automaton acts as a primitive to being an SSSC and thus is central for design perspectives.
LPV automata, defined over a field $\mathbb{F}$, are described by the following state space representation:

$$x_{t+1} = A_{\rho(t)} x_t + B m_t \tag{7.12}$$

$x_t \in \mathbb{F}^n$ is the state vector, $m_t \in \mathbb{F}$ is the input. The matrices $A \in \mathbb{F}^{n \times n}$ and $B \in \mathbb{F}^{n \times 1}$ are respectively the dynamical matrix and the input matrix. The output c_t is defined as

$$c_t = C x_t \tag{7.13}$$

with $C \in \mathbb{F}^{1 \times n}$ the output matrix. The matrix B is the input matrix and defines the component x_t^i on which the symbol m_t is added. Let us note that m_t can be added to several components. Such a system is called Linear Parameter-Varying because it is written with a linear dependency with respect to the state vector.The set of all varying parameters of A are collected on a vector denoted by

$$\rho(t) = \left[\ \rho^1(t), \rho^2(t), ..., \rho^L(t) \ \right] \in \mathbb{F}^L$$

where L is the total number of non-zero (possibly varying) entries. Such automata can exhibit nonlinear dynamics. Indeed, the nonlinearity is obtained by defining the varying parameters $\rho^i(t)$ as nonlinear functions $\varphi^i : \mathbb{F}^{s+1} \to \mathbb{F}$ of the output c_t (or a finite number of shifts) $\rho^i(t) = \varphi^i(c_t, c_{t-1}, \cdots, c_{t-s})$ with s a natural number. Let us notice that the notation $\rho^i(t)$ (usual in the literature for LPV systems) is somehow abusive because it does not reflect an explicit dependency with respect to the time t but on quantities, here c_t, indexed with t.
Furthermore, the LPV structure is suitable to construct non triangular state transition functions. Indeed, because of the varying entries, the state transition function is not triangular if there does not exist a common and constant triangularization basis over the whole set of matrices $A_{\rho(t)}$ with $\rho(t) \in \mathbb{F}^L$.

Hence, it suffices to select the position of the varying parameters $\rho^i(t)$ in the matrix A accordingly. As a simple illustration, the automaton governed by Equation (7.12) with the setting

$$A_{\rho(t)} = \begin{pmatrix} a & 0 \\ \rho^1(t) & \rho^2(t) \end{pmatrix}, \; B = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

with a a constant element in $\mathbb{F}$, $\rho^1(t) = c_t \cdot c_{t-1}$ and $\rho^2(t) = (c_{t-2})^2$ is an LPV automaton and does not admit a constant triangularization basis.

A flat LPV automaton is an LPV automaton of which state vector x_t verifies (7.11).
Let us recall from [108] the main proposition which allows to define a family of SSSC based on LPV automata. For brevity, we introduce the following notation. For $t_2 \geq t_1$, denote by $\prod_{l=t_2}^{t_1} A_{\rho(l)}$ the product of matrices $A_{\rho(l)}$ from t_2 to t_1. For $t_2 < t_1$, define $\prod_{l=t_2}^{t_1} A_{\rho(l)} = \mathbf{1}_n$ (the identity matrix of dimension n). Finally, let $\mathcal{T}$ be the scalar defined by $\mathcal{T} = C \prod_{l=t+r-1}^{t+1} A_{\rho(l)} B$.

Proposition 1 If the LPV finite state automaton defined by (7.12) is flat, defining the keystream with delay r as

$$z_{t+r} = C \prod_{l=t+r-1}^{t} A_{\rho(l)} x_t \tag{7.14}$$

and the ciphering function as

$$c_{t+r} = z_{t+r} + \mathcal{T} m_t, \tag{7.15}$$

the set of equations (7.12), (7.14) and (7.15) define the ciphering part of an SSSC.

On the other hand, consider the finite state automaton with internal state $\widehat{x}_t$ with dynamics given by

$$\widehat{x}_{t+1} \;=\; P_{\rho(t:t+r)} \widehat{x}_t + B \mathcal{T}^{-1} c_{t+r} \tag{7.16}$$

with

$$P_{\rho(t:t+r)} = A_{\rho(t)} - B \mathcal{T}^{-1} C \prod_{l=t+r-1}^{t} A_{\rho(l)} \tag{7.17}$$

along with the keystream $\widehat{z}_t$ defined as

$$\widehat{z}_{t+r} \;=\; C \prod_{l=t+r-1}^{t} A_{\rho(l)} \widehat{x}_t \tag{7.18}$$

and the deciphering function obeying

$$\widehat{m}_{t+r} = \mathcal{T}^{-1}(c_{t+r} - \widehat{z}_{t+r}). \tag{7.19}$$

Then, the set of equations (7.16-7.19) define the deciphering part of an SSSC.

The proof given in [108] consists in showing that, if the LPV finite state automaton defined by (7.12) is flat, then there exists an integer M such that the synchronization error $x_k - \widehat{x}_{t+r}$ reaches zero after a finite transient time of length M. The integer M is the synchronization delay. Actually, it is shown that flatness is equivalent to the existence of an integer M such that for all $t \geq 0$,

$$P_{\rho(t+M-1:t+M-1+r)} P_{\rho(t+M-2:t+M-2+r)} \cdots P_{\rho(t:t+r)} = 0 \tag{7.20}$$

where the product (7.20) results from the composition of the state transition functions of the deciphering automaton. Let us note that t and r are independent.

This LPV framework for the design of SSSC is new regarding the literature devoted to the design of SSSC. In particular, it really differs from the serial and parallel constructions proposed in the 90s by Maurer [179].

According to Remark 1, implementing the recursive form (7.12) and (7.16) instead of the canonical form (7.5) is more efficient from a computational point of view.

It is recalled that the non-linearity is obtained by defining the values of every varying parameters $\rho^i(t)$ $(i = 1, \ldots, L)$ involved in the matrices of (7.12-7.19) as non-linear functions φ^i of a finite number of past cryptograms $(\rho^i(t) = \varphi^i(c_t, c_{t-1}, \cdots, c_{t-s}))$. Those functions will be implemented in the form of S-boxes denoted S.

$$\begin{array}{rccc} \varphi^i : & \mathbb{F}^{s+1} & \rightarrow & \mathbb{F} \\ & (c_t, c_{t-1}, \cdots, c_{t-s}) & \mapsto & S(c_t, c_{t-1}, \cdots, c_{t-s}, SK_i) \end{array} \tag{7.21}$$

where SK_i is the subkey number i derived from the secret key denoted with K.

The point is that the LPV automaton defined by (7.12) must be flat for any secret key K and any realization of $\rho(t)$. In other words, flatness must be a generic property of (7.12). Designing an LPV automaton (7.12) which is generically flat relies on an admissible realization of a corresponding structured linear system. A structured linear system is a linear system only defined by the sparsity pattern of the state space realization matrices. In other words, for a structured linear system, we distinguish between the entries that are fixed to zero and the other ones that can take any value in $\mathbb{F}$, including the ones which are time-varying. Hence, a structured linear discrete-time system, denoted by Σ_Λ, is a system that admits the form:

$$\Sigma_\Lambda : \quad x_{t+1} \quad = \quad I_A x_t + I_B m_t \tag{7.22}$$

The entries of the matrices of (7.22) are '0' or '1'. In particular, the entries $A(i,j)$ of I_A (*resp.* $B(i)$ of I_B) that are '0' mean that there are no relation (dynamical interaction) between the state x^i_{t+1} at time $t+1$ and the state x^j_t at time t (*resp.* the state x^i_{t+1} at time $t+1$ and the input m_t at time t). The entries that are '1' mean that there is a relation. As a simple example, let us consider an LPV system with the setting

$$A_{\rho(t)} = \begin{pmatrix} a & 0 \\ \rho^1(t) & \rho^2(t) \end{pmatrix} \text{ and } B = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

where a is a constant element in $\mathbb{F}$, $\rho^1(t)$ and $\rho^2(t)$ are varying parameters in $\mathbb{F}$. The dynamical matrix and the input matrix I_A and I_B of the corresponding structured linear system read:

$$I_A = \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}, \; I_B = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

As a consequence, if the structural linear system (7.22) derived from (7.12) is flat, the flatness will hold for any $\rho(t)$ or equivalently any nonlinearity φ^i (any S-box will be admissible). Hence, the challenge is to define a methodology to construct flat linear structural systems. It is the purpose of the graph-based approach provided in [183] which follows the steps recalled in Appendix 7.9. Roughly speaking, given a triplet (n, r, n_a) with n the dimension of the state, r the delay and n_a the number of non-zero entries of the matrix A, a digraph $\mathcal{G}(\Sigma_\Lambda)$ is constructed according to given rules and the matrices I_A and I_B are derived.

The triplet (n, r, n_a), the number of non-linear functions φ^i and their location in the matrix I_A determine a family of flat LPV-based SSSC. Next subsection aims at summarizing the steps needed for the design of such a family. Then, a particular instantiation, leading to the SSSC called Stanislas, is given in next section.

Summary for the construction of SSSC from a flat LPV-based automaton

Choose a triplet (n, r, n_a) with n the dimension of the state, r the delay and n_a the number of non-zero entries of the matrix A.

Step S1: Choose a component x^i_t on which the plaintext symbol m_t is added. It follows that $B = (0 \ldots 1\ 0\ \ldots 0)^t$ (the entries 1 is located at column i).

Step S2: Choose a component x^i_t ($i \in \{1, \ldots, n\}$) as the desired flat output $y_t = x^i_t$. It follows that $C = (0 \cdots 0\ 1\ 0\ \ldots 0)$ (the only entry 1 is located at the i-th column of C). It can be shown that for the special case when $B = (1\ 0\ \ldots)$, i must be equal to r for $y_t = x^i_t$ to be a flat output.

Step S3: Construct the corresponding digraph $\mathcal{G}(\Sigma_\Lambda)$ according to Step 1-5 given in Appendix 7.9 and derive the matrices I_A and I_B of the structured linear system Σ_Λ.

Step S4: Replace some of the non-zero entries of I_A by a nonlinear function $\rho^i(t) = \varphi^i(c_t, c_{t-1}, \cdots, c_{t-s})$ to construct the matrix $A_{\rho(t)}$ of (7.12) and set $B = I_B$. Not all '1' entries of I_A must be assigned to a non-linear function. Some of them can be merely constant. The choice must obey a trade-off between complexity of the architecture and security (a matter discussed in next section). Since the construction ensures structural flatness, any choice will preserve the self-synchronization property.

Step S5: Complete the design by deriving the equations of Proposition 1. In particular, calculate the matrix (7.17) governing the state transition function of the automata (7.16) ensuring the deciphering.

Example: Consider the triplet $(n = 2, r = 2, n_a = 5)$. Choose

$$B = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \ , C = (0\ 1)$$

The particular setting of the matrix C means that the component x_t^2 of the state vector of the LPV system (7.12) is the desired flat output.
For $n_a = 5$, Step 1-5 given in Appendix 7.9 give

$$I_A = \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}, \ I_B = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \tag{7.23}$$

Let us keep constant and equal to 1 the first three entries of A and let the fourth entry be a nonlinear function. It is denoted by $\rho^1(t)$. This finally leads to the following matrix $A_{\rho(t)}$:

$$A_{\rho(t)} = \begin{pmatrix} 1 & 1 \\ 1 & \rho^1(t) \end{pmatrix}$$

Calculate $P_{\rho(t:t+2)} = A_{\rho(t)} - B \cdot C \cdot A_{\rho(t+1)} \cdot A_{\rho(t)}$. One obtains

$$P_{\rho(t:t+2)} = \begin{pmatrix} -\rho^1(t+1) & -\rho^1(t) \cdot \rho^1(t+1) \\ 1 & \rho^1(t) \end{pmatrix}.$$

Next sections are devoted to the specifications, design rationale and security analysis of a complete SSSC based on flat LPV dynamical systems as described above. The cipher is called Stanislas for Secure Transmission Algorithm with Nontriangular Iterative Structure Looking After Self-synchronization.

7.3 Specification of the flat LPV-based SSSC Stanislas

Stanislas operates over the 16 elements of the finite field $\mathbb{F}_{16}$ defined as: $\mathbb{F}_{16} = \mathbb{F}_2[X]/(X^4 + X + 1)$, the addition being the componentwise exclusive or, simply denoted $+$, and the multiplication, denoted by $\cdot$, being the polynomial multiplication modulo the primitive polynomial $X^4 + X + 1$.

7.3.1 Equations of Stanislas

The internal state $x_t \in \mathbb{F}_{16}^{40}$ of the cipher consists in a vector of dimension $n = 40$ with 4-bit components considered as elements in $\mathbb{F}_{16}$. The input m_t and the output c_t of the ciphering function are 4-bit respective elements in $\mathbb{F}_{16}$.

Ciphering equations

The ciphering equation defines the next internal state $x_{t+1} \in \mathbb{F}_{16}^{40}$ and the cipher output $c_t \in \mathbb{F}_{16}$. Note also that all elements that compose the matrix $A_{\rho(t)}$ and the vectors B and C are elements of $\mathbb{F}_{16}$. The ciphering equation obeys, for $t \geq 0$,

$$\text{cipher:} \quad \left\{ \begin{array}{rcl} x_{t+1} & = & A_{\rho(t)} x_t + B m_t \\ c_{t+1} & = & C x_{t+1} \end{array} \right. \tag{7.24}$$

where B is the column vector equal to $B = (1_{\mathbb{F}_{16}}, 0_{\mathbb{F}_{16}}, \ldots, 0_{\mathbb{F}_{16}})^T$ and C is the row vector equal to $C = (0_{\mathbb{F}_{16}}, 0_{\mathbb{F}_{16}}, 1_{\mathbb{F}_{16}}, 0_{\mathbb{F}_{16}}, \ldots, 0_{\mathbb{F}_{16}})$ with the $1_{\mathbb{F}_{16}}$ component located at column $r = 3$. In other words, the only non-zero component of C which equals $1_{\mathbb{F}_{16}}$ is the third component. Hence, the ciphertext symbol consists in the third component of the internal state. Let us note that in general, the ciphertext obeys Equation (7.13). Hence, the ciphertext results from a linear combination of the state vector components. Here, we propose a construction for which the linear combination reduces to the selection of one component. To guarantee a self-synchronization property (otherwise stated, to ensure Equation (7.20), the component that is selected must coincide with r, the delay of the system. This is why here, $r = 3$.

The matrix $A_{\rho(t)}$ is a 40×40 dimensional matrix. The entries a_{ij} of $A_{\rho(t)}$ are either zero of $\mathbb{F}_{16}$, or 1 of $\mathbb{F}_{16}$ or a nonlinear function of c_t. Among them, $n_a = 115$ entries are non-zero coefficients and

$L = 80$ entries correspond to nonlinear functions φ^i $(i = 1, \ldots, 80)$. Each function φ^i depends on the current ciphertext symbol c_t $(s = 0)$ and on a subkey SK_i of K. This subkey SK_i thus defines the corresponding function φ^i which depends on only one ciphertext symbol c_t.

$$\begin{array}{rccc} \varphi^i: & \mathbb{F}_{16} & \to & \mathbb{F}_{16} \\ & c_t & \mapsto & S(c_t \oplus SK_i) \end{array} \tag{7.25}$$

where the subkey SK_i is a 4-bit word derived from the secret key K as described in the key schedule of subsection 7.3.1 detailed further on. The function S is the bijective S-Box borrowed from Piccolo [217]. It is defined on 4-bit words by Table 7.1. The entries a_{ij} according to their location (row i, column j for $i, j \in \{1, \ldots, 40\}$) are given in Appendix 7.8 in a symbolic manner. In the symbolic representation of $A_{\rho(t)}$, denoted A_S, the functions φ^i $(i = 1, \ldots, 80)$ are assigned to the entries S. Thus, the first line of the Fig. 7.1 corresponds to the first row of the matrix A_S applied to the internal state x_t to produce the corresponding coefficient of the internal state x_{t+1} (combined with m_t to produce c_{t+1}).

Table 7.1: S-box in hexadecimal notation.

x	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
$S(x)$	E	4	B	2	3	8	0	9	1	A	7	F	6	C	5	D

In other words, taking into account the particular structure of the matrices B and C with only one non-zero coefficient, the ciphering process is governed by:

$$\begin{array}{rcl} x^1_{t+1} & = & \sum_{j=1}^{n} a_{1j} x^j_t + m_t \\ x^i_{t+1} & = & \sum_{j=1}^{n} a_{ij} x^j_t, \ i = 2, \cdots, n \\ c_{t+1} & = & C x_{t+1} = x^3_{t+1} \end{array} \tag{7.26}$$

Deciphering equations

The deciphering is governed by two equations defined, for $t \geq 0$, by

$$\text{decipher:} \quad \left\{ \begin{array}{rcl} \widehat{x}_{t+1} & = & P_{\rho(t:t+r)} \widehat{x}_t + B\mathcal{T}^{-1} c_{t+r} \\ \widehat{m}_{t+r} & = & \mathcal{T}^{-1}(c_{t+r} - C \prod_{l=t+r-1}^{t} A_{\rho(l)} \widehat{x}_t) \end{array} \right. \tag{7.27}$$

where the 40×40-dimensional matrix $P_{\rho(t:t+r)}$ over $\mathbb{F}_{16}$ verifies $P_{\rho(t:t+r)} = A_{\rho(t)} - B\mathcal{T}^{-1} C \prod_{l=t+r-1}^{t} A_{\rho(l)}$ and $\mathcal{T} = C \prod_{l=t+r-1}^{t+1} A_{\rho(l)} B$.

Otherwise stated, the deciphering consists of two equations. The first one achieves the computation of the next internal state $\hat{x}_{t+1}$ from the current state $\hat{x}_t$ and the delayed ciphertext c_{t+r}. The second equation ensures the recovery of the plaintext symbol m_t. The self-synchronization between the internal states x_t and $\widehat{x}_t$ of the cipher and the decipher automata and thus a proper decryption are guaranteed, by construction, after a finite transient time.

Let $X[i]$ denotes the i-th row of a matrix X. Taking into account the particular structure of the matrices A, B and C, the delay $r = 3$, and noticing that $\mathcal{T}$ equals one, the deciphering process is governed by:

$$\begin{array}{rcl} \widehat{x}^1_{t+1} & = & \left(P_{\rho(t:t+3)}[1] \cdot \widehat{x}_t\right) + c_{t+3} \\ \widehat{x}^i_{t+1} & = & \left(P_{\rho(t:t+3)}[i] \cdot \widehat{x}_t\right) \ i = 2, \cdots, 40 \\ \widehat{m}_{t+3} & = & \left(A_{\rho(t+2)} A_{\rho(t+1)} A_{\rho(t)}\right)[3] \cdot \widehat{x}_t + c_{t+3} \end{array} \tag{7.28}$$

where

$$\begin{array}{rcl} P_{\rho(t:t+3)}[1] & = & A_{\rho(t)}[1] - \left(A_{\rho(t+2)} A_{\rho(t+1)} A_{\rho(t)}\right)[3] \\ P_{\rho(t:t+3)}[i] & = & A_{\rho(t)}[i], \qquad 2 \leq i \leq n \end{array} \tag{7.29}$$

Key schedule

The matrix A consists of $n_a = 115$ non-zero entries, and among them, $L = 80$ are functions φ^i depending on the subkey SK_i. Thus, the key schedule process aims at generating 80 subkeys of 4-bit length: $SK_1, \cdots, SK_{80}$ from the 80-bit master key K arranged as 20 words $K_1, \ldots, K_{20}$ of 4-bit length.

To do so, the ciphering equations (7.26) with $m_t = 0$ are used. During this process, the parameters SK_i involved in the functions φ^i (see Equation (7.25)) are set to zero. The internal state x_0 is initialized by duplicating the master key K as $(x_0^1 \cdots x_0^{20}) = (K_1 \cdots K_{20})$ and $(x_0^{21} \cdots x_0^{40}) = (K_1 \cdots K_{20})$. Then, the initial internal state x_0 is updated ten times by using equations (7.26) with $m_t = 0$ for $t = 0, \ldots, 9$.

After those ten iterations, the ten subkeys $SK_1, \ldots, SK_{10}$ are respectively initialized with the following components of the internal state x_{10}: x_{10}^1, x_{10}^2, x_{10}^7, x_{10}^{10}, x_{10}^{17}, x_{10}^{18}, x_{10}^{22}, x_{10}^{26}, x_{10}^{28}, x_{10}^{34}.
This process is repeated 7 more times to initialize the other subkeys $SK_{10i+1}, \ldots, SK_{10i+10}$, for $i = 0, \ldots, 7$.

7.3.2 Ciphering Process

The plaintext consists of ℓ elements of $\mathbb{F}_{16}$: $m = m_0 \cdots m_{\ell-1}$. The initial state x_0 is first initialized with 40 random elements of $\mathbb{F}_{16}$. These initial values are kept secret and are not transmitted to the decipher. The way those elements are randomly picked is out of the scope of this chapter. Only consider that you have a source of randomness. x_0 could be considered as a secret nonce.

Then randomly pick $n - 1 = 39$ elements $m_{-39}, \cdots, m_{-1}$ of $\mathbb{F}_{16}$ as the synchronization sequence which is placed before the plaintext. It is recalled that at most $n = 40$ iterations are needed for the self-synchronization to be achieved.

Then, because of the parameter value $r = 3$ that induces a delay, randomly pick $r - 1 = 3 - 1 = 2$ more elements $m_\ell, m_{\ell+1}$ of $\mathbb{F}_{16}$ that will be placed at the end of the plaintext sequence m to process the two last plaintext symbols. Finally, the sequence that must feed the cipher is $m^\star = m_{-39}, \ldots, m_0, \ldots, m_\ell, m_{\ell+1}$.

The resulting ciphertext consists of the sequence $c_{-39}, \ldots, c_{\ell+1}$ of $(\ell+41)$ symbols in $\mathbb{F}_{16}$, computed with the ciphering Equations (7.26) using Matrix A_S given in Appendix 7.8 for t in $-39, \cdots, \ell + 1$:

$$\begin{array}{rcl} x_{t+1}^1 & = & S(x_t^{20} \oplus SK_0) \oplus S(x_t^{26} \oplus SK_1) \\ & & \oplus S(x_t^{29} \oplus SK_2) \oplus S(x_t^{40} \oplus SK_3) \oplus m_t \\ x_{t+1}^2 & = & x_t^1 \oplus S(x_t^2 \oplus SK_4) \oplus S(x_t^{31} \oplus SK_5) \\ & & \oplus S(x_t^{35} \oplus SK_6) \\ x_{t+1}^3 & = & x_t^2 \oplus S(x_t^3 \oplus SK_7) \\ x_{t+1}^4 & = & S(x_t^2 \oplus SK_8) \oplus S(x_t^3 \oplus SK_9) \\ & \vdots & \\ c_{t+1} & = & x_{t+1}^3 \end{array}$$

Those equations are the ones given in Fig. 7.1 corresponding with the coefficients of the matrix A_S.

Note that the Matrix A_S given in Appendix 7.8 could be seen as one round of a block cipher applied on a state with 40 4-bit words where after each round the 4-bit word x^3 is outputted whereas the 4-bit word x^0 is completely updated by other 4-bit words. At each clock, each 4-bit word crosses at least one S-box, except the 4-bit word x^1 that could be considered as a temporary variable (as it receives the 4-bit message m_t). Note also that the updated rule for most of the variables x^i $(5 \leq i \leq 40)$ could be written as $x_{t+1}^i = S(x_t^{i-1} \oplus SK_k) \oplus f(x_t^{i-2})$ or $S(x_t^{i-1} \oplus SK_k) \oplus f(x_t^{i-2}) \oplus S(x_t^j \oplus SK_{k'})$ for given j, k and k' and where f is the identity or the S-box S. It could be seen as a generalization of the so-called L-scheme with a circular permutation used in the block cipher MISTY1 with balanced inputs/outputs.

The complete ciphering process of Stanislas is illustrated on Fig. 7.1

7.3.3 Deciphering Process

The decipher receives the cryptogram consisting of $\ell + 41$ symbols $c_{-39}, \ldots, c_{\ell+1}$ in $\mathbb{F}_{16}$. The internal state $\widehat{x}_0$ is initialized to an arbitrary value, for example the zero value.

Then, the deciphering Equations (7.28)-(7.29) are applied to recover a $\ell + 41$-length message $\widehat{m} = \widehat{m}_{-41}, \ldots \widehat{m}_{\ell-1}$. The plaintext sequence is recovered as the last ℓ symbols $m = \widehat{m}_0 \ldots \widehat{m}_{\ell-1}$.

Remark 2 It could be surprising that a part of the ciphering process directly depends on a secret nonce (i.e. x_0). Instead, we could imagine that an 80-bit IV is used to generate the first value x_0 adding an IV schedule to the key schedule process. First, generate the subkeys using the key schedule, then initialize the internal state with the concatenation of the IV and of the key. Then, apply again the key schedule process (but including the generated subkeys SK_i in the S-boxes) to initialize the internal state x_0. But, note that in this case, the particular property that the internal states (that must be kept secret) of the ciphering part and of the deciphering part are not required to be equal is lost. Indeed, in this case where an IV is used, we will suppose that the internal state will be computed in the same way in both sides.

The matrix A consists of $n_a = 115$ non-zero entries, and among them, $L = 80$ are functions φ^i depending on the subkey SK_i. Thus, the key schedule process aims at generating 80 subkeys of 4-bit length: $SK_1, \cdots, SK_{80}$ from the 80-bit master key K arranged as 20 words $K_1, \ldots, K_{20}$ of 4-bit length. To do so, the ciphering equations (7.26) with $m_t = 0$ are used. During this process, the parameters SK_i

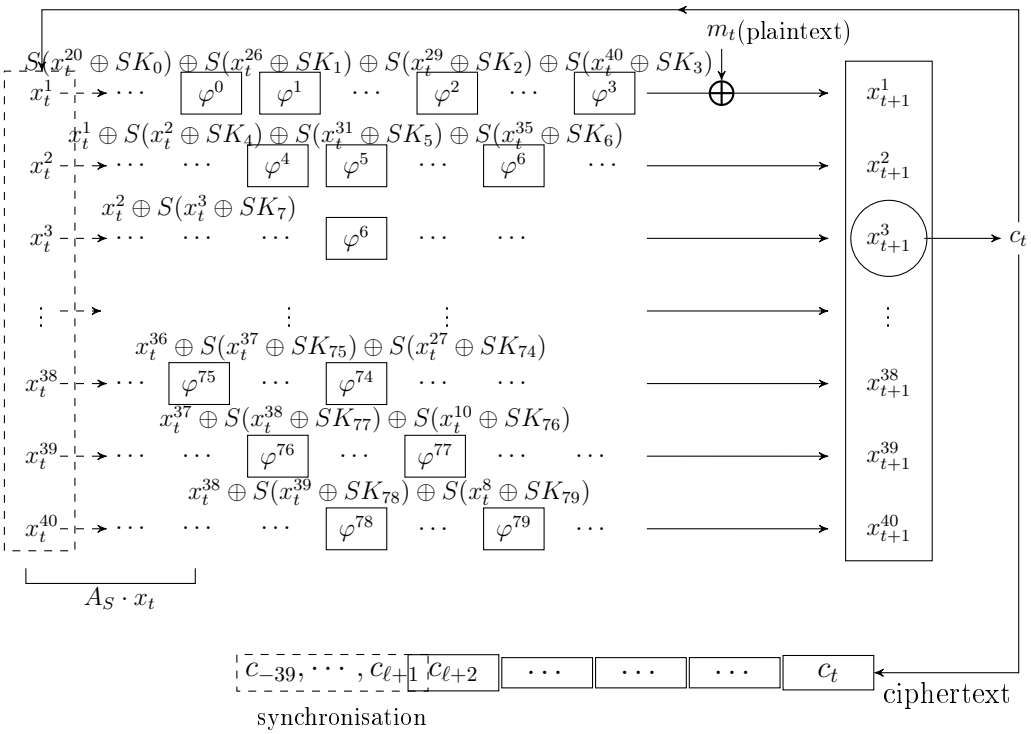


Figure 7.1: The complete ciphering process of Stanislas with $\varphi^i(x_t^j) = S(x_t^j \oplus SK_i)$. x_t^i and x_{t+1}^i are the coefficients of the internal state at time t and time $t+1$. m_t represents a plaintext symbol (4 bits), c_j the ciphertext symbols (4 bits) where $c_{-39}, \ldots, c_{l+2}$ are required for synchronization. The equations of the lines are derived from the coefficients of the rows of the matrix A_S.

involved in the functions φ^i (see Equation (7.25)) are set to zero. The internal state x_0 is initialized by duplicating the master key K as $(x_0^1 \cdots x_0^{20}) = (K_1 \cdots K_{20})$ and $(x_0^{21} \cdots x_0^{40}) = (K_1 \cdots K_{20})$. Then, the initial internal state x_0 is updated ten times by using equations (7.26) with $m_t = 0$ for $t = 0, \ldots, 9$.

7.4 Design Rationale and Security Analysis

In this section, we motivate the choices of the field on which the cryptosystem operates, the dimension n of the internal state, the delay r and the structure of the matrix A. Most of the choices rest on security criteria, other ones take into account practical considerations, regarding in particular the hardware implementation issues.

7.4.1 Design Rationale

Field on which the cryptosystem operates: Galois field $GF(16)$ Any quantities m_t, c_t, components of x_t and $\hat{x}_t$ and non-linear functions φ^i (S-boxes) inputs are 4-bit data. It is motivated by the fact that the cryptosystem is intended to be implemented on a digital equipment. Hence, field extensions and so, power of two are required. On the other hand, 8-bit would be too heavy for an embedded algorithm. In particular, S-boxes would involve too many logic gates.

Dimension n: 40 As the internal state components are 4-bit words, a dimension $n = 40$ provides an internal state of 160 bits and thus a security level of 80 bits. Indeed, to prevent time-memory trade-off attack [139] (an attack which is a trade-off between exhaustive search and table look-up), the internal state must be two times longer than the key length which defines the security level. This level is compatible with a real-world application.

Delay r: 3 The more the delay, the more the algebraic degree of the entries of $P_{\rho(t:t+r)}$, recalling that $P_{\rho(t:t+r)}$ involves the product of r matrices (see (7.29)). Thus, for a good resistance against an

algebraic attack ([90]), the algebraic degree should be as large as possible. On the other hand, the more the delay, the more the complexity of implementation and the less the computational performances. The delay $r = 3$ results from a trade-off between security with respect to algebraic attacks (to increase the overall algebraic degree) and ease of implementation (especially the implementation of the P deciphering matrix which involves several S-box multiplications (see Equation (7.17) in the general case and (7.29) for Stanislas)).

Structure of the matrices A and P We recall that the matrix A (and thus P from the computation (7.29)) is derived from the construction of a digraph (see Appendix 7.9) from which an adjacency matrix I_A is extracted. More precisely, the adjacency matrix I_A determines the entries of $A_{\rho(t)}$ that are zero and the others that are possibly non-zero. The number n_a of edges in the digraph $\mathcal{G}(\Sigma_\Lambda)$ corresponds to the number of non-zero entries of I_A. Hence, the number n_a also determines the number of non-zero entries of the state transition matrix P. Beyond the number of non-zero entries, their location (row and column) must also be chosen. Finally, it must be decided whether a non-zero entry will be 1 or will correspond to an S-box. All those issues have been addressed by considering several criteria regarding the security, in particular the good resistance to classical attacks and the good diffusion delay, while satisfying a trade-off with respect to the computational complexity for the sake of implementation. Let us introduce symbolic representations of $A_{\rho(t)}$ and $P_{\rho(t)}$ denoted by A_S and P_S where the coefficients of A_S and P_S belong to $\mathbb{Z}[S]$, S representing any non-linear function. The following considerations on A_S and P_S can be made.

- Diffusion Delay and Depth. The diffusion delay and the depth are properties related to the consideration of the powers A_S^p and of P_S^p as p, a natural integer, increases. Indeed, the power of matrices results from the successive iterations of the ciphering and the deciphering process. Let p denotes the power of a matrix $Z \in M_n(GF(16))$. The diffusion delay, introduced in [29], is the smallest value, denoted by d_0, of p such that Z^p does not have any zero coefficient. In other words, it is the smallest value of p such as each element of the initial internal state x_0 has influenced every element of x_t for $t \geq d_0$. The depth, introduced in [52], is the smallest value, denoted by d_1, of p such that any entry of Z^p are polynomials of degree at least 1. We are looking for the smallest values of d_0 and d_1.

- Algebraic Degree. Considering the matrix resulting from successive powers of A_S and P_S, we are first interested in ensuring that at least one entry has the largest algebraic degree. To this end, we must add a cycle on the r-th vertex of the digraph $\mathcal{G}(\Sigma_\Lambda)$, which equivalently means that the entry of A_S and P_S located at row r and column r must be an S-box. Furthermore, the evolution of this quantity, after successive iterations, must meet an ideal shape. It must increase by one at each iteration, must remain constant and equal to its maximum value as long as possible and finally must drop down to zero (let us recall that after 40 iterations, due to (7.20), the product reaches exactly zero).

- Full Rank Matrix. The fact that A_S is a full-rank matrix is a necessary condition to ensure a full diffusion of the internal state and to maximize the dependency between the involved terms at time t and the involved terms at time $t + 1$. Moreover, this condition guarantees that the encryption process does not collapse. By construction (see Appendix 7.9) to ensure flatness, every element of the subdiagonal of I_A from the r-th one is 1. Hence, for A_S to be full-rank, each column and each row must contain at least one non-zero element. And yet, by construction, only the $r - 1$ first elements of the last column can be non-zero. Hence, one of them must be non-zero. From the digraph point of view, it means that at least one of the $r - 1$ first vertices must be connected to the last vertex.

The symbolic matrix A_S which has been finally selected is given in Appendix 7.8. It has been obtained after 700000 random runs performed under the aforementioned constraints: best diffusion delay and depth, algebraic degree (especially increase by 1 at each iteration), full rank matrix. Several matrices correspond to the best choices (we add a sum indicator without weighing) and we finally choose the one with the best implementations for A_S and for P_S. The symbolic matrix P_S can be directly obtained by considering Equations (7.29). The matrix A_S involves $n_a = 120$ non-zero entries and its number of S-boxes is $L = 80$. The matrix extracted from A_S and P_S by removing the first r rows and columns have a special feature. The lower subdiagonal is full of coefficients S (corresponding to S-boxes and

denoted SB in the symbolic representation) and the subsubdiagonal just above is full of 1. Thus, each line has at least one S-box to ensure that the internal state x_t is updated in a non-linear way: this corresponds to a non-linear shift register.

The corresponding diffusion delay, $d_0 = 7$, could be considered as a good diffusion delay with regard to the synchronization delay that equals 40 (the system dimension). Indeed, the worst diffusion delay is equal to 40 and the best diffusion delay of 1 could only be reach with a matrix full of non-zero coefficients. Thus, we consider that a diffusion delay of 7 is a sufficiently good compromise between the best diffusion delay and a reasonable number of non-zero coefficients.

Let us notice that the dimension n of the system is the upper bound of the diffusion delay. Indeed, due to (7.20), the product reaches exactly zero in $GF(16)$ since the synchronization is achieved after at most 40 iterations. The depth is $d_1 = 7$ and is the best that we manage to achieve.

S-box Various classifications of 4-bit S-boxes exist in the literature [75, 169, 204]. For the sake of hardware optimization, the same S-box has been used to define the $L = 80$ nonlinear functions φ^i. Two kinds of criteria have been considered: theoretical and practical. On one hand, we have selected S-boxes to satisfy the maximum differential probability and the maximum (absolute) linear bias 2^{-2}, the algebraic degree 3, and no fixed-point. Four S-boxes that satisfy those criteria and that have simple algebraic expressions (i.e. a minimal number of non-linear transformations) have been selected, each one corresponding respectively to the four classes proposed in [204].

From an implementation point of view, it turns out that the S-box depicted in Table 7.1 induces the smallest gate count. It is the Piccolo S-box ([217]). The area is around 23 GEs[1]). It involves four NOR gates, three XOR gates and one XNOR gate. Let us notice that the masking method can be applied using only three shares, making this S-box suitable for efficient threshold implementations.

Key Schedule The key schedule has been chosen to reuse existing circuit while sufficiently mixing together the key words. To do so, we use the already implemented matrix A by applying it a sufficient number of times when looking at the diffusion degree and at the induced algebraic degree. The extracted 4-bit words of the internal state x_t - at positions 1, 2, 7, 10, 17, 18, 22, 26, 28, 34 - have been chosen to be among the ones that depend of the maximum number of other elements of x_t to ensure to maximize the diffusion effect of the initial state x_0.

From a theoretical point of view, if we consider that each S-box behaves as a random function (i.e. it has a behavior sufficiently near the one of a true random function) and using the direct extension of Lemma 9 and Theorem 7 of [180], we could say, that after $d_0 + 2$ of the matrix A on the input x_0, the applied transformation behaves as a random function. In other words, the operations done to fulfill the subkey words behaves as a random function.

7.4.2 Security Analysis

The following section focuses on the security of Stanislas against known attacks. We claim a 80-bit security level which corresponds to the key length (we could not have a security level greater than the key length). Moreover, this security level is also achieved regarding the length of the main register: 160 bits. Indeed, the Guess-and-Determine attacks described in [139] imply to double the length of the used register compare to the key length to achieve a security level corresponding to the key length. Thus, we try to derive security bounds for all known attacks against Stanislas and we found no attacks that work with a complexity smaller than 2^{80} operations which corresponds with our security claim.

Moreover, it has been proven in [109] that the canonical form of an SSSC is secure against Chosen Plaintext Attacks (IND-CPA secure) but not against Chosen Ciphertext Attacks (IND-CCA security). We do not claim any security result in this last model. Moreover, we suppose that the attacker has no access to the key and to the initial value of the internal state x_0. To prevent collision search attack, we limit the size of each plaintext to 2^{64} 4-bit words.

First, it seems very difficult to analyze the security of Stanislas in its true settings (i.e. the 160-bit internal state x_0 is a secret nonce). In those settings, we could only say that:

[1] To see the meaning of GE metric, please refer to Section 7.5.

- the time-memory-data trade-off attacks described in [33, 64, 139] apply when the internal state is smaller than two times the key length. This is why we choose the length of the internal state to be twice the key length to prevent this kind of attacks.

- Guess-and-Determine Attacks [139] consist in guessing a part of the state to further determine the remaining part of the state. Thus, at time t, suppose that we know x_t^2, x_t^3 and of course, c_t. Thus, we could suppose that we observe n consecutive outputs from c_t to c_{t+n}. Thus, how much does it cost to recover the induced subkey SK_i? First, from the equation $x_{t+1}^3 = x_t^2 \oplus S(x_t^3 \oplus SK_7)$, we could directly compute SK_7. Thus, we could derive the successive value of x_t^2 from t to n. Thus, from x_t^2 we derive non-linear equations where the unknowns are x_t^1, x_t^{31}, x_t^{35}, SK_5, SK_4 and SK_6 and the known terms are x_t^2 to x_{t+n}^2. Thus, as the Algebraic Normal Form (ANF) of the S-box has 2 equations with 4 terms, 1 equation with 9 terms and 1 equation with 7 terms, each new x_t^2 will induce a system of $4 \times 3 \times (n+1) + 3 \times 4 = 12n + 24$ unknown binary variables with 4 equations with in total 24 non-linear terms. Thus, we could not solve the system without guessing a part of it whatever the n value. Even considering that we guess a part of the system, the combinatorial explosion seems clear because each value of the internal state depends on at least one other value. Thus, we conjecture here that Stanislas is safe against this type of attacks since, even if a part of the internal state is guessed, each 4-bit word of this state is updated through a XOR with a 4-bit word of subkey and an S-box. Moreover, since the diffusion delay of A_S is $d_0 = 7$, this leads to guess after 7 outputs all the key. Thus, guessing a part of the state allows to guess a part of the key which, in then, amounts to a guess-and-determine attack with a complexity greater than the key exhaustive search.

Thus, instead, when looking at classical attacks, we will use the model described in Remark 2 where the initial state is derived in its first components after 20 iterations of the matrix A_S applied on the concatenation of an IV and of the master key K and in its last components after 14 more iterations. Those settings could be considered as a degrading mode of the original Stanislas specifications. Thus considering that the attacker has full access to the IV, we obtain the following bounds against classical attacks:

- Differential / Linear Cryptanalysis: we compute the lower bounds on the minimal number of active S-boxes for the computation of the internal state with remark 2. To do so, we implement the model of remark 2 using Constraint Programming to explore all the possible paths in the induced graph. Then, we obtain that, for the differential case, after 7 iterations, a minimal number of 38 S-boxes has been crossed, after 10 iterations, 46, after 14 iterations, 65. As the differential probability of the chosen S-box is equal to 2^{-2}, we could guarantee that the 80-bit key exhaustive search is less expensive than passing through more than 40 S-boxes, which is the case after 10 iterations of the A_S matrix. In the same way, for the linear case, we obtain after 7 iterations, 35 active S-boxes, after 10 iterations, 41 and after 14 iterations, 59. Thus, for the same reason, after 10 iterations, a 80-bit key exhaustive search is more efficient.

- Algebraic Attacks: This kind of attacks [90] is possible when the overall degree of the induced system of equations does not sufficiently increase at each clock. Especially, if the overall degree d in each of the n unknown variables (the key variables for example) of the system is such that $(n^d)^{2.5}$ is lower than the security bounds, it means that it is faster to solve the induced system by Gaussian elimination (considering that each new monomial is a new unknown variable) than trying all the keys of the system. Thus, we want to prevent this attack from happening as described below. The algebraic degree of each S-box component is the best one: equal to 3. Thus, each passing through an S-box increases the degree in the equations describing the internal state and in the equations describing the key. Even if the SK_i linearly depends on the master key bits $K_1, \cdots, K_{80}$ (there are 80 key bits that are unknown), if we write the number of variables after crossing the first S-box, we have $(80)^3$ monomials depending on the unknown key bits, after the second pass we obtain $(80)^6$ monomials also depending on the unkonwn key bits and so on. Thus, if we apply those estimations using the bounds on the number of active S-boxes of the differential/linear case, after 10 iterations, we have 46 active S-boxes, which means that the number of unknowns (considering that each new monomial is a new unknown) is lower bounded after 10 iterations by $(80)^{3\times 46} \approx 2^{872,16}$ where 80 are the unknowns coming from the master key, 3 is the algebraic degree of the S-box and 46 is the number of crossed S-boxes. Thus,

we conjecture that the complexity of the best algebraic attack is greater than the 80-bit key exhaustive search.

- Cube Attacks: As established in [104], a cipher is vulnerable to cube attacks if an output bit can be represented as a sufficiently low degree polynomial over $GF(2)$ of key and input bits. It works by summing an output bit value for all possible values of a subset of public input bits, chosen such that the resulting sum is a linear combination of secret bits. Repeated application of this technique gives a set of linear relations between secret bits that can be solved to discover these bits. In [209], the authors analyzed this kind of attacks on the block cipher Piccolo, especially its S-box. They stated that after 8 rounds, no relation with 63 input bits could be found. The minimal number of S-boxes crossed for 8 Piccolo rounds is 58 whereas in our case, it is 46 after 10 iterations. So we conjecture, that we cross a sufficient number of S-boxes after few iterations to prevent having low degree relations between secret key bits and public input bits.

In summary, we conjecture that most of the usual attacks which apply in the stream cipher context have a complexity greater than the exhaustive key search for Stanislas.

7.5 Hardware Performance and Implementation Aspects

We give hereafter the implementation results of a straightforward implementation of Stanislas. It produces one 4-bit word of ciphertext per clock cycle. Subkeys are computed in the initialization step and stored in dedicated registers, before the cipher state processing. The same material is used for the cipher state and the Key Schedule processes. The hardware implementation of Stanislas is not an optimized version targeting any specific performance. The main area occupation comes from the matrix update as it carries big registers during all the calculations. Those registers are the internal state which are mixed with the subkeys either with binary addition or multiplication. This means updating a 40×40 bits register all along the matrix update.

During ciphering process, the matrix update is solely made of S-boxes and XOR of 4-bit words. The lowest line of the matrix in terms of area occupation is made of 1 S-box and 2 additional XORs, and the biggest of 4 S-boxes and 8 additional XORs. The deciphering process implies adding 23 multiplications, 28 S-boxes and 39 XORs to the previous total which makes deciphering heavier in terms of area occupation.

The matrix is implemented line by line, calculated straightforwardly following the equations as depicted in Fig. 7.1. The S-boxes are implemented in a Look-Up-Table (LUT) way, so we let the compiler do its own optimizations. Our Stanislas implementation combines both the encryption and decryption process in order to ease comparison with others (synchronous or self-synchronous) stream ciphers.

We implemented Stanislas in VHDL and we provide in Table 7.2 FPGA hardware implementations and performance comparisons with synchronous SCs Trivium [101] and Grain [138], final members of the eSTREAM portfolio, another SSSC Moustique and the AES-based SSSC CFB1-AES128 as defined in NIST SP 800-38a [111]. The chosen FPGA platform for our benchmark is the Xilinx Spartan-6 XC6SLX75T, package FGG676. The ISE compiler was set on for high effort on area reduction. Post-place-and-route results are provided.

To get Trivium and Grain implementation results, we have reused VHDL reference implementations coming from their submission packages, without further optimization. We have implemented our own straightforward VHDL versions of Moustique and CFB1-AES128. For this latter, we have implemented the S-Boxes as LUTs, to be consistent with the S-Boxes LUT implementations of Stanislas.

	Area (slices)	**Init. (cycles)**	**Synchro. (cycles)**	**Freq. (MHz)**	**TP (Mbps)**
Trivium	47	1603	0	191	191
Grain	48	256	0	355	355
Moustique	166	105	105	309	309
CFB1-AES128	745	0	128	73	849
Stanislas	701	254	40	95	380

Table 7.2: Xilinx Spartan-6 XC6SLX75T (FPGA) Straightforward Implementation Results.

At first sight, we can check that some well-known properties are visible in the results. For example, Trivium and Grain are very compact, which can be explained by their low gate counts, flip-flops excluded. Moreover, the number of initialization cycles needed for Trivium, Grain and Moustique which is consistent with the specifications: e.g., Trivium needs a warm-up phase of minimum 1152 steps. This number is really slow for Stanislas where it just consists in a Key Schedule and it is equal to 0 for CFB1-AES128 where the key is processed on the fly.

Surprisingly, straightforward implementation of Stanislas provides the best throughput (TP) compared to the other stream ciphers, even self-synchronizing, to the exception of CFB1-AES128. The reason is that one 4-bit word is processed by clock cycle, so the throughput is given by: 95×4 (bits) $= 380$ Mbps. That justifies the design choice of processing 4-bit words instead of individual bits. CFB1-AES128, produces 128-bit words after each iteration of the AES in CFB mode(10 rounds plus one clock cycle for the final Xor operation), hence a throughput of $(128 * 73)/11 = 849$Mps. Compared to the standard approach CFB1-AES128, the time needed for synchronization is also shorter, along with a smaller area.

This encouraging result has to be mitigated if we consider the combined metric TP/area as shown on Table 7.3. This latter allows to estimate the cost of optimized parallel implementations (e.g., unfolded), where many bits can be processed in parallel, at the expense of additional area.

	TP/Area
Trivium	4.06
Grain	7.39
Moustique	1.86
CFB1-AES128	1.13
Stanislas	0.54

Table 7.3: Combined Metric TP/Area (Mbps/Slice on Xilinx Spartan-6 XC6SLX75T FPGA).

As we can see, straightforward implementation of Stanislas will suffer from the comparisons of its competitors' optimized versions. We can then estimate, in Table 7.4, the theoretical implementation results of all Stanislas competitors when all of them are unfolded versions which process 4 bits per clock cycle, as Stanislas.

	Area (slices)	TP (Mbps)
Trivium	188	764
Grain	192	1420
Moustique	664	1236
CFB4-AES128	2980	3396
Stanislas	701	380

Table 7.4: Theoretical Implementation Results of some (SS)SCs on Xilinx Spartan-6 XC6SLX75T FPGA for 4-bit Versions

As we can see, CFB4 mode has four-fold throughput speedup, beating Stanislas proposal with a significant margin, and it requires only 32 steps for synchronization. But it implies to occupy a big amount of FPGA slices, which is not always affordable for some constrained applications.

Future works will include the study of the cost of side-channel protected Stanislas implementations.

7.6 Conclusion

An instantiation called Stanislas, of a dedicated Self-Synchronyzing Stream Cipher (SSSC) has been proposed. Its main peculiarity comes from the fact that it involves an automaton with finite input memory using non-triangular state transition functions. The construction is based on a general and systematic methodology that uses automata (called Linear Parameter Varying, LPV) admitting a matrix representation and a special property called flatness. The security analysis allows to conjecture that most of the usual attacks which apply in the stream cipher context have a complexity greater than the key exhaustive search for Stanislas. But Stanislas could not be considered having a small hardware footprint.

However, when straightforward implementations are considerd, Stanislas provides bigger throughput than the considered stream ciphers, and its intrinsic synchronization delay is much smaller than the SSSC Moustique (40 clock cycles instead of 105) and the standard approach CFB1-AES128 (40 clock cycles instead of 128).

Moreover, the number of surviving Self Synchronizing Stream Ciphers after a phase of public cryptanalysis time is equal to zero. So, we hope that Stanislas will be the first one and we encourage the symmetric key cryptographic community to cryptanalyze it.

7.7 Appendices

7.8 The Matrix A_S

The matrix A_S is given in Fig. 7.2.

Figure 7.2: The Matrix A_S.

7.9 Construction of the Matrices of the SSSC

A digraph $\mathcal{G}(\Sigma_\Lambda)$ describing the structured linear system associated to the state equations (7.12), is the combination of a vertex set $\mathcal{V}$ and an edge set $\mathcal{E}$. The vertices represent the states and the input components of Σ_Λ while the edges describe the dynamic relations between these variables. One has $\mathcal{V} = \mathbf{X} \cup \{\mathbf{m}\}$ where $\mathbf{X}$ is the set of state vertices defined as $\mathbf{X} = \{\mathbf{x^1}, \ldots, \mathbf{x^n}\}$ and $\mathbf{m}$ is the input vertex. The edge set is $\mathcal{E} = \mathcal{E}_A \cup \mathcal{E}_B$, with $\mathcal{E}_A = \{(\mathbf{x^i}, \mathbf{x^j}) \,|A(i,j) \neq 0\}$ and $\mathcal{E}_B = \{(\mathbf{m}, \mathbf{x^i}) \,|B(i) \neq 0\}$. The entries of $A_{\rho(t)}$ correspond to the weights of the edges in the digraph. For convenience, we will denote by $\mathbf{v^j}$, $(j = 0, \ldots, n)$ a vertex of the digraph $\mathcal{G}(\Sigma_\Lambda)$ regardless of whether it is the input or a state vertex.

Given a triplet (n, r, n_a) with n the dimension of the state, r the delay and n_a the number of non-zero entries of the matrix A, the construction of the digraph $\mathcal{G}(\Sigma_\Lambda)$ related to the system Σ_Λ involves the following steps.

The system Σ_Λ is of dimension n and thus, the digraph $\mathcal{G}(\Sigma_\Lambda)$ involves $n+1$ vertices. The input is assigned to the vertex denoted by $\mathbf{v^0}$. The other n vertices are denoted by $\mathbf{v^1}, \ldots, \mathbf{v^n}$. Let $\mathbf{v^r}$ be the vertex that corresponds to the flat output $\mathbf{v^r}$.

Step 1: For, $i = 0, \ldots, n-1$, add the edges $(\mathbf{v^i}, \mathbf{v^{i+1}})$. There are r edges which connect $\mathbf{v^0}$ to $\mathbf{v^r}$. Hence, the delay of the automaton is r.

After Step 1, this line topology corresponds to quite trivial dynamical systems since it corresponds to state transition functions in the form of simple shifts. Let us recall that we aim at designing an automaton possibly involving state transition functions more general than T–functions. A shift is a special and trivial T–function. To this end, the following steps provide a way of adding edges $(\mathbf{v^i}, \mathbf{v^j})$ while guaranteeing flatness.

Step 2: Add the edges $(\mathbf{v^{r+i}}, \mathbf{v^{r+i+1}})$ for $i = 1, \ldots, n-r-1$. Step 2 allows vertex $\mathbf{v^j}$, $j = r+1, \ldots, n$ to have a predecessor. Indeed, if not so, the dynamics of the corresponding vertex $\mathbf{v^j}$ would reduce

to $x_{k+1}^j = 0$ and would be clearly useless. The resulting path is a so-called main directed path and is depicted in Figure 7.3.

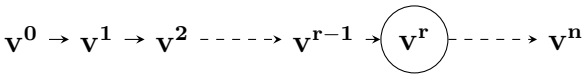


Figure 7.3: Digraph obtained after completion of Step 1-2. The vertex $\mathbf{v^r}$ corresponds to the flat output

Step 3: Add the edges $(\mathbf{v^r}, \mathbf{v^i})$, $i = 1, \ldots, n$ that connect the vertex $\mathbf{v^r}$ to any other vertices of the graph (except the vertex $\mathbf{v^0}$ related to the input).

Step 4: For every vertex $\mathbf{v^i}$, $i = 1, \ldots, r-1$, add the directed edge $(\mathbf{v^i}, \mathbf{v^j})$ for $j = 1, \ldots, i$.

The graph obtained after Step 1-4 is depicted in Figure 7.4.

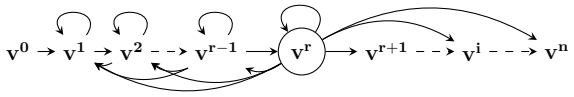


Figure 7.4: Graph obtained after Step 1-4.

Step 5: For every vertex $\mathbf{v_i}$, $i = r+1, \ldots, n$, add the directed edge $(\mathbf{v_i}, \mathbf{v_j})$ for $j = 1, \ldots, r$ and $j = i+2, \ldots, n$.

The resulting digraph after completion of Step 1-5 is depicted in Figure 7.5.

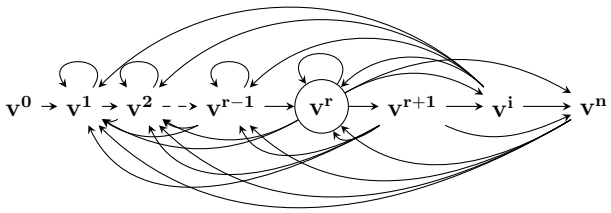


Figure 7.5: Graph obtained after completion of Step 1-5.

To sum up, the digraph $\mathcal{G}(\Sigma_\Lambda)$ is parametrized by the triplet (n, r, n_a). The number of vertices of the digraph is equal to $n+1$. Indeed, there are n vertices assigned to the state components and one assigned to the input. The delay r is the number of edges in the main directed path. The integer n_a defines the desired number of edges in the digraph $\mathcal{G}(\Sigma_\Lambda)$. It must satisfy $n_a \leq n_M$, where n_M is the maximal number of edges resulting from the construction Step 1-5. A simple counting leads to:

$$n_M = \frac{n(n+1)}{2} + r. \tag{7.30}$$

During the construction, at each step, we can decide whether we actually add the edges or not. That introduces flexibility in the perspective of providing distinct graphs and thus, distinct SSSC as detailed in Subsection 7.2.3.

Finally, the matrices I_A and I_B of the structural system Σ_Λ can be extracted from the adjacency matrix, denoted by $\mathcal{I}$, associated to the digraph $\mathcal{G}(\Sigma_\Lambda)$. Indeed, the adjacency matrix $\mathcal{I}$ associated to the digraph $\mathcal{G}(\Sigma_\Lambda)$ is the $(n+1) \times (n+1)$ matrix

$$\mathcal{I} = \left(\begin{array}{c|c} 0 & I_B^t \\ \hline 0 & \\ \vdots & I_A^t \\ 0 & \end{array} \right) \tag{7.31}$$

where I_A^t and I_B^t stands respectively for the transpose of the structured matrices I_A and I_B. The entries $\mathcal{I}_{ij}$ are defined as follows for $1 \leq i, j \leq n$

$$\mathcal{I}_{ij} = \begin{cases} 1 \text{ if there exists an edge from } \mathbf{v^j} \text{ to } \mathbf{v^i} \\ 0 \text{ otherwise.} \end{cases} \tag{7.32}$$

The adjacency matrix associated to $\mathcal{G}(\Sigma_\Lambda)$, obtained after completion of Step 1-5, is given by

$$\begin{array}{c} \\ \mathbf{v^0} \\ \mathbf{v^1} \\ \mathbf{v^2} \\ \mathbf{v^3} \\ \vdots \\ \mathbf{v^r} \\ \mathbf{v^{r+1}} \\ \vdots \\ \mathbf{v^{n-1}} \\ \mathbf{v^n} \end{array} \begin{array}{c} \begin{array}{cccccccccc} \mathbf{v^0} & \mathbf{v^1} & \mathbf{v^2} & \mathbf{v^3} & \cdots & \mathbf{v^r} & \mathbf{v^{r+1}} & \cdots & \mathbf{v^{n-1}} & \mathbf{v^n} \end{array} \\ \begin{pmatrix} 0 & 1 & 0 & 0 & \cdots & 0 & \cdots & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & \cdots & 0 & \cdots & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & \cdots & 0 & \cdots & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & \cdots & 0 & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & \cdots & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & \cdots & 1 & 0 & 1 & 1 & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \ddots & \vdots \\ 0 & 1 & 1 & 1 & \cdots & 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 & \cdots & 1 & 0 & 0 & 0 & 0 \end{pmatrix} \end{array}$$

The open source software Sagemath [76] has been used to elaborate the digraph $\mathcal{G}(\Sigma_\Lambda)$ corresponding to the triplet $(n = 40, r = 3, n_a = 120)$. The construction has been performed on an Intel CORE i7 CPU 2.26 GHz running Linux Ubuntu 14.04. All experiments ran single-threaded on the processors. It took 21 ms on the computer to obtain the digraph $\mathcal{G}(\Sigma_\Lambda)$.

Chapter 8

Implementation of some Sponge-based AEAD schemes

8.1 Introduction

We see more and more emerging areas in which small devices are interconnected, communicating wirelessly with one another and working together in order to accomplish various tasks (we can think about sensor networks, the Internet of Things or healthcare systems). Those devices are very limited in space and memory size while more and more functions need to be implemented in these always more compact objects. Due to the criticity and the need for confidentiality of the data transmitted, we need to get those objects as secured as possible. However the majority of cryptographic solutions developed until the 2000s were designed solely targeting desktops or server environments and are not fit for constrained devices.

The devices should be protected against a various range of cryptographic attacks, from private data theft to destructive attacks such as man-in-the-middle attacks. To prevent such attacks, cryptographic solutions should be able to provide confidentiality (ensure that an attacker cannot read sensitive data), integrity (prove that the data have not been modified), and authenticity (provide of proof of the identity of the sender). Authenticated encryption can answer all those needs using a single algorithm. This leads to improvement in performance and reduction of memory costs.

During DIAC 2013 [5], the cryptographic community started a common effort to update the state of the art in authenticated encryption which continued with the launch of the Competition for Authenticated Encryption: Security, Applicability and Robustness (CAESAR) [2]. From 2014 to 2019 all the submissions were carefully studied, from 58 in the first round to 17 in the third one. In 2019, six recommendations were made, two for each of the three different use cases: lightweight applications, high-performance applications and defense in depth, respectively ASCON[106], ACORN[234], AEGIS-128[235], OCB[166], Deoxys-II[146] and COLM[25].

In parallel, the NIST initiated several workshops on lightweight cryptography since 2015, which ended on the launch of a lightweight cryptography standardization process aiming to solicit, evaluate and standardize lightweight cryptography algorithms suitable for use in constrained environments in which the current standards are not acceptable. The point is to either develop new cryptographic solutions, enhance the existing ones or improve the implementations of current algorithms dedicated to authenticated encryption and optionally hash functions.

The proposed cryptographic solutions are authenticated encryption with associated data (AEAD) algorithms. This cryptographic function takes a four byte string as inputs and one byte-string as output. The four inputs are a variable-length plaintext, variable-length associated data, a fixed-length nonce, and a fixed-length key. The output is a variable-length ciphertext. The algorithms shall provide both authenticated encryption and decryption-verification. The cryptographic solutions should also provide confidentiality under adaptive chosen-plaintext attacks and integrity of the ciphertexts under adaptive forgery attempts and are expected to maintain security when the uniqueness of the nonce is

respected (any proof of security under nonce-reuse can be advertised as a feature). The minima in terms of size and security are a key of at least 128 bits, resisting attacks of at least 2^{112} computations on a classical computer in a single-key setting, and a key of 256 bits resisting attacks of at least 2^{224} computations. The recommended minimum sizes for nonce and tag are respectively 96 bits and 64 bits and the limit on the input sizes shall not be smaller than $2^{50}-1$ bytes. The entirety of the requirements are stated in [9].

57 candidates were proposed and 56 selected as Round 1 candidates. The focus during the first round was set on the cryptographic security of the submissions leading to two major selection criteria: maturation of the candidates and cryptanalysis of the candidates [224]. In September 2019, the NIST announced the candidates which made it to the round 2. For this second round, performance will play a larger role in the selection process, the goal being to obtain optimized versions fairly benchmarked on both software and hardware platforms.

We focused our efforts on sponge-based candidates as it appears to gather several interesting advantages as lightweight building technique and the design has been well studied those past years. We selected three of them because they each use different building techniques and it seemed constructive to compare them knowing that each candidate targets a different goal in terms of performances such as improved security margins, high throughput or power efficiency for example. This work is therefore oriented toward figuring out which construction leads to which performance enhancement and not simply on which one would be the best one. While some candidates were presented targeting software performances or a good trade-off between hardware and software efficiency, this work is only focused on hardware perfomances.
For this purpose, we implemented three of the round 2 candidates: ACE[13], WAGE[21] and PHOTON-BEETLE[40]. ACE is the first candidate in alphabetical order and shares an equivalent design with SPIX[20] and SpoC[19] which implementation results can already be found [192]. PHOTON-BEETLE is based on the Beetle sponge mode which design allows smaller state size for comparable security margins. WAGE is a stream cipher-based design. With the implementation of these candidates we cover a wide range of useful building techniques for sponge designs targeting AEAD.
We implemented these algorithms using the Hardware LWC API proposed [156] to fairly compare our results to those already available, as presented in the last section.

8.2 Implementation methodology

8.2.1 Hardware API for Lightweight Cryptography

The first public competition for cryptographic standards was the NIST call for the Advanced Encryption Standard [1]. During this competititon, the main criteria of competition were security and software performances; the hardware performances only came into consideration near the end of the standardization process. The same happened during the SHA-3 competition and there are still only a few referenced papers on hardware performances of the candidates. The real change came with the CAESAR competition, when a standard hardware API was developed and validated by the competition committee[141]. Hardware implementations were required for all submissions from round 3, with the development team helping candidates to use and develop their solutions. This API was not mandatory but helped define new guidelines for comparison for the second half of the competition. The implementation and linked results of each candidate can be found here [3].
For the NIST Lightweight Standardization process, no hardware implementation was mandatory but before the announcement of the round two, some candidates were found with a hardware implementation compliant with the CAESAR Hardware API. To fairly implement and compare the different candidates of the NIST standardization process for Lightweight Cryptography, a Framework for Benchmarking of Hardware Implementations was introduced and presented by Kaps et al. in [156] and [155]. Based on the framework developped for the CAESAR competition [2], it gives the developers a guideline to implement the different ciphers. This requires a few modifications to adapt the implementations based on the CAESAR API and the overhead from this API is well explained in [157]. With the beginning of round 2 in September 2019, more implementation compliant with this API are expected to create a fair basis for benchmarking hardware performances.

8.2.2 Specifications

In order to speed up the development process, both API feature a development package and an Implementer's Guide. Both designs use the same modules: the Preprocessor, FIFO, Postprocessor and the cryptographic core. The input data are split into two parts: the public data (pdi) and secret data (sdi) which are both received by the Preprocessor. After removing the header information, the data are passed to the cryptographic core which will implement the chosen cryptographic primitive (the designers only have to modify this part to implement their solution). The Postprocessor will then get the data output by the cryptographic core and add the API specific header data and send it to the output port (do). In addition to some changes, the LWC package also fully supports hash algorithms, which will not be considered in this chapter as all candidates don't specify one.

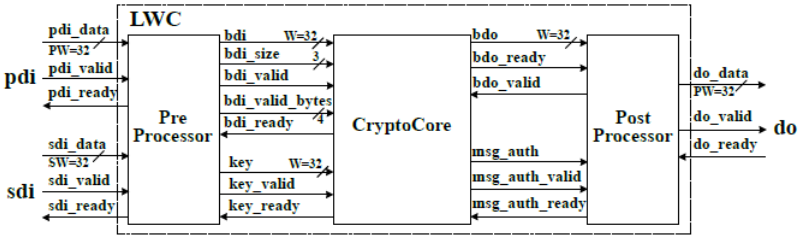


Figure 8.1: The LWC API

8.2.3 Implementation process

In order to get a fair comparison of all candidates, one method must be chosen and kept during the development process of each algorithm. We decided to follow the method chosen and explained in [192]. Each algorithm is developed following a basic-iterative construction, meaning that each step or round of an algorithm will be executed in one clock cycle. If the permutation of an algorithm is made of steps and rounds, such as ACE [13], one permutation will be executed in

$$number of steps * number of rounds$$

clock cycles (in this example $16 * 8 = 128$ clock cycles). Moreover, all implementations are fully compliant with the LWC Hardaware API.

8.3 Sponge function and sponge-based mode of operation

The Sponge construction was introduced by Bertoni et al. in [57] in 2007. It is based on a fixed-length permutation building a function mapping a variable-length input to a variable-length output. The construction was first used to build strong hash functions such as Keccak, the winner of the SHA-3 competition [12]. The sponge construction operates on a state of $b = r + c$ bits where b is called the width, r the rate and c the capacity. The data is padded and divided into words of r bits. The b bits of the state are initialized to zero. Then the construction proceeds in two phases. During the absorption, r bits are added to the state which goes through the f function. Once the whole data is absorbed, r bits are squeezed at a time with a call to the f function at each time.

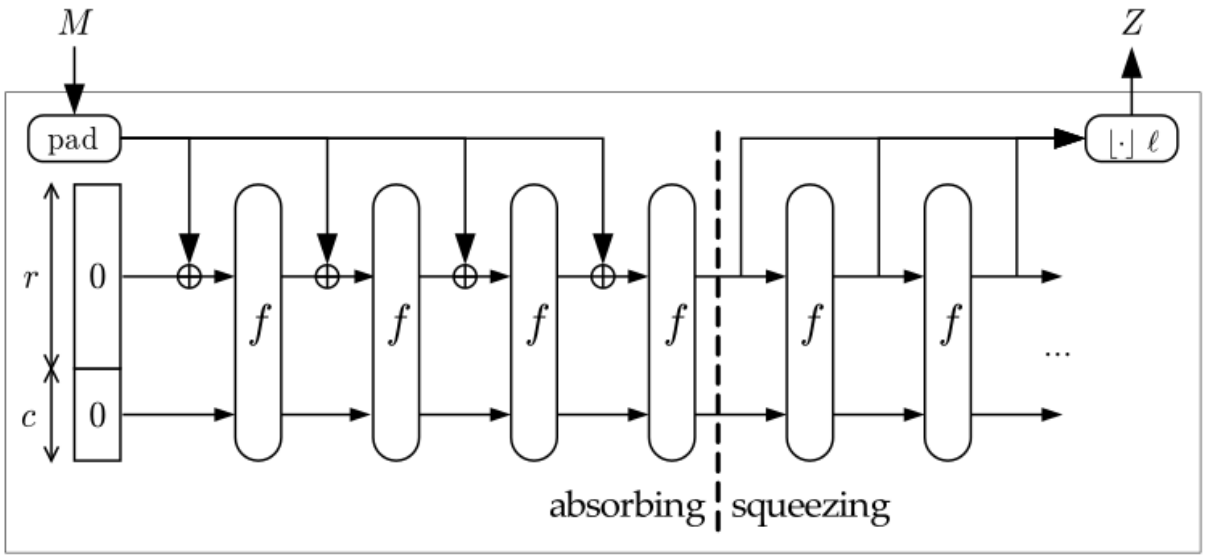


Figure 8.2: The sponge construction

From the sponge construction, keyed variants have been developed and have become a very popular mode of operation for a wide spectrum of cryptographic functionalities including authentication codes, pseudo-random functions, Extendable-Output Functions (XOFs)[12] and Authenticated Encryption modes.
The sponge construction allows the possibility to build lightweight design for hash functions. Since the introduction of the design, numerous lightweight hash algorithms have indeed been proposed, such as SPONGENT[67], Quark [31] and PHOTON[136]. One main advantage when compared to classical Merkle-Damgård hash functions is the size of the state which is fairly smaller.
The keyed Sponge principle also got adopted in Spritz, a new RC4-like stream cipher [197], and in 10 out of 57 submissions to the CAESAR[2] competition on authenticated encryption, with ASCON [106] as one of its best representative for this competition.

8.3.1 Existing security bounds

Encrypting using the Sponge construction is usually done via the Duplex construction[59]. As shown in Figure 8.3, it is a stateful construction made of two interfaces: one for initialization and one for duplexing. The initialization interface is used to initialize the state (which can be all-zero) while the duplexing interface absorbs a message of at most r bits and squeezes $\leq r$ bits of the outer part.

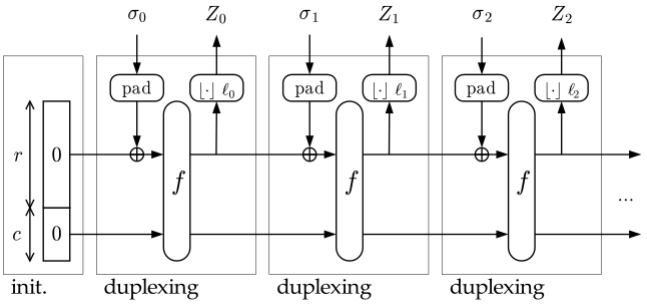


Figure 8.3: The Duplex construction

The security of the Duplex mode is directly linked to the indifferentiability of the classical Sponge, leading to a $O(2^{c/2})$ security bound. Bertoni et al.[59] showed that the Duplex construction, can be used for authenticated encryption in the form of SpongeWrap. This mode stands as the basis of the majority of Sponge-based submissions in the CAESAR competition and further work on improving sponge-based designs. Jovanovic et al. [153] claimed that Sponge-based constructions for authenticated encryption can achieve a significantly higher bound with a security of $min\{2^{b/2}, 2^c, 2^k\}$, with b the permutation

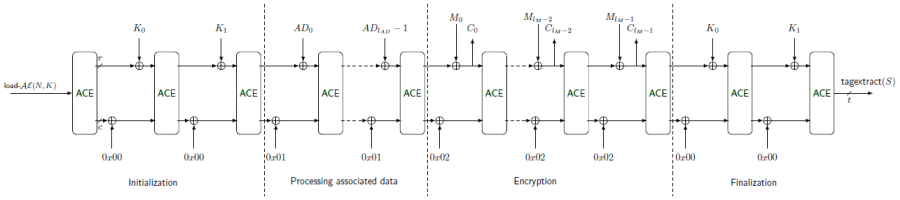


Figure 8.5: The ACE algorithm

size, c the capacity and k the key size. The author of the Beetle family of authenticated encryption ciphers [80] prove that these bounds do not stand for every application as an adversary can make a large number of decryption queries to mount the attack, which would then not satisfy the restrictions explained in [153]. The degraded security bound according to the Beetle authors is $min\{2^c, 2^k\}$.
The Beetle mode of operation is a duplex sponge mode combined with a feedback function such as in [81]. Using this feedback function only adds a little overhead in term of hardware ressources needed but helps to achieve the desired security bound. The authors claim a security of $min\{c - log(r), b/2, r\}$, with r the rate $r = b - c$.

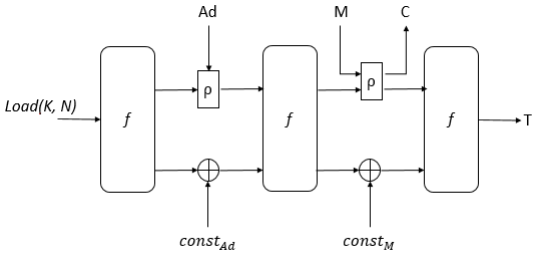


Figure 8.4: The Beetle construction

8.4 ACE: An Authenticated Encryption and Hash Algorithm

ACE [13] is a 320-bit sponge-based algorithm that can be used for both Authenticated Encryption with Associated Data and hashing functionalities. Its name comes from the strongest card of the deck. Following the designers' idea, ACE is designed to achieve a balance between hardware cost and software efficiency. The recommended parameter set states a rate r of 64 bits, a key of 128 bits, a nonce of 128 bits and a tag of 128 bits also. The capacity $c = b - r$ is then of 256 bits. This set of parameters is thought to be used for both AEAD and hash functionnalities.

ACE is based on the unified duplex sponge mode [22], a variant of the duplex sponge mode using the sLiSCP-light permutation. Two others NIST LWC candidates are based on the sLiSCP permutation: SPIX with a state of 256 bits [20] and SpoC with a state of 192 bits [19]. The sLiSCP-light uses a combination of a Type II Generalized Feistel Structure (GFS) and a round-reduced unkeyed Simeck box (SB). Each step consists of three transformations, namely, SubstituteSubblocks (SSb), AddStepconstants (ASc), and MixSubblocks (MSb). The non-linear operations are applied in the SSb, or SB. SBs consist of XORs, bitwise rotations, and a logical AND.
In the case of ACE, the 320-bit state is divided into five 64-bit words, A, B, C, D and E. The Simeck box is applied to three of the five words, namely to words A, C and E. The permutations are made in 16 steps.
In order to follow the basic-iterative construction, we built three SBs on 64-bits each, which operate on a total of 192 bits out of 320 bits of state. Round constants are supplied to each SB at the start of each SSb transformation. An SSb transformation requires 8 rounds, each of which executes in one clock cycle. Local state variables, as well as updated round constants, are stored during SSb transformations. The three round constants (rc0, rc1 and rc2) and three step constants (sc0, sc1 and sc2), each 8 bits, are implemented using look-up tables. The result of the C words through the second SB is then XORed to word B, and the same happens with words E and D. The three state constants are

added to words B, D and E. Finally, the MixSubblocks part mixes the 5 words to their corresponding output, as shown in Figure 8.6, one sLiSCP permutation is executed in 16 x 8 = 128 clock cycles.

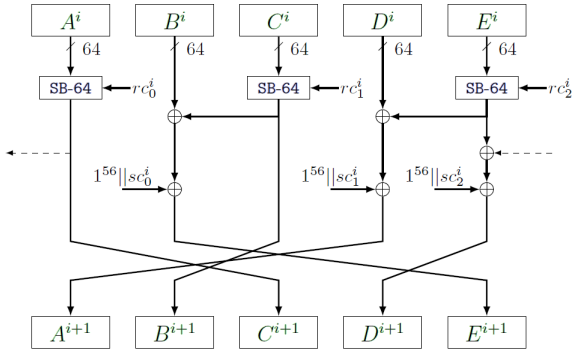


Figure 8.6: The ACE Permutation

The Simeck box shown in Figure 8.7 is an unkeyed independently parameterized variant of the round function of the Simon algorithm.

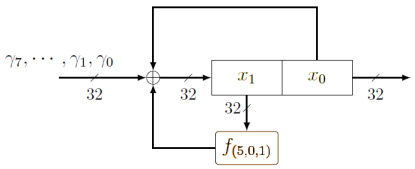


Figure 8.7: The Simeck Box

8.5 WAGE : An Authenticated Cipher

WAGE is a 259-bit AEAD algorithm design by Aagaard et al. [21]. Its mode of operation is an unified sponge duplex mode with an additionnal effort put for the initialization and the finalization. Its design is oriented towards hardware efficiency. The authors recommend using a rate r of 64 bits and a key, a nonce and a tag of 128 bits each.
The state is made of 259 bits and the permutation is based on the Welch-Gong stream cipher[187].

The state is divided into 37 7-bit words S_{36} to S_0. The rate part of the WAGE-AE-algorithm, called Sr is made of the last bit of S_{36} and $S_{35}, S_{34}, S_{28}, S_{27}, S_{18}, S_{16}, S_{15}, S_9, S_8$. The capacity part, of 195 bits is made of the rest of the state. The squeezing and absorbing phases of WAGE are shown in Figure 8.8 with the rate part shaded in orange and the green part representing the capacity.

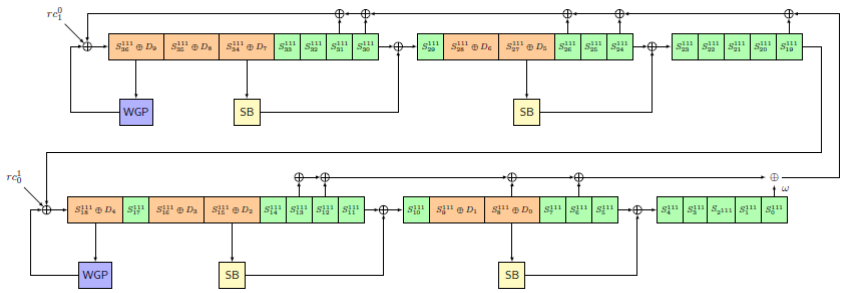


Figure 8.8: The squeezing and absorbing phases of WAGE

In case the processed data is not a multiple of the rate value, the padding rule for messages and additionnal data is the following : for a given vector X, $pad_r(X) \leftarrow X||1||0^{r-1-(|X| mod(r))}$.

The loading of the nonce and key consists of dividing it into 17 7-bits tuples of nonce and key plus one 7-bit tuple filled with the remaining bits and zeros and loading it by mixing nonce and key bits into the state.

The state is then initialized by loading the key in two blocks K_0 and K_1 to Sr and applying a WAGE permutation to the updated state.
The WAGE permutation operates over the finite field $\mathbb{F}_{2^7}$. Its round function is constructed by tweaking the initialization phase of the Welch-Gong cipher with four 7-bit SBoxes added to achieve a faster confusion and diffusion. The core components of the round function are an LFSR, two Welch-Gong Permutations (WGP) and four SBoxes. The state is updated 111 times as described next.
The Welch-Gong permutation is defined by:
$\mathrm{WGP}(x) = x^{13}+(x^{13}+1)^{33}+(x^{13}+1)^{39}+(x^{13}+1)^{41}+(x^{13}+1)^{104}$.
The state is updated as follow:

$$
\begin{aligned}
f_b &= S^i_{31} \oplus S^i_{30} \oplus S^i_{26} \oplus S^i_{24} \oplus S^i_{19} \oplus S^i_{13} \oplus Si_{12} \oplus S^i_8 \oplus S^i_6 \oplus (\omega \otimes S^i_{26}) \\
S^{i+1}_4 &= S^i_5 \oplus SB(S^i_8 \\
S^{i+1}_{10} &= S^i_{11} \oplus SB(S^i_{15} \\
S^{i+1}_{18} &= S^i_{19} \oplus WGP(S^i_8) \oplus rc^i_0 \\
S^{i+1}_{23} &= S^i_{24} \oplus SB(S^i_8) \\
S^{i+1}_{29} &= S^i_{30} \oplus SB(S^i_{15}) \\
S^{i+1}_{36} &= f_b \oplus SB(S^i_{36}) \oplus rc^i_1 \\
S^{i+1}_j &= S^i_{j+1}, j \in \{0, ..., 36\} \setminus \{4, 10, 18, 23, 29, 36\}
\end{aligned}
$$

with rc^i_0 and rc^i_1 round constants calculated with a LFSR of length 7.

8.6 PHOTON-BEETLE : An Authenticated Cipher

PHOTON-BEETLE[40] is an authenticated encryption and hash family which uses a sponge-based mode of operation called Beetle [80]. The innovation of this mode of operation resides in the feedback function used on the outer part of the state (the rate part). As stated in [80], this allows to achieve the required security bounds while keeping a relatively small state compared to other sponge-based designs. The feedback function used in the PHOTON-BEETLE algorithm is displayed in Figure 8.9.

$\rho(S, U)$
1 : $V \leftarrow \mathsf{Trunc}(\mathsf{Shuffle}(S), |U|) \oplus U$;
2 : $S \leftarrow S \oplus \mathsf{Ozs}_r(U)$;
return (S, V);

$\rho^{-1}(S, V)$
1 : $U \leftarrow \mathsf{Trunc}(\mathsf{Shuffle}(S), |V|) \oplus V$;
2 : $S \leftarrow S \oplus \mathsf{Ozs}_r(U)$;
return (S, U);

$\mathsf{Shuffle}(S)$
1 : $S_1 \| S_2 \xleftarrow{r/2} S$;
return $S_2 \| (S_1 \ggg 1)$;

Figure 8.9: PHOTON-BEETLE Feedback function

PHOTON-BEETLE operates a 256-bit state modified sponge mode of operation with 128 bits of key, nonce and tag prescribed. The prime recommendation from the author for the sole AEAD mode is a rate and a capacity of 128 bits. These parameters allow a good throughput while maintening sufficient security margins.

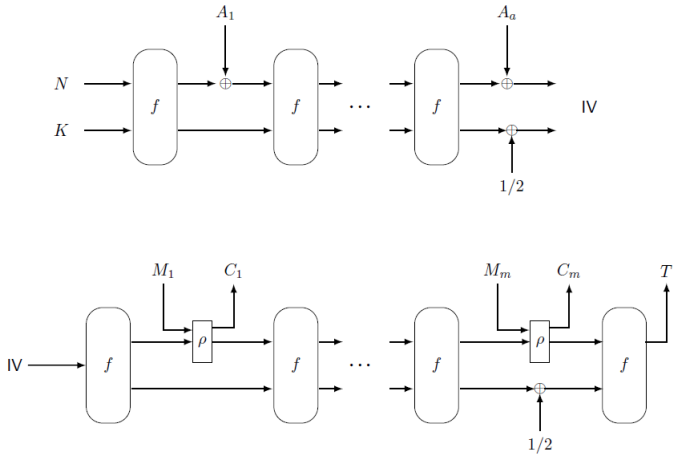


Figure 8.10: The PHOTON-BEETLE algorithm

PHOTON256[136] is used as the underlying 256-bit permutation. It is applied on a state of 64 elements of 4 bits each. The state is represented as a (8 x 8) matrix. The permutation is composed of 12 rounds, each iterating four layers: AddConstant, SubCells, ShiftRows and MixColumnSerial. AddConstant adds fixed constants, SubCells applies a 4-bit S-Box to each cells, ShiftRows rotates the position of the cells in each of the rows and MixColumnSerial linearly mixes all the columns independently using a serial matrix multiplication.

8.7 Hardware implementation of permutations

Comparison process All implementations are verified using Xilinx Vivado Simulator and the results are generated for Xilinx Artix-7 (xc7a100tcsg324-3), targeting area optimization. All candidates are implemented only in their AEAD form (if a hash functionnality exists, it is not targeted for this work) with the prime recommendation of the authors, as shown in Table 8.1.

Table 8.1: Characteristics of implemented ciphers

Cipher	key (bits)	nonce (bits)	steps	rounds	state (bits)	rate (bits)	tag (bits)
ACE	128	128	16	8	320	64	128
PHOTON-BEETLE	128	128	-	12	256	128	128
WAGE	128	128	111	-	259	64	128
SpoC-64	128	128	18	6	192	64	64
ASCON	128	128	-	6/12	320	64	128
SCHWAEMM	128	256	7/11	4	384	256	128

Table 8.2 details the number of clock cycles used for each step of each candidate and compares them using latency and throughput.
Latency is defined as the number of clock cycles needed to process one block of message from start to end (Initialization, computing and Finalization). While the throughput in Mbps is approximated by TP = f_{clk}* (bits/block) / (cycles/block) with f_{clk} the maximum frequency achieved by the design.

Table 8.2: Latency and throughput for selected ciphers

Ciphers	Encrypt	Latency	Throughput
ACE	651 + 130 x AD + 131 x PT	782	fclk x 64/131
PHOTON-BEETLE	23 + 14 x AD + 15 x PT	38	fclk x 128/15
WAGE	566 + 113 x AD + 114 x PT	680	fclk x 64/114
SpoC	219 + 109 x AD + 111 x PT	330	fclk x 64/111
Ascon	42 + 9 x AD + 10 x PT	52	fclk x 64/10
Schwaemm	114 + 43 x AD + 52 x PT	166	fclk x 256/52

Implementations are compared according to maximum frequency, area (given in LUTs), throughput and throughput-to-area ratio. We can thus compare our results to those of Hardware Implementation of NIST Lightweight Cryptographic Candidates: A First Look[192].

Table 8.3: Implementation results

Ciphers	Freq (MHz)	Area (LUTs)	TP (Mbps)	TPA (Mpbs/LUT)	Ref
ACE	128	1337	62.53	0.047	This work
PHOTON-BEETLE	139	**601**	1186.1	**1.974**	This work
WAGE	142	780	79.7	0.102	This work
SpoC	**268.0**	1172	154.5	0.132	[192]
Ascon-AEAD	263.0	1898	**1683.2**	0.887	[192]
Schwaemm	106.0	4313	521.8	0.121	[192]

From this table we can extract some interesting information. Looking at each candidate's parameters, it can be expected that smaller state size leads to smaller area occupation, which is not always true.
PHOTON-BEETLE has the lowest area occupation thanks to a relatively small state (256 bits) and a permutation known for its lightweight capabilities. Moreover, being designed targeting low latency leads to an excellent result considering throughput. WAGE, combining a 259-bits state with its LFSR-based permutation also achieves a small area occupation but this implementation only offers a throughput of 79.7 MHz. Coming third regarding area occupation, SpoC is designed to achieve sufficient security with the lowest state for a sponge-based design in the NIST project. Despite this state size of 192 bits, the mode of operation used for SpoC leads to a higher area occupation than the classical Beetle mode of operations which increases its total occupation. However, its implementation achieves the highest frequency with 268MHz. ACE and Ascon both use 320 bits of state and we can see that the ARX design of ACE allows a lower hardware footprint while ASCON achieves a better throughput.

8.8 Conclusion

In this work, we implemented three sponge-based algorithms in compliance with the provided Hardware API for Lightweight Cryptography. We compared the implementations with others from this paper[192] and completed the implementation process of NIST candidates for Lightweight Standardization Process.
This work presents and compares different algorithms, identifying different building techniques and providing a precised analysis on the performances that can be achieved on FPGA. One of the outcomes is that no algorithm presented really takes the lead as an outperformer when combining all studied characteristics. Indeed, when looking for the maximum achievable frequency, SpoC takes a

short lead on ASCON with 268 MHz. Ascon is the best regarding maximum throughput with 1683 Mbps. PHOTON-BEETLE achieves the smallest area occupation and the best throughput-to-area ratio by being at the same time very small to implement and achieving a very low latency.
To complete this work, future implementations of algorithms designed specifically for side-channel resistance and fault protection need to be done, targeting at first DryGASCON[194], ISAP[105] and Spook[48]. This will enrich our list of techniques and ideas, in particular those which enable a good resistance to physical and fault attacks.
Overall, the identified bricks will be useful to create variants or adaptations of the most promising lightweight AEAD solutions, enabling different use cases and especially targeting efficiency on hardware platforms and resistance against physical attacks.

Chapter 9

An attack on the SIV Mode of Operation

9.1 Introduction

A mode of operation is an algorithm that takes as core part a block cipher and uses it in a way to provide capabilities unachievable with a block cipher. A block cipher is only suitable for the cryptographic transformation of one fixed-length block.
Modes of operations can be used for different purposes : create a stream cipher, compute a Message Authentication Code (MAC), generate authenticated encryption, or create shorter or larger blocks. Several modes are defined and standardized by the National Institute for Standards and Technologies (NIST) to achieve confidentiality while using a properly chosen block cipher [111]. Other modes are defined to achieve authenticated encryption from a block cipher, which is the case of the CCM and the GCM modes of operation described in [113] and [112]. One mode is designed to perform authentication, namely CMAC [114].
In the first part, we present the notation and a quick reminder of what a block cipher and a MAC are. In the second part we introduce the different types of mode of operations standardized by the NIST. In the third part we present the SIV mode of operation and the attack we managed to conduct on. Finally we conclude this chapter.

9.2 Notations

This work is well inspired by the paper of Rogaway on the security of several modes of operations and we then decided to use the same notations he did [200]. By "strings" we mean (finite) binary strings. The empty string is written ϵ. The bit-length of a string X is written $|X|$. The byte-length of a byte string X is written $|X|_8$. Concatenation of strings A and B is written as $A||B$. We sometimes use conventional formal-language notation, like $\{0,1\}^*$ for binary strings or $(\{0,1\}^n)^+$ for strings having a positive multiple of n bits. The first t bits of a string X of length at least t is written $MSB_t(X)$, while the last t bits are $LSB_t(X)$. We write $A \oplus B$ for the bitwise exclusive-or of two equal-length strings. We also use this notation for the xor of unequal-length strings, where the understanding is to drop rightmost bits of the longer string until the two strings are of equal length, and then bitwise xor. We write $a \xleftarrow{\$} X$ for the experiment of randomly sampling from a space $\mathcal{X}$ and assigning the result to a. Either $\mathcal{X}$ will have a distribution associated to it or $\mathcal{X}$ will be finite and we mean the uniform distribution.

9.2.1 Block ciphers

A block cipher is a function $E : K \times \{0,1\}^n \to \{0,1\}^n$ such that $E_K(\cdot) = E(K,\cdot)$ is a permutation on $\{0,1\}^n$. The inverse of the block cipher E is $D = E^{-1}$ defined by $X = D_K(Y)$ being the unique $X \in \{0,1\}^n$ such that $E_K(X) = Y$. A typical block cipher is the AES [1]. Actually, to match the definition just given, the AES specification would be understood as any defining three block ciphers, AES-128, AES-192, and AES-256, one for each of three permitted key length. Typically a scheme's parameters, like the key length k for AES, must be fixed for the scheme, as described in the specification, to match the syntax we exploit in a definition. The usual way to quantify the security of a blockcipher $E : \mathcal{K} \times \{0,1\}^n \to \{0,1\}^n$ works as follows. Let's choose a random $K \xleftarrow{\$} \mathcal{K}$ and a random permutation π on n-bits. An adversary $\mathcal{A}$ is given black box access either to E_K or to π. The adversary tries to guess which kind of object it has. We let

$$Adv_E^{prp}(\mathcal{A}) = Pr[K \xleftarrow{\$} \mathcal{K} : \mathcal{A}^{E_K(\cdot)} \Rightarrow 1] - Pr[\pi \xleftarrow{\$} Perm(n) : \mathcal{A}_\pi(\cdot) \Rightarrow 1]$$

where Perm(n) denotes all the permutation on n-bit strings. An alternative measure of security for a blockcipher compares it against a random function instead of a random permutation. In that case we set

$$Adv_E^{prf}(\mathcal{A}) = Pr[K \xleftarrow{\$} \mathcal{K} : \mathcal{A}^{E_K(\cdot)} \Rightarrow 1] - Pr[\rho \xleftarrow{\$} Func(n,n) : \mathcal{A}^{\rho(\cdot)} \Rightarrow 1]$$

where Func(n, n) denotes the set of all functions from n-bit strings to n-bit strings. It is a standard result that $Pr[\mathcal{A}^\pi \Rightarrow 1] - Pr[\mathcal{A}^\rho \Rightarrow 1] \leq q^2/2^{n+1}$ for any adversary $\mathcal{A}$ that asks at most q queries. This makes the PRP (Pseudo-Random Permutation) and PRF (Pseudo-Random Function) notions of advantage close:

$$|Adv_E^{prp}(\mathcal{A}) - Adv_E^{prf}(\mathcal{A})| \leq q^2/2^{n+1}$$

if $\mathcal{A}$ asks q or fewer queries. The observation is sometimes known as the PRP/PRF Switching Lemma [82].

Yet another important security notion for blockciphers strengthens the requirement for resembling a random permutation by giving the adversary oracles not only for the forward direction of the cipher or the random permutation, but for the backwards direction, too. This is some-times called the "strong" notion of PRP security. It amounts to permitting a chosen-ciphertext attack (in addition to a chosen-plaintext attack). The definition of advantage becomes

$$Adv_E^{\pm prp}(\mathcal{A}) = Pr[K \xleftarrow{\$} \mathcal{K} : \mathcal{A}^{E_K(\cdot),E_K^{-1}(\cdot)} \Rightarrow 1] - Pr[\pi \xleftarrow{\$} Perm(n) : \mathcal{A}^{\pi(\cdot),\pi^{-1}(\cdot)} \Rightarrow 1]$$

It is now easy to see that a good $\pm prp$-security implies a good prp-security, but that the reverse does not hold. It is strongly believed that blockciphers like $E =$ AES are good in the PRP and strong-PRP senses of the word: "reasonable" adversaries have only a "small" value $Adv_E^{prp}(\mathcal{A})$ or $Adv \pm prp_E(\mathcal{A})$. We are not trying to define reasonable or small in this work. If one would like to specify an explicit assumption, one could say something like "we think that $Adv_{AES128}^{\pm prp}(\mathcal{A}) < 2^{-10}$ if $\mathcal{A}$ asks at most 2^{50} queries and uses at most 2^{90} time" (with time understood to include the adversary's description size and some particular computational model being fixed). But we certainly do not know how to prove any such statement, and making concrete conjectures like the one just given does not seem to have any real value.

Those blockciphers are what one might call conventional blockciphers; in particular: block size n is some fixed and rather small number, like $n = 64$ or $n = 128$, and the blockcipher is a "true" primitive. More generally, we can define a blockcipher as a function $E : K \times X \to X$ where $X \subseteq \{0,1\}^*$ is the message space and $E_K(\cdot) = E(K,\cdot)$ is a length-preserving permutation. The inverse to the blockcipher E is $D = E^{-1}$ defined by $D_K(Y)$ being the unique $X \in \{0,1\}^n$ such that $E_K(X) = Y$. To define the prp and $\pm prp$ security notions for blockcipher $E : K \times X \to X$ one adjusts the first and third equation to

$$Adv_E^{prp}(\mathcal{A}) = Pr[K \xleftarrow{\$} \mathcal{K} : \mathcal{A}^{E_K(\cdot)} \Rightarrow 1] - Pr[\pi \xleftarrow{\$} Perm(X) : \mathcal{A}^{\pi(\cdot)} \Rightarrow 1]$$

and

$$Adv_E^{\pm prp}(\mathcal{A}) = Pr[K \xleftarrow{\$} \mathcal{K} : \mathcal{A}^{E_K(\cdot),E_K^{-1}(\cdot)} \Rightarrow 1] - Pr[\pi \xleftarrow{\$} Perm(X) : \mathcal{A}^{\pi(\cdot),\pi^{-1}(\cdot)} \Rightarrow 1]$$

where Perm(X) denotes the set of all length-preserving permutations on X.

9.2.2 Message Authentication Code

Definition A message authentication code (MAC) is a block of information, or tag, used to authenticate a message (ensure that the message comes from the stated sender) and verify that it has not been modified. The MAC value protects both the integrity and the authenticity of a message, by allowing verifiers (who possess the secret key used to generate the MAC) to detect any changes to the message content.
A message authentication code system consists of three algorithms:

- A key generation algorithm that selects the key from the key space, uniformly at random.
- A signing algorithm that returns the tag computed from the key and the message.
- A verifying algorithm that verifies the authenticity of the message given the key and the tag. It returns either accepted if the MAC is correct or $\perp$ if not.

Standards Various standards exist that define MAC algorithms. These include:

- FIPS PUB 113 Computer Data Authentication, withdrawn in 2002, defined an algorithm based on DES.
- FIPS PUB 198-1 The Keyed-Hash Message Authentication Code (HMAC)
- ISO/IEC 9797-1 Mechanisms using a block cipher
- ISO/IEC 9797-2 Mechanisms using a dedicated hash-function
- ISO/IEC 9797-3 Mechanisms using a universal hash-function
- ISO/IEC 29192-6 Lightweight cryptography - Message authentication codes

ISO/IEC 9797-1 and -2 define generic models and algorithms that can be used with any block cipher or hash function, and a variety of different parameters. These models and parameters allow more specific algorithms to be defined by nominating the parameters. For example, the FIPS PUB 113 algorithm is functionally equivalent to ISO/IEC 9797-1 MAC algorithm 1 with padding method 1 and a block cipher algorithm of DES.

9.3 Modes of operation

When a block cipher is used in some scheme to accomplish a higher-level goal, we call the blockcipher using scheme a (blockcipher-based) mode of operation. Usually one sees the term mode of operation without the "block cipher" qualification; it is understood that the mode is based on a blockcipher. Besides using the block cipher, the mode may use other simple tools, such as simple bit manipulations, XOR operations, message padding, and even some finite-field arithmetic. If one started to use quite complex operations — such as a modular exponentiation or an elliptic-curve computation — one would probably draw the line and say that this no longer looked like a mode of operation: there is no precise meaning ascribed to the term modes of operation of a blockcipher, but we expect, somehow, that use of the blockcipher is the dominant thing that is going on. The goal of different modes of operation varies.

9.3.1 Confidentiality modes

The NIST currently specifies five modes of operation for block ciphers that achieve confidentiality. The first four modes (ECB, CBC, CFB and OFB) are classical, going back to FIPS 81 [11] (at which time the modes were specific to DES), more than 40 years ago.
The ECB mode is the simplest one as it only consists in several calls to the underlying block cipher up to the number of blocks that are needed to encrypt or decrypt. The disadvantage of this method is that identical plaintext blocks are encrypted into identical ciphertext blocks; thus, it does not hide data patterns well. In some senses, it does not provide a semantically secure encryption scheme, and is therefore not recommended for use in cryptographic protocols at all.
CBC is an improvement of ECB, where each block of plaintext is XORed with the previous ciphertext block before being encrypted. This way, each ciphertext block is dependent on all plaintext blocks processed up to that point. CBC has been the most commonly used mode of operation. Its main drawbacks are that encryption is sequential (i.e., it cannot be parallelized), and that the message must be padded to a multiple of the cipher block size.
CFB is closely related to CBC as it transforms a block cipher into a self-synchronizing stream cipher. The mode of operation is very similar; specifically, CFB decryption is almost identical to CBC encryption performed in reverse. CFB can be used as a self-synchronizing stream cipher mode of operation as described in 7.
OFB makes a block cipher into a synchronous stream cipher. It generates keystream blocks, which are then XORed with the plaintext blocks to get the ciphertext. Each output feedback block cipher operation depends on all previous ones, and so cannot be performed in parallel. However, because the plaintext or ciphertext is only used for the final XOR, the block cipher operations may be performed in advance, allowing the final step to be performed in parallel once the plaintext or ciphertext is available.
The fifth mode, CTR, was suggested by Diffie and Hellman just as early as the basic-four modes, yet was not included in the initial batch. CTR is often seen as the "best" choice among the classical confidentiality-only techniques. Its parallelizability and obvious correctness, when based on a good block cipher, makes it a good candidate in any modern portfolio of modes.

9.3.2 Authentication modes

The NIST only specifies one mode of operation for constructing a message authentication code from a block cipher, the cipher-based MAC or CMAC [114]. The original definition of the CMAC mode of operation was the Cipher Block Chaining MAC algorithm (CBC-MAC) that was then withdrawn for security deficiences. The design was then refined by Black and Rogaway and published under the name XCBC [129]. Iwata and Kurosawa proposed an improvement of XCBC and named the resulting algorithm One-Key CBC-MAC (OMAC) and later submitted OMAC1 as a refinement of OMAC. OMAC1 efficiently reduces the key size of XCBC.
The CMAC algorithm depends on the choice of an underlying symmetric key block cipher. NIST only approves the use of two block cipher algorithms, the AES [1] and the TDEA [41]. The CMAC mode does not exploit the inverse function of the block cipher.
The mode of operation accepts messages divided in blocks of size b, depending on the block cipher

used: $b = 128$ in th ecase of the AES and $b = 64$ in the case of the TDEA. The CMAC key is the block cipher key (the key, for short). The key is denoted K, and the resulting forward cipher function of the block cipher is denoted $CIPH_K$.
For a given block cipher and key, the input to the MAC generation function is a bit string called the message, denoted M. The bit length of M is denoted $Mlen$. In principle, there is no restriction on the lengths of messages. In practice, however, the system in which CMAC is implemented may restrict the length of the input messages to the MAC generation function.
The output of the MAC generation function is a bit string called the MAC, denoted T with a length denoted $Tlen$, a parameter that shall be fixed for all invocations of CMAC with the given key.
During the computation, subkeys are needed and are computed using the zero vector (a vector made of zeros with a length of b). (details on the exact computation method of subkeys K_1 and K_2 can be found in [114]).

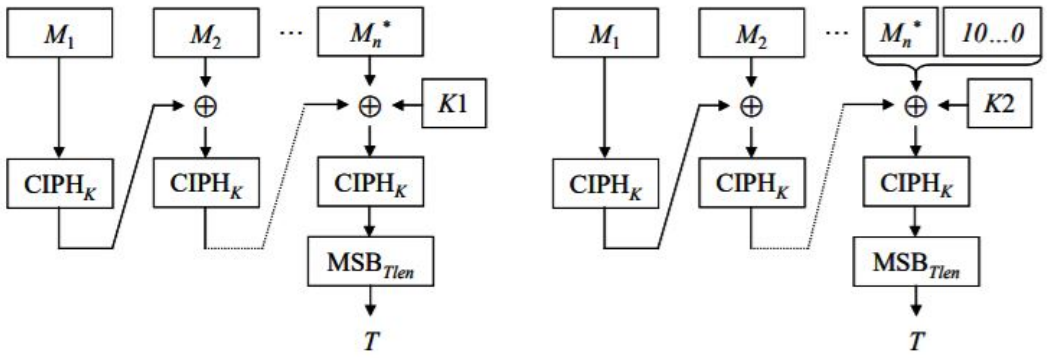


Figure 9.1: the CMAC computation with two cases

The two cases of MAC Generation are illustrated in the figure above. On the left is the case where the message length is a positive multiple of the block size; on the right is the case wherethe message length is not a positive multiple of the block length and shall therefore be padded.

9.3.3 Authenticated-encryption modes

Currently, the NIST has standardized two modes for authenticated encryption, CCM [113] and AES-GCM [112]. A third one is currently in the scope of a potential standardization: the AES-GCM-SIV [134].
CCM is a nonce-based AEAD scheme that combines CTR mode encryption and the raw CBC-MAC. It is inherently serial, which can limit speed in some contexts. It is provably secure with good bounds, assuming the underlying block cipher is a good Pseudo-Random Permutation. The mode is widely standardized and used. It is simpler to implement than GCM because it does not require the multiplication and can be used as a nonce-based MAC.
GCM is a nonce-based AEAD scheme that combines CTR mode encryption and a GF(2^{128})-based universal hash function. It has good efficiency characteristics for some implementation environments. It also has good provably-secure results assuming minimal tag truncation but there exist attacks that lead to poor provable-security bounds in the presence of substantial tag truncation. It can also be used as a nonce-based MAC, which is then called GMAC. The recommendation using this mode of operation is to restrict nonces to 96-bits and tags to at least 96 bits. This mode is also widely standardized and used.
Both are nonce-based (meaning that the user must supply a never-repeating IV). Both allow "associated-data" (for example, a message header which can be constituted of metadata describing the message) to be authenticated alongside whatever is being encrypted. The methods are called AEAD (authenticated-encryption with associated-data) schemes.

9.3.4 Deterministic Authenticated Encryption modes

A deterministic algorithm is an algorithm that will always output the same results given the same inputs. That is usually a characteristic that is not desired in cryptography as it can lead to several attacks. Therefore, mode of operations are most of the time defined with Initialization Vectors (IV) or nonces and specified with the notion that the triplet message, key and IV shall not be used more than once. However some applications or communication protocols cannot carry or use IV and thus fall under other security definitions. Rogaway defined first the notion of deterministic authenticated-encryption or DAE in his paper on the Key-Wrap problem [202].
A key-wrap scheme is a shared-key encryption scheme that aims to provide privacy and integrity protection for data such as cryptographic keys without relying on the use of a nonce or any random bits (IV or counters). The goal of such a scheme is to protect cryptographic keys that are encrypted as the message, while a header composed for example of the meta-data of the associated keys is used and authenticated.
So key-wrap's purpose is to remove Authenticated Encryption's reliance on a nonce or random bits. At least in the context of transporting cryptographic keys, a deterministic scheme should be just as good as a probabilistic one, anyway. Another goal of key wrap is to provide integrity protection for cleartext associated data, typically control information about the wrapped key.

9.4 Our contribution: Attack on SIV

9.4.1 The SIV Mode of Operation

There exist several algorithms that can be used to achieve deterministic authenticated encryption without the use of an initialization vector, such as the HBS mode of operation [145]. We decided to focus our study on the SIV mode of operation which is, to the best of our knowledge the first candidate to tackle this problem. The Synthetic Initialization Vector (SIV) Authenticated Encryption is a mode of operation described in [199]. The algorithm is composed of two operations:

- The S2V operation that takes as input the plaintext, the authentication key, plus the optional additional data and generates a tag T. The computation is a CBC-MAC using the AES as internal block cipher.
- The other operation, that we will call from now on S1V, uses the tag T as initialization vector and encrypts the plaintext using the AES-CTR mode of operation.

The mode of operation is based on one block cipher that will compute blocks of 128 bits. Usually, the recommended block cipher is the AES. Two keys are used, one for each operation. The mode is defined with a key K that will be divided into the two keys K_1 for the S2V operation and K_2 for the S1V operation. K can be of 256, 384 or 512 bits.

The S2V operation is a slightly modified CMAC mode of operation. It consists of the doubling and Xoring of the outputs of the block-cipher used for this mode of operation, computing each individual 128 bits strings of the input. The first operation is the computation of the zero vector. Then each block of input is computed and the result is Xored with the doubling of the previous ciphered block (multiplication by 2 (mod ()2^n). The last block is padded if needed (if the size of the last block is less than 128 bits). Finally, the result is computed through a last instantiation of the block cipher to output the Synthetic Initialization Vector, that is also used as the authentication tag, called V.

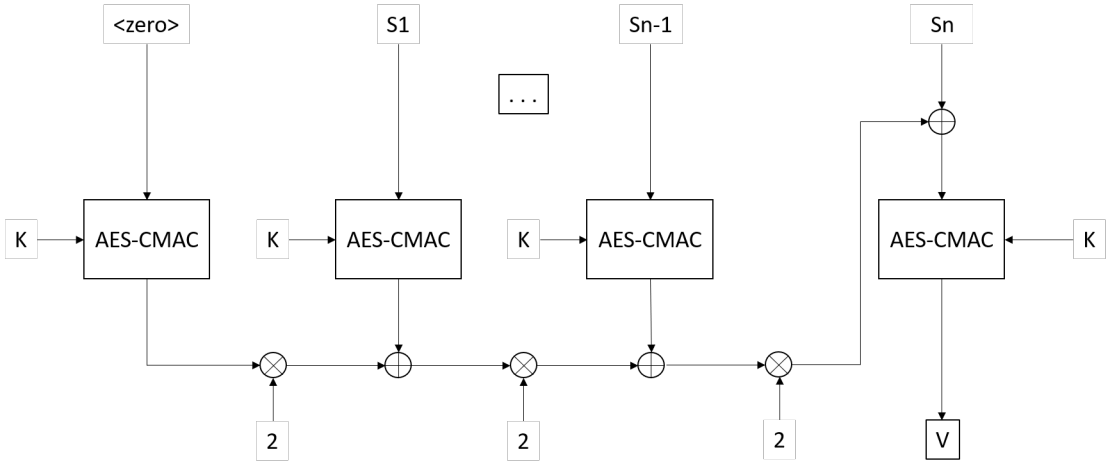


Figure 9.2: the S2V operation

The second part of the mode of operation consists of a Counter mode of operation taking as inputs the blocks of message and the Synthetic Initialization Vector resulting from the S2V operation. Each block are computed independently from each other, by Xoring them to the output of the block cipher. Each block M_i is Xored to the output X_i resulting of the computation of the counter which is the Synthetic Initialization Vector concatenated with the incremented counter i. Put in equation :

$$X_i = E_K(V||i); C_i = M_i \oplus X_i \tag{9.1}$$

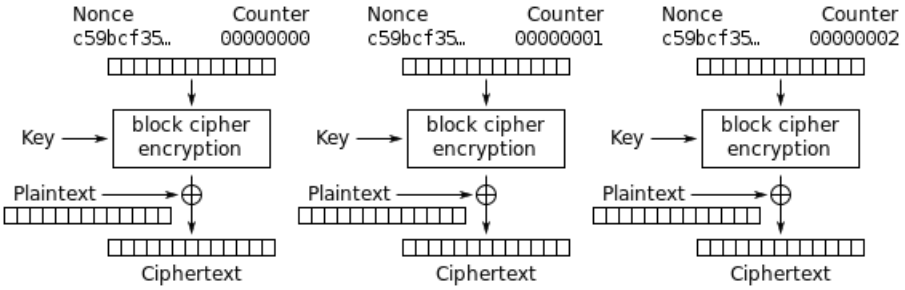


Figure 9.3: the CTR operation

Put together, the SIV mode of operation will first compute the Synthetic Initialization Vector V from the header and the message. This SIV that will be used both as authentication tag and nonce for the Counter Mode of operation. The message will then be processed through the CTR mode of operation to compute the ciphertext C. The result C is concatenated with V to produce the end result $Z = V||C$.

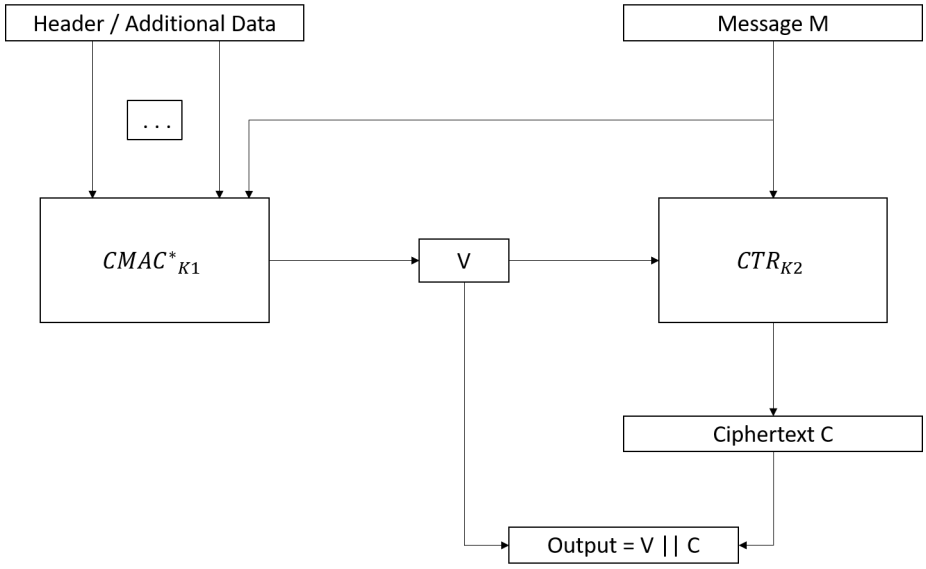


Figure 9.4: the SIV mode of operation

9.4.2 The attack

Consider a plaintext M such as $M = (M_0, M_1, ..., M_n)$.
Consider a SIV oracle that takes as input $(M_0, M_1, ...M_n)$ and returns $(V, C_0, C_1, ..., C_n)$ with V the tag resulting of the computation of the plaintext through the S2V operation and C the ciphertext such as $C = (C_0, C_1, ..., C_n)$.
An attacker having access to this SIV oracle will start by sending requests with k messages of two blocks (M_0^α, M_1^α), receiving k answers $(V^\alpha, C_0^\alpha, C_1^\alpha)$, with $\alpha \in \{0, k\}$.
According to the birthday bond, in $2^{n/2}$ messages the attacker will have two different messages (M_0^i, M_1^i) and (M_0^j, M_1^j) such as $V^i = V^j$.
The attacker then sends two challenges with messages of only one block each, namely M_0^i and M_0^j to obtain T_i and T_j.
It is then easy to extend these messages to two blocs (M_0^i, M_1^i) and (M_0^j, M_1^j) such as the S2V operation returns the same T_i and T_j by selecting $M_1^i = M_0^i \oplus T_i$ and $M_1^j = M_0^j \oplus T_j$.
The attacker can then recursively extend the plaintext, one block at a time while obtaining the same T_i or T_j.
The CTR mode of operation computes the value of $AES_K(T_i + cpt)$ Finally, knowing the ciphertext C^i corresponding to a plaintext M^i, the attacker can retrieve the ciphertext C^j from M^j. Hence we do have a distinguisher in $\mathcal{O}(2^{n/2})$.

9.5 Conclusion

The SIV mode of operation is described in several different documents, each with different security notions or security requirements. Our analysis clarifies these notions and shows that no protocol using the SIV mode of operation should be used with more than $2^{64} - 1$ messages under the same key. This is reminiscent of other papers, such as the CMAC paper [114] that specifies that no protocol using the CMAC mode of operation should be used with more than 2^{48} messages under the same key, but does not provide analysis explaining this limit.

Part IV

Outlooks and future work

There were two main goals that drove me through this work: developing and implementing on FPGA a new family of customizable block ciphers with an appropriate mode of operation and a lightweight, side-channel prone, authenticated encryption scheme. At the moment I finish writing this thesis, I have defined, specified and implemented on FPGA the new family of proprietary algorithms, in order to fulfill their primary need. The details about those algorithms cannot be disclosed here, however, I tried to give as much incentives as possible on the work that I have led during the last years to achieve those results.

The research for the state-of-the-art presented in part II filled various purposes: first of all, for me it was a way of learning new techniques, review and better apprehend different notions and cryptographic primitives. Secondly, I hope it helps the reader to better understand the context of this thesis and brings reminders on the different notions. Lastly, I used those studies to begin an introduction to cryptographic notions for those of my coworkers that felt the need to embrace the cryptographic world and its applications.

Particularly, the chapter about block ciphers, their design and application was a very important topic for the work I did during those years and the work to come, as I had to create a design for a peculiar application. In the context of HENSOLDT's applications, the protocols I have been working on entail that both the block cipher and the mode of operation can not follow the current cryptographic standards. Hence a particular effort is required to better understand what has been done in the field of block ciphers and design of modes of operation to be able to tweak it to cope with our needs.

This chapter leads naturally to the work realized on the deterministic authenticated encryption and the attack we proposed against the SIV mode of operation. Some applications such as Key-wrapping mechanisms require the use of a mode of operation that offers confidentiality, integrity and authenticity, even in protocols were no random value (such as Initialization Vector (IV) or nonce) are provided. In other words, it needs to hold its security properties in the case of nonce-misuse. In the event of protecting such protocols, the need to define the boundaries of allowed quantity of messages processed using the same key is very important as it can lead to disastrous attacks. In this scope, we provided an attack on the SIV mode of operation in the event of a protocol allowing at least 2^{64} messages processed using the same key.

This work will be carried on for one of the main future needs for HENSOLDT applications: developing a new mode of operation usable with a lightweight block cipher, using out of standards messages without common protocols. The SIV mode of operation was looked upon because it provides security without the use of a proper IV which is an interesting research strategy for this particular application. Security of such a mode of operation and its limitations are a good basis for a future research. The next step will be to design a new mode of operation, that can provide good security boundaries without the use of an actual IV and then implement it on FPGA.

Speaking of peculiar protocols, some applications need the ability to auto-recover in the case of data loss during the transfer. The Self-Synchronizing Stream Cipher Stanislas is, to the best of our knowledge, the only Self-Synchronizing Stream Cipher (SSSC) scheme that has not already been broken. Although we encourage the cryptographic community to give a try to attacking it, we have good hopes that this scheme can be of some use. Indeed, the self-synchronizing capability, with only a few 40 bits lost can be used to secure some protocols, such as SCADA protocol, and could find some interesting application on proprietary protocols. Moreover, even if the algorithm is not very efficient in terms of area occupation, at least implemented as we did, straightforwardly, its relatively high throughput offers the potential to secure even high performances applications. A future step on this algorithm would be to look after the best possibilities to secure this scheme against side-channel analysis and study the impact on area occupation. Furthermore, provide an optimized implementation on FPGA could also be a good add to see if this algorithm could be used in some embedded protocols.

Finally, the part that took most of the time was the study and design and implementation of an authenticated encryption scheme. I started by studying the different methods to achieve such schemes, namely the three approaches Encrypt-then-MAC, Encrypt-and-MAC and MAC-then-Encrypt. Then

I followed the two cryptographic competitions: CAESAR and the NIST LWC standardization process to gather data on the last techniques and improvements in terms of designs, security and performances (throughput and area occupation).
I then focused on the sponge construction for several reasons. It is a well understood and studied scheme, as it is the basis of the SHA-3 hash standard. I find the design quite easy to understand and tweak if needed and quite elegant. Moreover, it achieves good performances in terms of both area occupation and throughput, with for example ASCON, first choice at the term of the CAESAR competition in the lightweight solutions use case and ranking in the top three of round 2 at the NIST LWC standardization process in terms of throughput [185]. This last point is helped by the fact that sponge-based designs are one-pass for verification and decryption mechanisms, as opposed as most of other schemes that first need one pass to verify the tag and another one for decryption. In the purpose of defining which one would be the best choice depending on the application requirements, I implemented some of the round 2 candidates on FPGA based on different design choices: ACE whose permutation is based on the Simeck algorithm, WAGE, which has a peculiar permutation on 259 bits, and PHOTON-BEETLE which employs a modified mode of operation; the BEETLE mode, in order to achieve the same security while using a shorter internal state. I compared these implementations with the ones from [192] which displayed SpoC; another design made to shorten the internal state; ASCON and Schwaemm.
Finally, I started focusing on the different choices of permutations that could had been designed by the sponge-based designs community in order to find the most interesting depending on the needs. We now have a large spectrum of choices in terms of sponge-based mode of operation (with for example DuplexSponge, BEETLE, SpoC, ISAP, TETSponge, TEDTSponge) and a large choice of permutation, that can be tweaked in order to achieve the required size. The future designs can be created by using those different blocks as LEGO®blocks and by designing a scheme with requirements needs in terms of area occupation, throughput, throughput-to-area ratio, latency and side-channel security.
The next step in this direction could be to try to improve the existing FPGA implementation of the candidates of the NIST LWC standardization process that made it up to round three, using a sponge-based design. Moreover, a work using the different blocks and giving a proper FPGA implementation of every possible mixes is an interesting topic of research as it could help better apprehend the whole possibilities offered by sponge-based designs for authenticated encryption.

The work done on the Useful-Work protocol, presented in Appendix A was for me a first step to the blockchain universe that I have found very interesting. My contribution to this work helped me better apprehend this domain that has, in my opinion, a lot to offer in the near future as some new challenges appear to bring a lot of attention. If this protocol finds its public, the next step would be to define potential applications and implement them to create a new type of cryptocurrency rewarding useful works.

Bibliography

[1] AES: the advanced encryption standard, https://competitions.cr.yp.to/aes.html

[2] Crypto competitions: CAESAR: Competition for Authenticated Encryption: Security, Applicability, and Robustness, https://competitions.cr.yp.to/caesar.html

[3] Database of FPGA results for authenticated ciphers, https://cryptography.gmu.edu/athenadb/fpga_auth_cipher/table_view

[4] Diac 2013: Directions in authenticated ciphers, http://2013.diac.cr.yp.to/

[5] DIAC 2013: Introduction, http://2013.diac.cr.yp.to

[6] Ecrypt symlab - symmetric techniques virtual lab, https://www.ecrypt.eu.org/ecrypt2/symlab/

[7] estream: the ecrypt stream cipher project, https://www.ecrypt.eu.org/stream/

[8] Nist lightweight cryptography standardization, https://csrc.nist.gov/projects/lightweight-cryptography

[9] Submission Requirements and Evaluation Criteria for the Lightweight Cryptography Standardization Process. Tech. rep., National Institute of Standards and Technology, https://csrc.nist.gov/CSRC/media/Projects/Lightweight-Cryptography/documents/final-lwc-submission-requirements-august2018.pdf

[10] Vampire - virtual applications and implementations research lab, http://hyperelliptic.org/ECRYPTII/vampire/

[11] Des modes of operation. Federal Information Processing Standards Publication 81 (december 1980), https://csrc.nist.gov/csrc/media/publications/fips/81/archive/1980-12-02/documents/fips81.pdf

[12] Sha-3 standard: Permutation-based hash and extendable-output functions. Tech. rep., National Institute of Standards and Technology, Gaithersburg, MD (2015). https://doi.org/10.6028/NIST.FIPS.202, https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.202.pdf

[13] Aagaard, M., AlTawy, R., Gong, G., Mandal, K., Rohit, R.: ACE: An authenticated encryption and hash algorithm p. 49 (2019)

[14] Adams, C.M.: CAST-256, a submission for the advanced encryption standard. First AES Candidate Conference (1998)

[15] Agency, N.S.: SKIPJACK and KEA algorithm specifications (1998), https://csrc.nist.gov/CSRC/media//Projects/Cryptographic-Algorithm-Validation-Program/documents/skipjack/skipjack.pdf

[16] Al-Kindi: Al-Kindi s Treatise On Cryptanalysis (2003)

[17] Albrecht, M., Bernstein, D., Chou, T., Cid, C., Gilcher, J., Lange, T., Maram, V., vonMaurich, I., Misoczki, R., Niederhagen, R., Paterson, K., Persichetti, E., Peters, C., Schwabe, P., Sendrier, N., Szefer, J., Tjhai, C., Tomlinson, M., Wang, W.: Classic mceliece:conservative code-based cryptography (2020), https://classic.mceliece.org/nist/mceliece-20201010.pdf

[18] Alkim, E., Bos, J.W., Ducas, L., Longa, P., Mironov, I., Naehrig, M., Nikolaenko, V., Peikert, C., Raghunathan, A.: FrodoKEM learning with errors key encapsulation (2020), https://frodokem.org/files/FrodoKEM-specification-20200930.pdf

[19] AlTawy, R., Gong, G., He, M., Jha, A., Mandal, K., Nandi, M., Rohit, R.: SpoC: An Authenticated Cipher Submission to the NIST LWC Competition p. 31 (2019)

[20] AlTawy, R., Gong, G., He, M., Mandal, K., Rohit, R.: Spix: An Authenticated Cipher Submission to the NIST LWC Competition p. 38 (2019)

[21] AlTawy, R., Gong, G., Mandal, K., Rohit, R.: Wage: An authenticated encryption with a twist. Cryptology ePrint Archive, Report 2020/435 (2020), https://eprint.iacr.org/2020/435

[22] AlTawy, R., Rohit, R., He, M., Mandal, K., Yang, G., Gong, G.: sLiSCP: Simeck-Based Permutations for Lightweight Sponge Cryptographic Primitives. In: Adams, C., Camenisch, J. (eds.) Selected Areas in Cryptography – SAC 2017, vol. 10719, pp. 129–150. Springer International Publishing (2018), series Title: Lecture Notes in Computer Science

[23] Anderson, D.P., Cobb, J., Korpela, E., Lebofsky, M., Werthimer, D.: Seti@home: An experiment in public-resource computing. Commun. ACM (2002)

[24] Anderson, R., Biham, E.: Two practical and provably secure block ciphers: Bear and lion. In: Gollmann, D. (ed.) Fast Software Encryption. pp. 113–120. Springer Berlin Heidelberg, Berlin, Heidelberg (1996)

[25] Andreeva, E., Bogdanov, A., Data, N., Luykx, A., Bart, M., Nandi, M., Elmar, T., Yasuda, K.: COLM v1 p. 12 (2016)

[26] Aoki, K., Ichikawa, T., Kanda, M., Matsui, M., Moriai, S., Nakajima, J., Tokita, T.: Camellia: A 128-bit block cipher suitable for multiple platforms — design andanalysis. In: Stinson, D.R., Tavares, S. (eds.) Selected Areas in Cryptography. pp. 39–56. Springer Berlin Heidelberg, Berlin, Heidelberg (2001)

[27] Aragon, N., Barreto, P., Bettaieb, S., Bidoux, L., Blazy, O., Deneuville, J.C., Gaborit, P., Gueron, S., Guneysu, T., Melchor, C.A., Misoczki, R., Persichetti, E., Sendrier, N., Tillich, J.P., Zemor, G., Vasseur, V., Ghosh, S.: BIKE: Bit flipping key encapsulation (2020), https://bikesuite.org/files/v4.1/BIKE_Spec.2020.10.22.1.pdf

[28] Arnault, F., Berger, T.P., Lauradoux, C., Minier, M., Pousse, B.: A new approach for fcsrs. In: Selected Areas in Cryptography - SAC 2009. Lecture Notes in Computer Science, vol. 5867, pp. 433–448. Springer (2009)

[29] Arnault, F., Berger, T.P., Minier, M., Pousse, B.: Revisiting lfsrs for cryptographic applications. IEEE Trans. Information Theory 57(12), 8095–8113 (2011)

[30] Aumasson, J.P., Bernstein, D.J., Beullens, W., Dobraunig, C., Eichlseder, M., Fluhrer, S., Gazdag, S.L., Hülsing, A., Kampanakis, P., Kölbl, S., Lange, T., Lauridsen, M.M., Mendel, F., Niederhagen, R., Rechberger, C., Rijneveld, J., Schwabe, P., Westerbaan, B.: Sphincs+ (2020), https://sphincs.org/data/sphincs+-round3-specification.pdf

[31] Aumasson, J.P., Henzen, L., Meier, W., Naya-Plasencia, M.: Quark: A lightweight hash. In: Mangard, S., Standaert, F.X. (eds.) Cryptographic Hardware and Embedded Systems, CHES 2010. pp. 1–15. Springer Berlin Heidelberg, Berlin, Heidelberg (2010)

[32] Aumasson, J.P., Neves, S., Wilcox-O'Hearn, Z., Winnerlein, C.: Blake2: simpler, smaller, fast as md5. Cryptology ePrint Archive, Report 2013/322 (2013), https://eprint.iacr.org/2013/322

[33] Babbage, S.: A Space/Time Trade-Off in Exhaustive Search Attacks on Stream Ciphers. In: European Convention on Security and Detection. No. 408, IEEE Conference Publication (1995)

[34] Babbage, S., Dodd, M.: The MICKEY stream ciphers 4986 (2008)

[35] Bai, S., Ducas, L., Kiltz, E., Lepoint, T., Lyubashevsky, V., Schwabe, P., Seiler, G., Stehlé, D.: Crystals-dilithium algorithm specifications and supporting documentation (2021), https://pq-crystals.org/dilithium/data/dilithium-specification-round3-20210208.pdf

[36] Balasch, J., Faust, S., Gierlichs, B.: Inner product masking revisited. Cryptology ePrint Archive, Report 2015/105 (2015), https://eprint.iacr.org/2015/105

[37] Baldominos, A., Saez, Y.: Coin.ai: A proof-of-useful-work scheme for blockchain-based distributed deep learning (2019)

[38] Ball, M., Rosen, A., Sabin, M., Vasudevan, P.N.: Proofs of useful work. IACR Cryptol. ePrint Arch. (2021), http://eprint.iacr.org/2017/203

[39] Banik, S., Bogdanov, A., Isobe, T., Shibutani, K., Hiwatari, H., Akishita, T., Regazzoni, F.: Midori: A block cipher for low energy (extended version). Cryptology ePrint Archive, Report 2015/1142 (2015), https://eprint.iacr.org/2015/1142

[40] Bao, Z., Chakraborti, A., Datta, N., Guo, J., Nandi, M., Peyrin, T., Yasuda, K.: PHOTON-beetle : Authenticated encryption and hash family p. 14

[41] Barker, E., Mouha, N.: Recommendation for the triple data encryption standard (tdea) block cipher. NIST Special Publication 800-67, Revision 2 (2017), https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-67r2.pdf

[42] Basso, A., Bermudo Mera, J., D'Anvers, J.P., Karmakar, A., Roy, S., Van Beirendonck, M., Vercauteren, F.: Saber: Mod-lwr based kem (round 3 submission) (2020), https://www.esat.kuleuven.be/cosic/pqcrypto/saber/files/saberspecround3.pdf

[43] Böck, H., Zauner, A., Devlin, S., Somorovsky, J., Jovanovic, P.: Nonce-disrespecting adversaries: Practical forgery attacks on gcm in tls. Cryptology ePrint Archive, Report 2016/475 (2016), https://eprint.iacr.org/2016/475

[44] Beaulieu, R., Shors, D., Smith, J., Treatman-Clark, S., Weeks, B., Wingers, L.: The SIMON and SPECK families of lightweight block ciphers. Cryptology ePrint Archive, Report 2013/404 (2013), https://eprint.iacr.org/2013/404

[45] Beauregard, S.: Circuit for shor's algorithm using 2n+3 qubits. Quantum Inf. Comput. 3(2), 175–185 (2003). https://doi.org/10.26421/QIC3.2-8, https://doi.org/10.26421/QIC3.2-8

[46] Beberg, A.L., Ensign, D.L., Jayachandran, G., Khaliq, S., Pande, V.S.: Folding@home: Lessons from eight years of volunteer distributed computing. In: IEEE International Symposium on Parallel Distributed Processing (2009)

[47] Beierle, C., Jean, J., Kölbl, S., Leander, G., Moradi, A., Peyrin, T., Sasaki, Y., Sasdrich, P., Sim, S.M.: The skinny family of block ciphers and its low-latency variant mantis. Cryptology ePrint Archive, Report 2016/660 (2016), https://eprint.iacr.org/2016/660

[48] Bellizia, D., Berti, F., Bronchain, O., Cassiers, G., Duval, S., Guo, C., Leander, G., Leurent, G., Levi, I., Momin, C., Pereira, O., Peters, T., Standaert, F.X., Wiemer, F.: Spook: Sponge-Based Leakage-Resistant Authenticated Encryption with a Masked Tweakable Block Cipher p. 22

[49] Ben-Sasson, E., Chiesa, A., Genkin, D., Tromer, E., Virza, M.: Snarks for c: Verifying program executions succinctly and in zero knowledge. In: Canetti, R., Garay, J.A. (eds.) Advances in Cryptology – CRYPTO 2013. pp. 90–108. Springer Berlin Heidelberg (2013)

[50] Bennett, C.H., Brassard, G.: Quantum cryptography: Public key distribution and coin tossing. Theoretical Computer Science 560, 7–11 (Dec 2014). https://doi.org/10.1016/j.tcs.2014.05.025, http://dx.doi.org/10.1016/j.tcs.2014.05.025

[51] Berbain, C., Billet, O., Canteaut, A., Courtois, N., Gilbert, H., Goubin, L., Gouget, A., Granboulan, L., Lauradoux, C., Minier, M., Pornin, T., Sibert, H.: Sosemanuk, a Fast Software-Oriented Stream Cipher, pp. 98–118. Springer Berlin Heidelberg, Berlin, Heidelberg (2008)

[52] Berger, T.P., Minier, M.: Some results using the matrix methods on impossible, integral and zero-correlation distinguishers for feistel-like ciphers. In: Progress in Cryptology - INDOCRYPT 2015. Lecture Notes in Computer Science, vol. 9462, pp. 180–197. Springer (2015)

[53] Bernstein, D., Brumley, B., Chen, M.S., Chuengsatiansup, C., Lange, T., Marotzke, A., Peng, B.Y., Tuveri, N., van Vredendaal, C., Yang, B.Y.: NTRU prime: round 3 (2020), https://ntruprime.cr.yp.to/nist/ntruprime-20201007.pdf

[54] Bernstein, D.J., Kölbl, S., Lucks, S., Massolino, P.M.C., Mendel, F., Nawaz, K., Schneider, T., Schwabe, P., Standaert, F.X., Todo, Y., , Viguier, B.: Gimli: a cross-platform permutation (2017), https://cryptojedi.org/papers/gimli-20170627.pdf

[55] Bernstein, D.: The salsa20 family of stream ciphers 4986 (2008)

[56] Berti, F., Guo, C., Pereira, O., Peters, T., Standaert, F.X.: Tedt, a leakage-resist aead mode for high physical security applications. IACR Transactions on Cryptographic Hardware and Embedded Systems pp. 256–320 (11 2019). https://doi.org/10.46586/tches.v2020.i1.256-320

[57] Bertoni, G., Daemen, J., Peeters, M., Assche, G.: On the indifferentiability of the sponge construction. vol. 4965, pp. 181–197 (04 2008)

[58] Bertoni, G., Daemen, J., Peeters, M., Assche, G.V.: The keccak SHA-3 submission (2011), https://keccak.team/files/Keccak-submission-3.pdf

[59] Bertoni, G., Daemen, J., Peeters, M., Van Assche, G.: Duplexing the Sponge: Single-Pass Authenticated Encryption and Other Applications. In: Miri, A., Vaudenay, S. (eds.) Selected Areas in Cryptography. pp. 320–337. Springer Berlin Heidelberg, Berlin, Heidelberg (2012)

[60] Biham, E., Shamir, A.: Differential Cryptanalysis of the Data Encryption Standard. Springer-Verlag, Berlin, Heidelberg (1993)

[61] Biham, E., Shamir, A.: Differential fault analysis of secret key cryptosystems. In: Kaliski, B.S. (ed.) Advances in Cryptology — CRYPTO '97. pp. 513–525. Springer Berlin Heidelberg, Berlin, Heidelberg (1997)

[62] Bilgin, B., S.Nikova, V.Nikov, V.Rijmen, G.Stütz: Threshold implementations of all 3x3 and 4x4 s-boxes. Cryptology ePrint Archive, Report 2012/300 (2012), https://eprint.iacr.org/2012/300

[63] Bilgin, B., Gierlichs, B., Nikova, S., Nikov, V., Rijmen, V.: Higher-order threshold implementations. Cryptology ePrint Archive, Report 2014/751 (2014), https://eprint.iacr.org/2014/751

[64] Biryukov, A., Shamir, A.: Cryptanalytic time/memory/data tradeoffs for stream ciphers. In: Advances in Cryptology - ASIACRYPT 2000. Lecture Notes in Computer Science, vol. 1976, pp. 1–13. Springer (2000)

[65] Boesgaard, M., Vesterager, M., Zenner, E.: The rabbit stream cipher 4986 (2008)

[66] Bogdanov, A., Knudsen, L.R., Leander, G., Paar, C., Poschmann, A., Robshaw, M.J.B., Seurin, Y., Vikkelsoe, C.: PRESENT: An ultra-lightweight block cipher. In: Paillier, P., Verbauwhede, I. (eds.) Cryptographic Hardware and Embedded Systems - CHES 2007. pp. 450–466. Springer Berlin Heidelberg, Berlin, Heidelberg (2007)

[67] Bogdanov, A., Knezevic, M., Leander, G., Toz, D., Varici, K., Verbauwhede, I.: Spongent: The design space of lightweight cryptographic hashing. Cryptology ePrint Archive, Report 2011/697 (2011), https://eprint.iacr.org/2011/697

[68] Bois, A., Cascudo, I., Fiore, D., Kim, D.: Flexible and efficient verifiable computation on encrypted data. In: Garay, J.A. (ed.) Public-Key Cryptography - PKC 2021 - 24th IACR International Conference on Practice and Theory of Public Key Cryptography, Virtual Event, Proceedings, Part II. Lecture Notes in Computer Science, Springer (2021)

[69] Boneh, D., DeMillo, R.A., Lipton, R.J.: On the importance of checking cryptographic protocols for faults. In: Fumy, W. (ed.) Advances in Cryptology — EUROCRYPT '97. pp. 37–51. Springer Berlin Heidelberg, Berlin, Heidelberg (1997)

[70] Boneh, D., Eskandarian, S., Hanzlik, L., Greco, N.: Single secret leader election. IACR Cryptology ePrint Archive (2020)

[71] Borghoff, J., Canteaut, A., Güneysu, T., Kavun, E.B., Knezevic, M., Knudsen, L.R., Leander, G., Nikov, V., Paar, h., Rechberger, C., Rombouts, P., Thomsen, S.S., Yalçın, T.: Prince – a low-latency block cipher for pervasive computing applications. In: Wang, X., Sako, K. (eds.) Advances in Cryptology – ASIACRYPT 2012. pp. 208–225. Springer Berlin Heidelberg, Berlin, Heidelberg (2012)

[72] Bos, J., Ducas, L., Kiltz, E., Lepoint, T., Lyubashevsky, V., Schanck, J.M., Schwabe, P., Seiler, G., Stehlé, D.: Crystals – kyber: a cca-secure module-lattice-based kem. Cryptology ePrint Archive, Report 2017/634 (2017), https://eprint.iacr.org/2017/634

[73] Brier, E., Clavier, C., Olivier, F.: Correlation power analysis with a leakage model. In: Joye, M., Quisquater, J.J. (eds.) Cryptographic Hardware and Embedded Systems - CHES 2004. pp. 16–29. Springer Berlin Heidelberg, Berlin, Heidelberg (2004)

[74] Burwick, C., Coppersmith, D., D'Avignon, E., Gennaro, R., Halevi, S., Jutla, C., Jr., S.M.M., Peyravian, M., O'Connor, L., Safford, D., Zunic, N.: MARS – a candidate cipher for aes. p. 337–342 (1998)

[75] Cannière, C.D.: Analysis and Design of Symmetric Encryption Algorithms. Ph.D. thesis, Katholieke Universiteit Leuven (2007)

[76] Casamayou, A., Cohen, N., Connan, G., Dumont, T., Fousse, L., Maltey, F., Meulien, M., Mezzarobba, M., Pernet, C., Thiéry, N., et al.: Calcul mathématique avec Sage. available online: https://hal.inria.fr/inria-00540485v2/document (2013)

[77] Casanova, A., Faugere, J.C., Macario-Rat, G., Patarin, J., Perret, L., Ryckeghem, J.: GeMSS: A great multivariate short signature (2017), https://www-polsys.lip6.fr/Links/NIST/GeMSS_specification_round2.pdf

[78] Cascudo, I., David, B.: SCRAPE: scalable randomness attested by public entities. In: Applied Cryptography and Network Security - 15th International Conference, ACNS 2017, Kanazawa, Japan, Proceedings. Lecture Notes in Computer Science, Springer (2017)

[79] Cevallos, A., Stewart, A.: A verifiably secure and proportional committee election rule (2020)

[80] Chakraborti, A., Datta, N., Nandi, M., Yasuda, K.: Beetle family of lightweight and secure authenticated encryption ciphers. Cryptology ePrint Archive, Report 2018/805 (2018), https://eprint.iacr.org/2018/805

[81] Chakraborti, A., Iwata, T., Minematsu, K., Nandi, M.: Blockcipher-based authenticated encryption: How small can we go? Cryptology ePrint Archive, Report 2017/649 (2017), https://eprint.iacr.org/2017/649

[82] Chang, D., Nandi, M.: A short proof of the prp/prf switching lemma (2008), http://eprint.iacr.org/2008/078, pointchang@gmail.com 13929 received 20 Feb 2008

[83] Chari, S., Jutla, C.S., Rao, J.R., Rohatgi, P.: Towards sound approaches to counteract power-analysis attacks. In: Wiener, M. (ed.) Advances in Cryptology — CRYPTO' 99. pp. 398–412. Springer Berlin Heidelberg, Berlin, Heidelberg (1999)

[84] Chari, S., Rao, J.R., Rohatgi, P.: Template attacks. In: Kaliski, B.S., Koç, ç.K., Paar, C. (eds.) Cryptographic Hardware and Embedded Systems - CHES 2002. pp. 13–28. Springer Berlin Heidelberg, Berlin, Heidelberg (2003)

[85] Chase, M., Derler, D., Goldfeder, S., Orlandi, C., Ramacher, S., Rechberger, C., Slamanig, D., Zaverucha, G.: Post-quantum zero-knowledge and signatures from symmetric-key primitives. Cryptology ePrint Archive, Report 2017/279 (2017), https://eprint.iacr.org/2017/279

[86] Chatterjee, K., Goharshady, A.K., Pourdamghani, A.: Hybrid mining: Exploiting blockchain's computational power for distributed problem solving. In: Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing. SAC '19, Association for Computing Machinery (2019)

[87] Chen, C., Danba, O., Hoffstein, J., Hülsing, A., Rijneveld, J., Schanck, J.M., Schwabe, P., Whyte, W., Zhang, Z.: NTRU (2019), https://ntru.org/f/ntru-20190330.pdf

[88] Cogliati, B., Lampe, R., Seurin, Y.: Tweaking even-mansour ciphers. Cryptology ePrint Archive, Report 2015/539 (2015), https://eprint.iacr.org/2015/539

[89] Coron, J.S., Goubin, L.: On boolean and arithmetic masking against differential power analysis. In: Koç, Ç.K., Paar, C. (eds.) Cryptographic Hardware and Embedded Systems — CHES 2000. pp. 231–237. Springer Berlin Heidelberg, Berlin, Heidelberg (2000)

[90] Courtois, N., Klimov, A., Patarin, J., Shamir, A.: Efficient algorithms for solving overdefined systems of multivariate polynomial equations. In: Advances in Cryptology - EUROCRYPT 2000. Lecture Notes in Computer Science, vol. 1807, pp. 392–407. Springer (2000)

[91] Crowley, P.: Mercy: A fast large block cipher for disk sector encryption. pp. 49–63 (2000)

[92] Cui, T., Wang, M., Fan, Y., Hu, K., Fu, Y., Huang, L.: Ballet: A software-friendly block cipher 6(6), 704 (2019). https://doi.org/10.13868/j.cnki.jcr.000335, http://www.jcr.cacrnet.org.cn/EN/abstract/abstract359.shtml

[93] Daemen, J., Govaerts, R., Vandewalle, J.: On the design of high speed self-synchronizing stream ciphers. In: Proc. of the ICCS/ISITA'92 conference. vol. 1, pp. 279–283. Singapore (November 1992)

[94] Daemen, J., Kitsos, P.: The Self-Synchronizing Stream Cipher MOSQUITO: eSTREAM Documentation, Version 2. eSTREAM, ECRYPT Stream Cipher Project, Report 2005/018 (2005), available online at http://www.ecrypt.eu.org/stream/p3ciphers/mosquito/mosquito.pdf

[95] Daemen, J., Kitsos, P.: The self-synchronizing stream cipher moustique. In: New Stream Cipher Designs - The eSTREAM Finalists, Lecture Notes in Computer Science, vol. 4986, pp. 210–223. Springer (2008)

[96] Daemen, J.: Limitations of the Even-Mansour construction. In: Imai, H., Rivest, R.L., Matsumoto, T. (eds.) Advances in Cryptology — ASIACRYPT '91. pp. 495–498. Springer Berlin Heidelberg, Berlin, Heidelberg (1993)

[97] Daemen, J., Rijmen, V.: The design of Rijndael: AES — the Advanced Encryption Standard. Springer-Verlag (2002)

[98] Daian, P., Pass, R., Shi, E.: Snow white: Robustly reconfigurable consensus and applications to provably secure proof of stake. In: Goldberg, I., Moore, T. (eds.) Financial Cryptography and Data Security. Springer International Publishing (2019)

[99] David Jao, D., Azarderakhsh, R., Campagna, M., Costello, C., De Feo, L., Hess, B., Hutchinson, A., Jalali, A., Karabina, K., Koziel, B., LaMacchia, B., Longa, P., Naehrig, M., Pereira, G., Renes, J., Soukharev, V., Urbanik, D.: Supersingular isogeny key encapsulation (2020), https://sike.org/files/SIDH-spec.pdf

[100] De Cannière, C., Dunkelman, O., Knežević, M.: Katan and ktantan — a family of small and efficient hardware-oriented block ciphers. In: Clavier, C., Gaj, K. (eds.) Cryptographic Hardware and Embedded Systems - CHES 2009. pp. 272–288. Springer Berlin Heidelberg, Berlin, Heidelberg (2009)

[101] De Cannière, C., Preneel, B.: Trivium 4986 (2008)

[102] Diffie, W., Hellman, M.: New directions in cryptography. In: IEEE Transactions on Information Theory (1976)

[103] Ding, J., Chen, M.S., Kannwischer, M., Patarin, J., Petzoldt, A., Schmidt, D., Yang, B.Y.: Rainbow - algorithm specification and documentation (2020)

[104] Dinur, I., Shamir, A.: Cube attacks on tweakable black box polynomials. In: Advances in Cryptology - EUROCRYPT 2009. Lecture Notes in Computer Science, vol. 5479, pp. 278–299. Springer (2009)

[105] Dobraunig, C., Eichlseder, M., Mangard, S., Mendel, F., Mennink, B., Primas, R., Unterluggauer, T.: Isap v2.0. IACR Transactions on Symmetric Cryptology 2020(S1), 390–416 (Jun 2020). https://doi.org/10.13154/tosc.v2020.iS1.390-416, https://tosc.iacr.org/index.php/ToSC/article/view/8625

[106] Dobraunig, C., Eichlseder, M., Mendel, F., Schläffer, M.: Ascon p. 48 (2021)

[107] Douceur, J.R.: The sybil attack. In: Druschel, P., Kaashoek, F., Rowstron, A. (eds.) Peer-to-Peer Systems. Springer Berlin Heidelberg (2002)

[108] Dravie, B., Guillot, P., Millérioux, G.: Design of self-synchronizing stream ciphers: A new control-theoretical paradigm. In: IFAC World Congress, (IFAC 2017). Toulouse, France (July 2017)

[109] Dravie, B., Guillot, P., Millérioux, G.: Security proof of the canonical form of self-synchronizing stream ciphers. Des. Codes Cryptography 82(1-2), 377–388 (2017)

[110] Dunkelman, O., Keller, N., Shamir, A.: Minimalism in cryptography: The even-mansour scheme revisited. In: Pointcheval, D., Johansson, T. (eds.) Advances in Cryptology – EUROCRYPT 2012. pp. 336–354. Springer Berlin Heidelberg, Berlin, Heidelberg (2012)

[111] Dworkin, M.: Nist special publication 800-38a, recommendation for block cipher modes of operation-methods and techniques. National Institute of Standards and Technology/US Department of Commerce (2001), https://csrc.nist.gov/publications/detail/sp/800-38a/final

[112] Dworkin, M.: Recommendation for block cipher modes of operation: Galois/counter mode (gcm) and gmac. NIST Special Publication 800-38D (2007), https://nvlpubs.nist.gov/nistpubs/legacy/sp/nistspecialpublication800-38d.pdf

[113] Dworkin, M.: Recommendation for block cipher modes of operation: The ccm mode for authentication and confidentiality. NIST Special Publication 800-38C (2007), https://nvlpubs.nist.gov/nistpubs/legacy/sp/nistspecialpublication800-38c.pdf

[114] Dworkin, M.: Recommendation for block cipher modes of operation : The CMAC mode for authentication. NIST Special Publication 800-38B (2016), https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-38b.pdf

[115] Ehrsam, W.F., Meyer, C.H.W., Smith, J.L., Tuchman, W.L.: Message verification and transmission error detection by block chaining (1976), https://patents.google.com/patent/US4074066A/en

[116] Ekert, A.K.: Quantum cryptography based on bell's theorem. Phys. Rev. Lett. 67, 661–663 (Aug 1991). https://doi.org/10.1103/PhysRevLett.67.661, https://link.aps.org/doi/10.1103/PhysRevLett.67.661

[117] Even, S., Mansour, Y.: A construction of a cipher from a single pseudorandom permutation. In: Imai, H., Rivest, R.L., Matsumoto, T. (eds.) Advances in Cryptology — ASIACRYPT '91. pp. 210–224. Springer Berlin Heidelberg, Berlin, Heidelberg (1993)

[118] Eyal, I., Sirer, E.G.: Majority is not enough: Bitcoin mining is vulnerable. CoRR abs/1311.0243 (2013), http://arxiv.org/abs/1311.0243

[119] Farooq, U., Faisal Aslam, M.: Comparative analysis of different AES implementation techniques for efficient resource usage and better performance of an FPGA. Journal of King Saud University - Computer and Information Sciences (2016). https://doi.org/10.1016/j.jksuci.2016.01.004, https://hal.sorbonne-universite.fr/hal-01298000

[120] Ferguson, N.: Authentication weakness in GCM (2005)

[121] Ferguson, N., Lucks, S., Schneier, B., Whiting, D., Bellare, M., Kohno, T., Callas, J., Walker, J.: The skein hash function family (2010), https://www.schneier.com/wp-content/uploads/2015/01/skein.pdf

[122] Fouque, P.A., Karpman, P.: Security amplification against meet-in-the-middle attacks using whitening. Cryptology ePrint Archive, Report 2013/618 (2013), https://eprint.iacr.org/2013/618

[123] Francq, J., Besson, L., Huynh, P., Guillot, P., Millerioux, G., Minier, M.: Non-triangular self-synchronizing stream ciphers. IEEE Transactions on Computers pp. 1–1 (2020). https://doi.org/10.1109/TC.2020.3043714

[124] Friedman, J.: TEMPEST : A Signal Problem (1972), https://www.nsa.gov/Portals/70/documents/news-features/declassified-documents/cryptologic-spectrum/tempest.pdf

[125] Gauravaram, P., Knudsen, L.R., Matusiewicz, K., Mendel, F., Rechberger, C., Schläffer, M., Thomsen, S.S.: Grøstl– a SHa-3 candidate (2011), http://www.groestl.info/Groestl.pdf

[126] Gaži, P., Pietrzak, K., Tessaro, S.: The exact prf security of truncation: Tight bounds for keyed sponges and truncated cbc. In: Gennaro, R., Robshaw, M. (eds.) Advances in Cryptology – CRYPTO 2015. pp. 368–387. Springer Berlin Heidelberg, Berlin, Heidelberg (2015)

[127] Gennaro, R., Gentry, C., Parno, B.: Non-interactive verifiable computing: Outsourcing computation to untrusted workers. In: Rabin, T. (ed.) Advances in Cryptology – CRYPTO 2010. Springer Berlin Heidelberg (2010)

[128] Gilad, Y., Hemo, R., Micali, S., Vlachos, G., Zeldovich, N.: Algorand: Scaling byzantine agreements for cryptocurrencies. In: Proceedings of the 26th Symposium on Operating Systems Principles, Shanghai, China. ACM (2017)

[129] Gligor, V.D., Donescu, P.: Fast encryption and authentication: Xcbc encryption and xecb authentication modes. In: Matsui, M. (ed.) Fast Software Encryption. pp. 92–108. Springer Berlin Heidelberg, Berlin, Heidelberg (2002)

[130] Goldenberg, D., Hohenberger, S., Liskov, M., Schwartz, E.C., Seyalioglu, H.: On tweaking luby-rackoff blockciphers. In: ASIACRYPT. pp. 342–356 (2007)

[131] Golić, J.D., Tymen, C.: Multiplicative masking and power analysis of aes. In: Kaliski, B.S., Koç, ç.K., Paar, C. (eds.) Cryptographic Hardware and Embedded Systems - CHES 2002. pp. 198–212. Springer Berlin Heidelberg, Berlin, Heidelberg (2003)

[132] Goubin, L., Patarin, J.: Des and differential power analysis the "duplication" method. In: Koç, Ç.K., Paar, C. (eds.) Cryptographic Hardware and Embedded Systems. pp. 158–172. Springer Berlin Heidelberg, Berlin, Heidelberg (1999)

[133] Grothe, M., Niemann, T., Somorovsky, J., Schwenk, J.: Breaking and fixing gridcoin. In: Proceedings of the 11th USENIX Conference on Offensive Technologies. WOOT'17, USENIX Association (2017)

[134] Gueron, S., Langley, A., Lindell, Y.: Aes-gcm-siv: Specification and analysis. Cryptology ePrint Archive, Report 2017/168 (2017), https://ia.cr/2017/168

[135] Guo, C., Pereira, O., Peters, T., Standaert, F.X.: Towards low-energy leakage-resistant authenticated encryption from the duplex sponge construction. Cryptology ePrint Archive, Report 2019/193 (2019), https://eprint.iacr.org/2019/193

[136] Guo, J., Peyrin, T., Poschmann, A.: The PHOTON Family of Lightweight Hash Functions. In: Hutchison, D., Kanade, T., Kittler, J., Kleinberg, J.M., Mattern, F., Mitchell, J.C., Naor, M., Nierstrasz, O., Pandu Rangan, C., Steffen, B., Sudan, M., Terzopoulos, D., Tygar, D., Vardi, M.Y., Weikum, G., Rogaway, P. (eds.) Advances in Cryptology – CRYPTO 2011, vol. 6841, pp. 222–239. Springer Berlin Heidelberg, Berlin, Heidelberg (2011), series Title: Lecture Notes in Computer Science

[137] Hawkes, P., Paddon, M., Rose, G.G., Miriam, W.V.: Primitive specification for sss. Tech. rep., e-Stream Project (2004), available at: http://www.ecrypt.eu.org/stream/ciphers/sss/sss.pdf

[138] Hell, M., Johansson, T., Maximov, A., Meier, W.: The grain family of stream ciphers 4986 (2008)

[139] Hellman, M.E.: A cryptanalytic time-memory trade-off. IEEE Trans. Information Theory 26(4), 401–406 (1980)

[140] Hoffstein, P.A.F.J., Kirchner, P., Lyubashevsky, V., Pornin, T., Prest, T., Ricosset, T., Seiler, G., Whyte, W., Zhang, Z.: Falcon: Fast-fourier lattice-based compact signatures over NTRU (2020), https://falcon-sign.info/falcon.pdf

[141] Homsirikamol, E., Diehl, W., Ferozpuri, A., Farahmand, F., Yalla, P., Kaps, J.P., Gaj, K.: CAESAR Hardware API. Cryptology ePrint Archive, Report 2016/626 (2016), https://eprint.iacr.org/2016/626

[142] Hong, D., Sung, J., Hong, S., Lim, J., Lee, S., Koo, B.S., Lee, C., Chang, D., Lee, J., Jeong, K., Kim, H., Kim, J., Chee, S.: HIGHT: A New Block Cipher Suitable for Low-Resource Device. In: Goubin, L., Matsui, M. (eds.) Cryptographic Hardware and Embedded Systems - CHES 2006. pp. 46–59. Springer Berlin Heidelberg, Berlin, Heidelberg (2006)

[143] Huang, J., Lai, X.: What is the effective key length for a block cipher: an attack on every block cipher. Cryptology ePrint Archive, Report 2012/677 (2012), https://eprint.iacr.org/2012/677

[144] Institute of Systems, Information Technologies and Nanotechnologies (ISIT): PQCrypto 2016 : The Seventh International Conference on Post-Quantum Cryptography, https://pqcrypto2016.jp/

[145] Iwata, T., Yasuda, K.: Hbs: A single-key mode of operation for deterministic authenticated encryption. In: Dunkelman, O. (ed.) Fast Software Encryption. pp. 394–415. Springer Berlin Heidelberg, Berlin, Heidelberg (2009)

[146] Jean, J., Nikolic, I., Seurin, Y., Peyrin, T.: Deoxys v1.41 p. 38 (2016)

[147] Jean, J., Nikolić, I., Peyrin, T.: Tweaks and keys for block ciphers: the tweakey framework. Cryptology ePrint Archive, Report 2014/831 (2014), https://eprint.iacr.org/2014/831

[148] Jian Guo, Thomas Peyrin, A.P., Robshaw, M.: The led block cipher. Cryptology ePrint Archive, Report 2012/600 (2012), https://eprint.iacr.org/2012/600

[149] Johnson, D., Menezes, A., Vanstone, S.: The elliptic curve digital signature algorithm (ECDSA). In: International Journal of Information Security. pp. 36–63 (2001), https://link.springer.com/article/10.1007/s102070100002

[150] Joux, A., Muller, F.: Loosening the KNOT. In: Fast Software Encryption - FSE 2003. Lecture Notes in Computer Science, vol. 2887, pp. 87–99. Springer (2003)

[151] Joux, A., Muller, F.: Two attacks against the HBB stream cipher. In: Fast Software Encryption - FSE 2005. Lecture Notes in Computer Science, vol. 3557, pp. 330–341. Springer (2005)

[152] Joux, A., Muller, F.: Chosen-ciphertext attacks against MOSQUITO. In: Fast Software Encryption - FSE 2006. Lecture Notes in Computer Science, vol. 4047, pp. 390–404. Springer (2006)

[153] Jovanovic, P., Luykx, A., Mennink, B.: Beyond 2c/2 security in sponge-based authenticated encryption modes. In: Sarkar, P., Iwata, T. (eds.) Advances in Cryptology – ASIACRYPT 2014. pp. 85–104. Springer Berlin Heidelberg, Berlin, Heidelberg (2014)

[154] Juričić, V., Radošević, M., Fuzul, E.: Optimizing the resource consumption of blockchain technology in business systems. Business Systems Research Journal (2020)

[155] Kaps, J.P., Diehl, W., Tempelmeier, M., Farahmand, F., Homsirikamol, E., Gaj, K.: A comprehensive framework for fair and efficient benchmarking of hardware implementations of lightweight cryptography. Cryptology ePrint Archive, Report 2019/1273 (2019), https://eprint.iacr.org/2019/1273

[156] Kaps, J.P., Diehl, W., Tempelmeier, M., Homsirikamol, E., Gaj, K.: Hardware api for lightweight cryptography (2019), https://cryptography.gmu.edu/athena/LWC/LWC_HW_API.pdf

[157] Karl, P., Tempelmeier, M.: A detailed report on the overhead of hardware APIs for lightweight cryptography (2020), https://csrc.nist.gov/CSRC/media/Events/lightweight-cryptography-workshop-2020/documents/papers/detailed-report-overhead-hardware-lwc2020.pdf

[158] Käsper, E., Rijmen, V., Bjørstad, T.E., Rechberger, C., Robshaw, M.J.B., Sekar, G.: Correlated keystreams in moustique. In: Progress in Cryptology - AFRICACRYPT 2008. Lecture Notes in Computer Science, vol. 5023, pp. 246–257. Springer (2008)

[159] Kerckhoffs, A.: La cryptographie militaire (1883)

[160] Kiayias, A., Russell, A., David, B., Oliynykov, R.: Ouroboros: A provably secure proof-of-stake blockchain protocol. In: CRYPTO. Springer (2017)

[161] King, S.: Primecoin: Cryptocurrency with prime number proof-of-work. http://primecoin.io/bin/primecoin-paper.pdf (2013)

[162] Klíma, V.: Cryptanalysis of hiji-bij-bij (HBB). IACR Cryptology ePrint Archive, Report 2005/3 (2005)

[163] Knudsen, L.R.: DEAL - a 128-bit block cipher (1998)

[164] Kocher, P., Jaffe, J., Jun, B.: Differential power analysis. In: Wiener, M. (ed.) Advances in Cryptology — CRYPTO' 99. pp. 388–397. Springer Berlin Heidelberg, Berlin, Heidelberg (1999)

[165] Kocher, P.C.: Timing attacks on implementations of diffie-hellman, rsa, dss, and other systems. In: Koblitz, N. (ed.) Advances in Cryptology — CRYPTO '96. pp. 104–113. Springer Berlin Heidelberg, Berlin, Heidelberg (1996)

[166] Krovetz, T., Rogaway, P.: OCB (v1.1) p. 46

[167] Kuhn, M.: Compromising emanations: eavesdropping risks of computer displays (UCAM-CL-TR-577) (dec). https://doi.org/10.48456/tr-577, https://www.cl.cam.ac.uk/techreports/UCAM-CL-TR-577.pdf

[168] Lai, X.: On the design and security of block ciphers. Hartung-Gorre (1992), https://doi.org/10.3929/ethz-a-000646711

[169] Leander, G., Poschmann, A.: On the classification of 4 bit s-boxes. In: Arithmetic of Finite Fields, First International Workshop, WAIFI 2007. Lecture Notes in Computer Science, vol. 4547, pp. 159–176. Springer (2007)

[170] Lihu, A., Du, J., Barjaktarevic, I., Gerzanics, P., Harvilla, M.: A proof of useful work for artificial intelligence on the blockchain (2020)

[171] Liskov, M., Rivest, R.L., Wagner, D.: Tweakable block ciphers. In: Yung, M. (ed.) Advances in Cryptology — CRYPTO 2002. pp. 31–46. Springer Berlin Heidelberg, Berlin, Heidelberg (2002)

[172] Loughry, J., Umphress, D.: Information leakage from optical emanations. ACM Trans. Inf. Syst. Secur. 5, 262–289 (2002)

[173] Lu, X., Liu, Y., Zhang, Z., Jia, D., Xue, H., He, J., Li, B., Wang, K.: Lac: Practical ring-lwe based public-key encryption with byte-level modulus. Cryptology ePrint Archive, Report 2018/1009 (2018), https://eprint.iacr.org/2018/1009

[174] Luby, M., Rackoff, C.: How to construct pseudorandom permutations from pseudorandom functions (1987)

[175] Lucks, S., Callas, J.: The skein hash function family (2009)

[176] Mandal, K., Saha, D., Sarkar, S., Todo, Y.: Sycon: A new milestone in designing ascon-like permutations. Cryptology ePrint Archive, Report 2021/157 (2021), https://eprint.iacr.org/2021/157

[177] Matsui, M.: Linear cryptanalysis method for des cipher. In: Helleseth, T. (ed.) Advances in Cryptology — EUROCRYPT '93. pp. 386–397. Springer Berlin Heidelberg, Berlin, Heidelberg (1994)

[178] Matsui, M.: New block encryption algorithm misty. In: Biham, E. (ed.) Fast Software Encryption. pp. 54–68. Springer Berlin Heidelberg, Berlin, Heidelberg (1997)

[179] Maurer, U.M.: New approaches to the design of self-synchronizing stream ciphers. In: Advances in Cryptology - EUROCRYPT '91. Lecture Notes in Computer Science, vol. 547, pp. 458–471. Springer (1991)

[180] Maurer, U.M.: Indistinguishability of random systems. In: Advances in Cryptology - EUROCRYPT 2002. Lecture Notes in Computer Science, vol. 2332, pp. 110–132. Springer (2002)

[181] Melchor, C., Aragon, N., Bettaieb, S., Bidoux, L., Blazy, O., Deneuville, J.C., Gaborit, p., Persichetti, E., Zémor, G., Bos, J.: Hamming quasi-cyclic (HQC) (2020), http://pqc-hqc.org/doc/hqc-specification_2020-10-01.pdf

[182] Mennink, B.: Optimally secure tweakable blockciphers. Cryptology ePrint Archive, Report 2015/363 (2015), https://eprint.iacr.org/2015/363

[183] Millerioux, G., Boukhobza, T.: Characterization of flat outputs for LPV discrete-time systems: A graph-oriented approach. In: 54th IEEE Conference on Decision and Control, CDC 2015. pp. 759–764. IEEE (2015)

[184] Mitsuda, A., Iwata, T.: Tweakable pseudorandom permutation from generalized feistel structure. In: Baek, J., Bao, F., Chen, K., Lai, X. (eds.) Provable Security. pp. 22–37. Springer Berlin Heidelberg, Berlin, Heidelberg (2008)

[185] Mohajerani, K., Haeussler, R., Nagpal, R., Farahmand, F., Abdulgadir, A., Kaps, J.P., Gaj, K.: Fpga benchmarking of round 2 candidates in the nist lightweight cryptography standardization process: Methodology, metrics, tools, and results. Cryptology ePrint Archive, Report 2020/1207 (2020), https://eprint.iacr.org/2020/1207

[186] National Bureau of Standards: DATA ENCRYPTION STANDARD (DES) (1999), http://archive.wikiwix.com/cache/display2.php/fips46-3.pdf?url=http%3A%2F%2Fcsrc.nist.gov%2Fpublications%2Ffips%2Ffips46-3%2Ffips46-3.pdf

[187] Nawaz, Y., Gong, G.: The WG stream cipher (2005)

[188] Nikova, S., Rechberger, C., Rijmen, V.: Threshold implementations against side-channel attacks and glitches. In: Ning, P., Qing, S., Li, N. (eds.) Information and Communications Security. pp. 529–545. Springer Berlin Heidelberg, Berlin, Heidelberg (2006)

[189] Nyberg, K.: Generalized feistel networks. In: Kim, K., Matsumoto, T. (eds.) Advances in Cryptology — ASIACRYPT '96. pp. 91–104. Springer Berlin Heidelberg, Berlin, Heidelberg (1996)

[190] P. Dusart, G.L., Vivolo, O.: Differential fault analysis on a.e.s. Cryptology ePrint Archive, Report 2003/010 (2003), https://eprint.iacr.org/2003/010

[191] Pass, R., Shi, E.: The sleepy model of consensus. In: Takagi, T., Peyrin, T. (eds.) Advances in Cryptology – ASIACRYPT 2017. Springer International Publishing (2017)

[192] Rezvani, B., Coleman, F., Sachin, S., Diehl, W.: Hardware implementation of nist lightweight cryptographic candidates : A first look. Cryptology ePrint Archive, Report 2019/824 (2019), https://eprint.iacr.org/2019/824

[193] Rijmen, V., Daemen, J., Preneel, B., Bosselaers, A., De Win, E.: The cipher shark. In: Gollmann, D. (ed.) Fast Software Encryption. pp. 99–111. Springer Berlin Heidelberg, Berlin, Heidelberg (1996)

[194] Riou, S.: Drygascon p. 107

[195] Rivest, R., Shamir, A., Adleman, L.: A method for obtaining digital signatures and public- key cryptosystems pp. 120–126 (1978), http://people.csail.mit.edu/rivest/pubs/RSA78.pdf

[196] Rivest, R.L., Robshaw, M.J.B., Yin, Y.L.: RC6 as the AES. pp. 337–342 (2000)

[197] Rivest, R.L., Schuldt, J.C.N.: Spritz—a spongy rc4-like stream cipher and hash function. Cryptology ePrint Archive, Report 2016/856 (2016), https://eprint.iacr.org/2016/856

[198] Rogaway, P.: Efficient instantiations of tweakable blockciphers and refinements to modes ocb and pmac. In: Lee, P.J. (ed.) Advances in Cryptology - ASIACRYPT 2004. pp. 16–31. Springer Berlin Heidelberg, Berlin, Heidelberg (2004)

[199] Rogaway, P.: The SIV mode of operation for deterministic authenticated-encryption (key wrap) and misuse-resistant nonce-based authenticated-encryption (2007), https://web.cs.ucdavis.edu/~rogaway/papers/siv.pdf

[200] Rogaway, P.: Evaluation of some blockcipher modes of operation (2011), https://www.cs.ucdavis.edu/~rogaway/papers/modes.pdf

[201] Rogaway, P., Bellare, M., Black, J., Krovetz, T.: OCB Mode. Cryptology ePrint Archive, Report 2001/026 (2001), https://eprint.iacr.org/2001/026

[202] Rogaway, P., Shrimpton, T.: Deterministic authenticated-encryption: A provable-security treatment of the key-wrap problem. Cryptology ePrint Archive, Report 2006/221 (2006), https://ia.cr/2006/221

[203] Rueppel, R.A.: Analysis and design of stream ciphers (2012)

[204] Saarinen, M.J.O.: Cryptographic Analysis of All 4 x 4 - Bit S-Boxes. IACR Cryptology ePrint Archive, Report 2011/218 (2005)

[205] Saarinen, M.J.O.: Cycling attacks on gcm, ghash and other polynomial macs and hashes. In: Canteaut, A. (ed.) Fast Software Encryption. pp. 216–225. Springer Berlin Heidelberg, Berlin, Heidelberg (2012)

[206] Sakram, B., Malathi, N., Subba Rao, D.: Welch–gong (wg) 128 bitstream cipher for encryption anddecryption algorithm. International Journal of Emerging Engineering Research and Technology (2015), http://www.ijeert.org/pdf/v3-i8/19.pdf

[207] Sarkar, P.: Hiji-Bij-Bij: A New Stream Cipher with a Self-Synchronizing Mode of Operation (2003)

[208] Sasaki, Y., Todo, Y., Aoki, K., Naito, Y., Sugawara, T., Murakami, Y., Matsui, M., Hirose, S.: Minalpher v1.1 (2015), https://competitions.cr.yp.to/round2/minalpherv11.pdf

[209] Sato, H., Mimura, M., Tanaka, H.: Analysis of division property using milp method for lightweight blockcipher piccolo. In: 2019 14th Asia Joint Conference on Information Security (AsiaJCIS). pp. 48–55 (2019)

[210] Schindler, P., Judmayer, A., Hittmeir, M., Stifter, N., Weippl, E.R.: Randrunner: Distributed randomness from trapdoor vdfs with strong uniqueness. In: 28th Annual Network and Distributed System Security Symposium, NDSS. The Internet Society (2021)

[211] Schindler, P., Judmayer, A., Stifter, N., Weippl, E.R.: Hydrand: Efficient continuous distributed randomness. In: 2020 IEEE Symposium on Security and Privacy, SP. IEEE (2020)

[212] Schneier, B.: Description of a new variable-length key, 64-bit block cipher (blowfish). pp. 191–204. Springer-Verlag (1994)

[213] Schramm, K., Leander, G., Felke, P., Paar, C.: A collision-attack on aes. In: Joye, M., Quisquater, J.J. (eds.) Cryptographic Hardware and Embedded Systems - CHES 2004. pp. 163–175. Springer Berlin Heidelberg, Berlin, Heidelberg (2004)

[214] Schroepel, R.: Hasty pudding cipher (1998)

[215] Shannon, C.E.: Communication theory of secrecy systems. The Bell System Technical Journal 28(4), 656–715 (1949). https://doi.org/10.1002/j.1538-7305.1949.tb00928.x

[216] Shannon, C.: A mathematical theory of cryptography (1945), https://www.iacr.org/museum/shannon/shannon45.pdf

[217] Shibutani, K., Isobe, T., Hiwatari, H., Mitsuda, A., Akishita, T., Shirai, T.: Piccolo: An ultra-lightweight blockcipher. In: Preneel, B., Takagi, T. (eds.) Cryptographic Hardware and Embedded Systems – CHES 2011. pp. 342–357. Springer Berlin Heidelberg, Berlin, Heidelberg (2011)

[218] Shimizu, A., Miyaguchi, S.: Fast data encipherment algorithm feal. In: Chaum, D., Price, W.L. (eds.) Advances in Cryptology — EUROCRYPT' 87. pp. 267–278. Springer Berlin Heidelberg, Berlin, Heidelberg (1988)

[219] Shirai, T., Shibutani, K., Akishita, T., Moriai, S., Iwata, T.: The 128-bit blockcipher clefia (extended abstract). In: Biryukov, A. (ed.) Fast Software Encryption. pp. 181–195. Springer Berlin Heidelberg, Berlin, Heidelberg (2007)

[220] Shor, P.W.: Scheme for reducing decoherence in quantum computer memory. Phys. Rev. A 52, R2493–R2496 (Oct 1995), https://link.aps.org/doi/10.1103/PhysRevA.52.R2493

[221] Standaert, F.X.: Analysing the leakage-resistance of some round-2 candidates of the nist's lightweight crypto standardization process. NIST Lightweight Crypto Workshop 2019 (2019), https://csrc.nist.gov/CSRC/media/Events/lightweight-cryptography-workshop-2019/documents/papers/analyzing-the-leakage-resistance-of-round-2-candidates-lwc2019.pdf

[222] Steane, A.: Multiple-Particle Interference and Quantum Error Correction. Proceedings of the Royal Society of London Series A 452(1954), 2551–2577 (Nov 1996). https://doi.org/10.1098/rspa.1996.0136

[223] Syta, E., Jovanovic, P., Kokoris-Kogias, E., Gailly, N., Gasser, L., Khoffi, I., Fischer, M.J., Ford, B.: Scalable bias-resistant distributed randomness. In: 2017 IEEE Symposium on Security and Privacy, SP. IEEE Computer Society (2017)

[224] Turan, M.S., McKay, K.A., Çalık, a., Chang, D., Bassham, L.: Status report on the first round of the NIST lightweight cryptography standardization process. Tech. Rep. NIST IR 8268, National Institute of Standards and Technology, Gaithersburg, MD (Oct 2019). https://doi.org/10.6028/NIST.IR.8268, https://nvlpubs.nist.gov/nistpubs/ir/2019/NIST.IR.8268.pdf

[225] van Eck, W.: Electromagnetic radiation from video display units: An eavesdropping risk? Computers & Security 4(4), 269–286 (1985). https://doi.org/https://doi.org/10.1016/0167-4048(85)90046-X, https://www.sciencedirect.com/science/article/pii/016740488590046X

[226] Vaudenay, S.: On the lai-massey scheme. In: Lam, K.Y., Okamoto, E., Xing, C. (eds.) Advances in Cryptology - ASIACRYPT'99. pp. 8–19. Springer Berlin Heidelberg, Berlin, Heidelberg (1999)

[227] Wang, L., Guo, J., Zhang, G., Zhao, J., Gu, D.: How to build fully secure tweakable blockciphers from classical blockciphers. Cryptology ePrint Archive, Report 2016/876 (2016), https://eprint.iacr.org/2016/876

[228] Wang, X., Yu, H., Yin, Y.: Efficient collision search attacks on sha-0. In: CRYPTO (2005)

[229] Webster, A.F., Tavares, S.E.: On the design of s-boxes. In: Williams, H.C. (ed.) Advances in Cryptology — CRYPTO '85 Proceedings. pp. 523–534. Springer Berlin Heidelberg, Berlin, Heidelberg (1986)

[230] Wiesner, S.: Conjugate coding. SIGACT News 15(1), 78–88 (Jan 1983). https://doi.org/10.1145/1008908.1008920, https://doi.org/10.1145/1008908.1008920

[231] von Willich, M.: A technique with an information-theoretic basis for protecting secret data from differential power attacks. In: Honary, B. (ed.) Cryptography and Coding. pp. 44–62. Springer Berlin Heidelberg, Berlin, Heidelberg (2001)

[232] Wu, H.: The stream cipher HC-128 4986 (2008)

[233] Wu, H.: The hash function jh (2011), https://www3.ntu.edu.sg/home/wuhj/research/jh/jh_round3.pdf

[234] Wu, H.: ACORN: A Lightweight Authenticated Cipher (v3) p. 72 (2015)

[235] Wu, H., Preneel, B.: AEGIS: A Fast Authenticated Encryption Algorithm. Cryptology ePrint Archive, Report 2013/695 (2013), https://eprint.iacr.org/2013/695

[236] WU, W., ZHANG, L., ZHENG, Y., LI, L.: The block cipher ublock 6(6), 690 (2019). https://doi.org/10.13868/j.cnki.jcr.000334, http://www.jcr.cacrnet.org.cn/EN/abstract/abstract358.shtml

[237] Yang, G., Zhu, B., Suder, V., Aagaard, M.D., Gong, G.: The simeck family of lightweight block ciphers. Cryptology ePrint Archive, Report 2015/612 (2015), https://eprint.iacr.org/2015/612

[238] Zhang, F., Eyal, I., Escriva, R., Juels, A., van Renesse, R.: REM: resource-efficient mining for blockchains. In: 26th USENIX Security Symposium, USENIX Security 2017, Vancouver, BC, Canada, August 16-18, 2017

[239] Zhang, J., Yu, Y., Fan, S., Zhang, Z., Yang, K.: Tweaking the asymmetry of asymmetric-key cryptography on lattices: Kems and signatures of smaller sizes. Cryptology ePrint Archive, Report 2019/510 (2019), https://eprint.iacr.org/2019/510

Appendix A

Useful work: a new protocol to ensure usefulness of PoW-based consensus for blockchain

Context This work was made by Louis Goubin, Jacques Patarin, Ambre Toulemonde and myself, in parallel to my main thesis topic. It tackles the will to create a protocol for blockchain keeping as goal to reward useful works.

Abstract The blockchain is a new technology that attracts several actors since 2009, and in particular in the financial domain with the emergence of cryptocurrencies such as the well-known Bitcoin.
In a blockchain, seen also as a distributed ledger or a chain of blocks, the participants use a consensus protocol to add new data into the ledger. For example, in the Bitcoin Proof-of-Work consensus protocol, the participants have to unnecessarily invest a huge amount of energy to add a new block of transactions, and therefore to also win the coin associated to this block.
In this paper, we present a new protocol called Useful Work (UW) protocol that is based on the Proof-of-Stake and Proof-of-Work mechanisms where the computing work is dedicated to useful problems. The participants get a chance to win coins after performing honest and useful work for a submitted problem.
We present a high-level description of our UW protocol that is configurable and propose some variants of the protocol.
We discuss also some new and well-known issues that our protocol prevents.

A.1 Introduction

For years, the blockchain technology has aroused a great interest in several applications such that financial transactions. A blockchain is a distributed ledger or chain of blocks intended to be immutable, i.e. new data or transactions can only be appended into the ledger. To this end, participants use a consensus protocol enabling to choose one of them as leader who wins the right to write the next block into the blockchain. The choice of the consensus protocol is one of the main challenges in the blockchain technology.

The first consensus protocol for blockchain is the Proof-of-Work (PoW) used for transactions of the Bitcoin cryptocurrency. In PoW protocol, the leader is the first participant who solves the hash puzzle, i.e. finding a block's hash lower than a target value. This process is also called mining and the participants are called miners. Thus, the miners are selected as leaders proportionally to their computing power since the chance of finding a valid hash is proportional to the computing power owned. The PoW consensus protocol has been designed to achieve new requirements of scalability and incentivation. The scalability enables to handle a variable number of participants since in the Bitcoin blockchain, anyone at any time can participate to the consensus protocol. The incentivization aims to motivate the participation in the consensus protocol, e.g. the leaders are rewarded with Bitcoin cryptocurrencies.

However, the Bitcoin PoW consensus protocol has some limits. For example, there is a huge waste

of computing power since the mining can be performed only by brute force and so requires an important number of resources (computers, electricity, etc.). The current consumption of energy used to mine a new block and so new Bitcoin cryptocurrencies is equivalent to the electric consumption of a country with some tens of millions of inhabitants. Several alternative solutions have been proposed to replace the Bitcoin PoW consensus protocol or to make the work performed during the mining useful, while taking into account the new scalability and incentivization requirements.

A.1.1 Related works

The first solutions to prevent the PoW issues have been constructed on a Proof-of-Stake (PoS) mechanism, where a participant's probability of being leader is proportional to the amount of money he has invested in the system. The PoS mechanism may be combined with a random leader election like in Algorand [128] and Ouroboros [160].

Instead of replacing the PoW mechanism, other solutions [154] aim to make the computations useful by substituting the hash puzzle by another useful problem. For example, in Primecoin [161], the leader is the first participant who find a chain of prime numbers. Other mechanisms [37, 170] have been proposed to perform honest machine learning training work in order to have the chance to be the leader. The Folding@home [46] and SETI@home [23] projects aim to find solutions to real world problems such that researching various diseases and extra-terrestrial life respectively.

Zhang et al. [238] proposed the Resource-Efficient Mining (REM) which is a consensus protocol using Trusted Execution Environment (TEE). The TEE is a protected address space, called also an enclave, using Intel Software Guard Extension (SGX) which is a set of new instructions available on certain Intel CPUs. In REM, one or several clients propose any useful works in form of tasks that REM miners run in their SGX enclave. The verification and measurement of the work performed in the enclave are done through the Intel attestation service. The work of miners is metered on a per-instruction basis. The SGX enclave randomly determines whether the work results in a block by treating each instruction as a Bernoulli trial. Thus, in REM protocol, each executed useful work instruction is analogous to one hash query in the Bitcoin PoW consensus protocol. The miner is elected as leader proportionally to the number of executed useful-work instructions, and the SGX enclave works as a trusted random oracle. However, REM is a partially decentralized solution since it relies on Intel services.

Ball et al. proposed their Proof-of-Useful-Work (PoUW) [38] protocol to replace the Bitcoin PoW puzzle by resolving the k-Orthogonal Vectors (k-OV) problem. The first block contains the initial k-OV problem. All participant in the network are solving the same k-OV problem and the first one who has a better solution than the solution in the previous block becomes the leader. However, the work performed for PoUW protocol provides only new insights into the hardness problem and does not solve any real-world problem.

Hybrid solutions have also been proposed to decrease the energy waste of mining process by combining the hash puzzle with useful works. In the Hybrid Mining protocol of [86], anyone can submit any NP-complete problem along with a reward by translating it into the boolean satisfiability (SAT) problem. Then, miners solve either the submitted problem or the hash puzzle to have the chance of being leader. The consensus protocol of Gridcoin [133] is also an hybrid solution that combines PoW puzzle with useful computational work performed for BOINC projects. In Gridcoin, miners can perform the work for a BOINC project in order to make the PoW puzzle easier to solve. Indeed, in exchange of his solution to any BOINC project, the miner obtains BOINC credit that enables to increase the mining target, and so decreases the difficulty of the hash puzzle. Thus, the leader is the first miner who provides a block with a valid solution of PoW puzzle.

A.1.2 Our contribution

In this paper, we propose the Useful Work (UW) protocol which is a variant of the PoW mechanism where the work is used to solve any real world problem. Our UW protocol is designed to be used in permissionless setting, i.e. anyone at any time can participate in the protocol, and it is based on PoW and PoS mechanisms. We assume that while more than two thirds of coins are owned by honest participants, then our UW protocol works.

Broadly speaking, our UW protocol proceeds as follows. It is divided in rounds. At each round k, the participants compete to win useful coins. To this end, they have to perform the computational work of problems submitted by clients and provide a proof of their correct computations. When the computational works are validated by a group of verifiers, a random election is run to choose one of the participants to win useful coins. Then, a participant is elected to generate the block B_k of valid transactions that contains also the data related to the winner of useful coins.

First, we present a high-level description of our UW protocol with eight steps that may be flexible. We present a first method in each step and propose some possible variants. Then, we describe some issues and well-known attacks against consensus protocols and blockchains, and discuss the resilience of our UW protocol under these attacks.

A.2 Background

In this section, we recall the classical security properties of consensus protocols. Then, we mention the main vulnerabilities and issues of the Bitcoin PoW protocol that are discussed in more detail in Section A.5.

A.2.1 Security properties

A consensus protocol enables a set of n participants to agree on the current state. Generally, they select in distributed manner one of them as leader to provide the next data to add into the ledger. Safety and liveness are the basic security properties of consensus protocols. The safety property means that (i) only a value that has been proposed may be chosen; (ii) only a single value is chosen, and (iii) a participant never considers that a value has been chosen unless it actually has been. The liveness property requires that a consensus can be achieved even if some fraction of participants may be malicious or inactive. New consensus protocols designed for blockchain aim to achieve these security properties while avoiding the Bitcoin PoW issues.

A.2.2 Vulnerabilities and issues

In addition to the waste energy problem, the Bitcoin PoW consensus protocol may lead to a sort of centralization with participants grouping in pools to share the workload since it requires intensive computation to solve the hash puzzle. Being in the pool, the participant takes the advantage to smooth the incomes. Moreover, the Bitcoin PoW protocol may also suffer from fork problems leading to two valid blocks that extend the same block and compete to be the main chain. In PoW protocol, fork is solved with the longest chain rule approved by the majority of participants. The Bitcoin PoW protocol is also vulnerable to selfish mining strategies [118] due to the possibility to fork. Indeed, a rational adversary can leverage the usual forking in the protocol to temporarily hide one or several blocks in order to increase incomes by revealing her blocks at a suitable time. PoW protocol is also vulnerable to the Majority attack where an adversary owning more than half of computing power may add any block of transactions.

A.3 Entities and building blocks

At each round k, our UW protocol enables to generate and add a block B_k into the blockchain. Our UW protocol is designed to be used in permissionless setting, i.e. anyone at any time can participate in the protocol.

Participants called workers perform the work submitted by clients and provide proofs of their correct computations. Then, when the computations are validated by a set of verifiers, one of workers is elected as winner to win some useful coins. Finally, a writer is elected to generate the next block that contains valid transactions and data related to the winner.

We assume that we have a random number generator that provides trusted and verifiable random values. Several works study how to generate such random values in distributed manner [78, 223, 211, 210] which is out of scope of this paper.

A.3.1 Two type of entities

There are two types of entities in our Useful Work protocol: one or more clients and several participants.

Clients

The clients are external entities who need computing power for any real world problem, e.g. scientific experiments, mathematical problems, etc. The clients submit problems in the form of a code to run. Each client $\mathcal{C}_i$ owns an account $\texttt{acc}_{\mathcal{C}_i}$ that contains an amount of useful coins related to a pair of public and secret keys $(pk_{\mathcal{C}_i}, sk_{\mathcal{C}_i})$.

Participants

The participants perform the computational work of submitted problems and maintain the blockchain. They may have one or several roles: worker, voter, verifier and writer. For example, a worker may be also selected as a verifier or as a writer.

- Workers perform computation works of submitted problems to win useful coins. To this end, each worker has to run the code of a submitted problem and provides a result along with a proof of correct computation.
- Voters are selected to define which are the most relevant problems to run.
- Verifiers are selected to verify the result published and/or the generated block. The verifiers may earn also useful coins for their verification.
- A writer is elected to generate a block of valid transactions and add it into the blockchain. The writer is also rewarded for his work.

Each participant $\mathcal{M}_i$ owns a wallet $\texttt{acc}_{\mathcal{M}_i}$ that contains an amount of useful coins related to a pair of public and secret keys $(pk_{\mathcal{M}_i}, sk_{\mathcal{M}_i})$.

A.3.2 Proof of correct computation

For each executed code, a worker has to prove the correct execution of the problem's code. This ensures that the workers have invested computing power and time into the blockchain, and they may be rewarded for their work.

A first method may be to provide a hash of the computation executed. Then, the set of verifiers execute the corresponding code and compute the hash of their computations. If a threshold of verifiers obtains a hash equal to the hash provided by the worker, then the work is validated.

Proofs as proposed in [49] may also be used to prove the correct computations. Indeed, the authors of [49] present a Succinct Non-interactive ARgument of Knowledge (SNARK) to prove the correct execution of C programs. Given a program Φ in C and a time bound t to execute Φ, on any input x, it allows to prove the correct execution of Φ after a one-time setup requiring $O(|\Phi| \cdot t)$ cryptographic operations. The prover requires $O(|\Phi| \cdot t)$ cryptographic operations to generate the SNARK proof and the verifier performs $O(|x|)$ cryptographic operations to verify the proof.

Other solutions such that [127, 68] may be also interesting solutions that can be adapted without the privacy property. In our UW protocol, the work performed has to be useful and honest, and so the privacy of problem's data may not be necessary.

We denote by `PROVE`, the algorithm run by the workers to prove the correctness of their computations at the Useful work process and result announcement step as described in Section A.4. `PROVE` may take as input the problem's code Φ, the time t, the input x and the necessary keys and data according the chosen protocol, and output a result *out* and a proof of correct computation $proof_c$.

A.3.3 Group selection

Our UW protocol uses a mechanism of group selection to constitute three groups: (1) a jury of voters to select the relevant problems, (2) a verifier set to verify the performed work and (3) a verifier set to check the generated block. A participant may be selected proportionally to the amount of useful coins

owned and his contribution to the blockchain, e.g the number of generated blocks in the chain. All voters and verifiers, have the same weight for the vote and verification, respectively, in their group.

A group selection as in [79] may be a first solution. The authors of [79] propose a group selection where the selected group represents the participants proportionally to their stake and with no minority being neither underrepresented nor overrepresented. The general idea is to begin with an empty group and add a new member over a number of iterations, following some specific rules for the candidate selection.

The random secret election of Algorand may also be an efficient group selection. Indeed, the Algorand selection uses a random secret selection based on a Verifiable Random Function (VRF) that outputs a unique random value α with a proof π that α has been correctly computed. In the random secret selection, each participant privately learns if he is selected and later reveals his selection along with a proof of his selection.

We define by `SELECTION`, the algorithm executed by the participants at each step of a round to select (or learn his owned selection in) the group of voters and verifiers. It is run at the Relevant problems selection, Work verification and Writer election and validation steps as described in Section A.4. `SELECTION` may take as input the index of the round k, a trusted random value $r_{1,k}$, the role of the current step, i.e. voters, verifiers of the performed work or verifiers for the generated block, and other necessary data for the chosen protocol. It outputs the selection proof of verifiers.

A.3.4 Winner and writer election

In our UW protocol, we use an election mechanism to elect (1) a winner who wins the useful coins for his performed work and (2) a writer who generates the next block to be added into the blockchain. The winner and the writer may be elected proportionally to the amount of useful coin owned. The writer is elected among the workers who provide a valid work for a submitted problem. The winner and the writer may be or not the same participant. Indeed, the writer may be selected among the workers. In addition to provide valid work, the participant may provide also the next block to be added into the blockchain. Note that, in this case, an alternative mechanism to select a writer is required when there is no problem available. The writer may be also selected among all participants who want to be writer.

We can use the random election protocol proposed in [70]. Boneh et al. [70] describe an election where participants aim to randomly choose exactly one leader such that the identity of the leader will only be known by the chosen leader. Later, the elected leader can reveal her identity while proving that she indeed won the election.

We denote by `ELECTION`, the algorithm that elects one winner or writer among a set of participants. It is run by the participants at the Winner election and Writer election and validation steps described in Section A.4. `ELECTION` takes as inputs the index of the round k, the trusted random value $r_{2,k}$ and other necessary data according to the chosen protocol, and outputs the winner (or the writer) with an eligibility proof.

A.3.5 Verification process

In our UW protocol, the sets of verifiers are selected to verify the computation works and the generated blocks. These processes prevent the workers to provide false computations and the writer to add conflicting data into the blockchain, e.g. double spending transactions. A solution to validate the works and blocks may be to collect a threshold of acknowledgements th from the verifiers. The two thirds classical threshold may guarantee security requirements, such as to be resilient to the Majority attack (as described in Section A.5). Thus, when a participant receives at least $th = \frac{2n_v}{3} + 1$ acknowledgements that validate a computation work (or a block), with n_v the group size, then he can consider the work (or the block) as valid (or as a part of the blockchain).

A.3.6 Reward distribution

When the block is considered as a part of the blockchain, the participants involved in this block are rewarded. A fixed reward RW is distributed among the winner, verifiers and writer. The winner earns rw_{worker}, each verifier who has verified the work of the winner earns $rw_{verifier}$ and the writer earns

rw_{writer}. Thus, $RW = rw_{worker} + n_v \cdot rw_{verifier} + rw_{writer}$. The part of the worker rw_{worker} may be proportional to the work performed. Each verifier may earn the same amount $rw_{verifier}$.

A mechanism to punish malicious participation may also be planned. For example, if a threshold of verifiers invalidates the result of a worker, this latter may lose the coins committed for his participation.

A.4 Our Useful work protocol

Our Useful Work (UW) protocol is divided in rounds where at each round k, one block B_k is added into the blockchain. A round is divided in eight steps: 1) Problem proposal, 2) Resources provision, 3) Relevant problems selection, 4) Problems distribution, 5) Useful work process and result announcement, 6) Work verification, 7) Winner election and 8) Writer election.

A.4.1 Problem proposal

In our UW protocol, a client $\mathcal{C}_i$ submits his problem $PB_{\mathcal{C}_i}$ with the following information and broadcasts it to the participants:

- Φ the code of the problem
- α_c the expected number of computations to run Φ
- α_m the necessary memory to run Φ
- t the necessary time to run Φ
- f_1 the constant proposal problem fees
- $f_2 \cdot |\Phi|$ the problem storage fees
- (optional) *cert* a certification provided by an external entity
- $sign_{\mathcal{C}_i}$ the signature of the client $\mathcal{C}_i$ who submits the problem

A client $\mathcal{C}_i$ submits his problem under the form of a code Φ, e.g. in C, to be run. The client estimates α_c the number of computations, α_m the memory necessary and t the time to run Φ. The constant proposal problem fees f_1 is paid by the client who submits his problem and prevent any client to flood the network of problem. The problem storage fees $f_2 \cdot |\Phi|$ is also paid by the client to prevent computational waste in code and limit the code to only useful computation steps. Optionally, the client may provide also a certification *cert* generated by an external entity, e.g. laboratory or a university, to legitimate the origin of the problem. The client signs his problem proposal $sign_{\mathcal{C}_i}$ to prove that he has submitted the problem.

A.4.2 Resources provision

Each participant who wants to be a worker, i.e. running the code of a problem in order to win useful coins, has to make available his computing capacity. The worker $\mathcal{M}_i$ may commit a stake which is an amount of coins $\mathtt{acc}_{\mathcal{M}_i}$ and the amount of memory β_m available for the computations. A worker who proposes a certain quantity of memory will obtain a problem proposal requiring an equivalent quantity of memory.

Note that, a minimum amount of committed coins may be required to prevent an attacker to create several identities with low stake participation.

A.4.3 Relevant problems selection

Each problem proposal may be subject to a vote from a jury of voters to select the relevant problems. Indeed, selecting some problems may prevent clients to submit problems which may have a malicious impact for the society, e.g. breaking a secret key, or to propose an already submitted problem with a slight modification in order to earn easily useful coins.

The initial voters may be the creators of the blockchain. Each initial voter may have the same weight of vote, i.e. one voter is equivalent to one vote. Next, for each new block, a jury of voters is selected proportionally to his stake, i.e. a participant with a large stake has more chance to be selected in the jury of voters. The participants use the `SELECTION` algorithm to know who is selected as voter for the round k.

The role of the voters is to select the most relevant problems. The vote for a problem may be based on the number of computations, the quantity of memories and the time necessary to run the problem's code. Moreover, additional information such as a certificate *cert* signed by an external entity may also legitimate the problem. A voter $\mathcal{M}_i$ broadcasts his vote for a problem $PB_{\mathcal{C}_j}$ along with his proof of selection as voter outputted by `SELECTION`.

At the end of this step, the l problems $PB_{\mathcal{C}_1}, PB_{\mathcal{C}_2}, \ldots, PB_{\mathcal{C}_l}$ that have received the most votes are selected for the next step.

A.4.4 Problems distribution

The relevant problems $PB_{\mathcal{C}_1}, PB_{\mathcal{C}_2}, \ldots, PB_{\mathcal{C}_l}$ may be randomly distributed to the workers $\mathcal{M}_1, \ldots, \mathcal{M}_n$ who have committed their computing capacities at Resources provision step. A worker who has proposed a large quantity of memory will obtain a problem proposal requiring an equivalent quantity of memory.

The jury of voters agree on which worker has which problem. For each problem $PB_{\mathcal{C}_j}$, the voters randomly assign $PB_{\mathcal{C}_j}$ to the worker $\mathcal{M}_i$ such that $\beta_m \geq \alpha_m$ where β_m is the quantity of memory committed by the worker $\mathcal{M}_i$ and α_m the memory estimated by the client $\mathcal{C}_j$. An assignment proof $assign_{PB_{\mathcal{C}_j},\mathcal{M}_i}$ is generated for each worker $\mathcal{M}_i$ that proves that $\mathcal{M}_i$ has to run the code of $PB_{\mathcal{C}_j}$ problem.

Note that, the value n may be larger or equal to l. In the case where $l < n$, two cases may be possible. First, only l workers may have a problem to execute and the jury does not assign a problem to the other $n - l$ workers. Another possible solution may assign one problem PB to several workers. In this case, a seed chosen for each worker is added into the code of the problem PB to guarantee that each worker performs the computation works.

A.4.5 Useful work process and result announcement

The worker $\mathcal{M}_i$ executes the corresponding code Φ of the assigned problem $P_{\mathcal{C}_j}$. Then, $\mathcal{M}_i$ computes $HASH(work_c)$ where $work_c$ is the set of computations executed.

The worker $\mathcal{M}_i$ broadcasts the following result $RES_{\mathcal{M}_i}$ to prove that he has performed the problem $PB_{\mathcal{C}_j}$:

- $PB_{\mathcal{C}_j}$ the problem proposal
- $votes_{PB_{\mathcal{C}_j}}$ the set of votes from the jury of voters for the proposal $PB_{\mathcal{C}_j}$
- $assign_{PB_{\mathcal{C}_j},\mathcal{M}_i}$ the proof of problem assignment generated in Problem distribution step
- $(out, proof_c)$ the final output of the code Φ and the proof of correct computation outputted by `PROVE`
- $HASH(work_c)$ the hash of the computation executed $work_c$
- $sign_{\mathcal{M}_i}$ the signature of the participant $\mathcal{M}_i$ who executes the code Φ

The set of votes $votes_{PB_{\mathcal{C}_j}}$ enables to prove that the corresponding problem is in the list selected by the jury of voters in the Relevant problems selection. The worker $\mathcal{M}_i$ can prove his legitimacy to execute the code of the problem $PB_{\mathcal{C}_j}$ with the assignment proof $assign_{PB_{\mathcal{C}_j},\mathcal{M}_i}$. The worker $\mathcal{M}_i$ provides the values $(out, proof_c)$ outputted by `PROVE` algorithm to prove his honest and correct computational works. The hash value $HASH(work_c)$ ensures the integrity of his computation $work_c$. The worker $\mathcal{M}_i$ signs his result message $sign_{\mathcal{M}_i}$ to prove that he has run the assigned problem's code.

A.4.6 Work verification

Each participant receives a set of results $RES_{\mathcal{M}_1}, RES_{\mathcal{M}_2}, \ldots, RES_{\mathcal{M}_n}$. The participant runs `SELECTION` to learn who is selected as verifiers for the performed work for the round k.

If the participant $\mathcal{M}_i$ is a verifier, then $\mathcal{M}_i$ verifies the proof $proof_c$ of $RES_{\mathcal{M}_j}$. If $proof_c$ is correct, then $\mathcal{M}_i$ broadcasts his acknowledgement for the result $RES_{\mathcal{M}_j}$ along with the proof of selection outputted by `SELECTION`.

If there is a threshold th of acknowledgements for the same result $RES_{\mathcal{M}_j}$, then $RES_{\mathcal{M}_j}$ is considered as valid and the corresponding worker becomes a candidate for the Winner election step described in the next paragraph.

A.4.7 Winner election

Among the workers who provide valid results, a winner is selected to win the useful coins for his computational work. A random selection chooses a worker as the winner. The participants use the `ELECTION` algorithm to know who is elected as the winner to win the useful coins. The winner may be elected proportionally to the amount of useful coins owned in his wallet and his contribution to the blockchain. Note that, a participant can earn useful coins when he is selected as winner, verifier of the result provided by the winner or writer, and his contribution may be measured to the number of blocks that he added into the blockchain, for example.

The winner and the verifiers who verify the winner's result are written into the blockchain as described in the Writer election and validation step of the next paragraph. When the block is validated and considered as a part of the blockchain, then the writer and verifiers earn their corresponding part of the reward for his performed work and their verification, respectively.

The other workers who have not been chosen as winner and have proposed a valid result may recover their stake if they do not want participate in the next winner election. In either case, the fees paid by clients may be sent to the workers who provide a valid result. A problem that has been assigned to a worker but has not received a valid result may be assigned to the next round by adding a seed in the problem's code. This prevents a participant to propose a result already performed.

A.4.8 Writer election and validation

The Writer election step enables to select a participant as writer to generate and add the new block of transactions. The random election protocol `ELECTION` enables to choose one of them as writer and this latter wins the right to add his block into the blockchain. A new block B_k contains the following values:

- $HASH(B_{k-1})$ where B_{k-1} is the last block added into the blockchain
- TXS a set of valid transactions
- $(winner, \pi_{winner})$ the identity of the selected worker who wins useful coin and his winning proof
- π_{writer} the writer proof
- RW the fixed reward distributed among the winner, verifiers and writer

The value $HASH(B_{k-1})$ is the classical hash value to link the blocks of the blockchain. The set TXS has been verified to prevent issues such as the double spending as described in Section A.5. The values $(winner, \pi_{winner})$ enables to write in immutable manner who wins the useful coin for the performed work. The π_{winner} value may contain the output of `ELECTION` during the Winner election step along with the result message of the winner, the set of verifiers who verify his result and the threshold of acknowledgements. The writer proves his legitimacy with the value π_{writer} outputted by `ELECTION` during this step.

Then, a set of verifiers is selected to verify the block of the writer. The participant $\mathcal{M}_i$ uses the `SELECTION` to know who is selected as a verifier for verifying the generated block of the round k. When the block B_k receives the threshold th of acknowledgements from the verifiers, B_k is considered as valid and a part of the blockchain. The block B_k is then added into the blockchain along with the threshold

th of acknowledgements. Finally, each involved participant receives his reward, i.e. the winner, the verifiers of the result provided by the winner and the writer who generates the corresponding block.

Note that, in the case where the elected writer is malicious or offline, and so does not provide the expected block, a mechanism to replace the writer is required. Another solution is to ensure a selection among the online participants until the end of this step as considered in [191, 98].

A.5 Security analysis

In this section, we discuss some well-known and new attacks, and indicate how our UW protocol prevents them.

A.5.1 Malicious problem proposal

A malicious client may propose a problem that is a malware or has a bad impact for the society. A client may also be a worker that proposes an already executed problem with (or not) a slight modification in order to easily win coins. We prevent these problems thanks to the Relevant problems selection step that filters the problems submitted by the clients. This selection is based on the number of computations, the quantity of memory and the time necessary to run the problem's code. Additional information may be added into the problem proposal to refine the selection such as a certificate provided by a trusted party or a problem description that argue the relevant aspect of the problem.

A.5.2 Coalition problem

It is possible that several workers receive the same code to execute when there is less submitted problems than workers. In a set of malicious workers who receives the same problem, only one worker may perform the computational works for the assigned problem and shares her result and computations to others malicious workers. Thus, the set can decrease its computational effort while having the same chance to win useful coins. Our UW protocol avoids this by adding different seeds in the same code executed by several workers. Thus, the computation for a same code may be different from a worker to another and a coalition cannot provide the same result.

A.5.3 Work theft

A malicious participant may steal the result of a worker and convince other participants that she performed the work of the stolen result. Thus, the malicious participant may participate to the winner selection and so win useful coins without performing any work. Our UW protocol prevents this theft by providing a proof that a specific entity receives a submitted problem to run. Indeed, in the Problem distribution step, the voters distribute the relevant problem to the participants who want to be a worker. The agreement on who has which problem is verifiable via the assignment proof generated during the Problem distribution step. Moreover, in the case where several workers receive the same problem, the different seeds added in the problem's code during the Problem distribution step guarantee that each computational work is different and so a malicious worker cannot convince other participants that she has performed the work of another worker.

A.5.4 Fork problem

A fork in a blockchain occurs when two blocks extend the same block and compete to be in the main chain. This issue is limited in our UW protocol since in the writer election, only one writer is elected. Moreover, we ensure that only one block can reach the two thirds threshold of acknowledgements and so validate one block at each round.

A.5.5 Denial of Service attack

In a Denial of Service (DoS) attack, an attacker may flood the network of a huge amount of messages to make it unavailable. In our UW protocol, the attacker may be a malicious client that floods of fake or useless problems. We mitigate this issue thanks to the proposal and storage problem fees that limits a

client to submit several and large codes. A DoS attack may also be run by a malicious worker. Indeed, she may create several identities in order to have several assignment problems and sent to one or several victim participants false results. We prevent this issue with requiring a minimum amount of committed coins to win the right to participate at the winner election and with a punishment mechanism where the workers who provide invalid results lose their committed coins.

A.5.6 Sybil attack

In a Sybil attack [107], an attacker creates several malicious participants under different identities to participate in the protocol, e.g. to increase her influence on the voting and verifying steps. Our UW protocol is Sybil resilient since the selection algorithm `SELECTION` is based on PoS mechanism with a minimum amount of committed stake. Thus, an attacker splitting her currencies for her different identities cannot increase her influence on the protocol. She may succeed a Sybil attack only by investing at least as much coin as honest participants.

A.5.7 Majority attack

A majority attack may occur when the attacker controls more than the majority of resources in the system, i.e. useful coins in our UW protocol. An adversary that controls the majority of coin can make the blockchain useless by not forwarding messages or not participating in the vote and verification processes. However, with the two thirds threshold th of acknowledgements as described in Section A.3, she cannot validate fake work or conflicting blocks while she controls less than two thirds of the total coin. We assume then, that our UW protocol works while two thirds of the total coin owned to honest participants. Moreover, in the long run of a cryptocurrency context, for example, collecting more than two thirds of total coin could be difficult to achieve if the useful coin acquires more value.

A.5.8 Selfish mining strategies

In a Selfish mining strategy [118], an attacker takes advantage of the usual forking to temporarily hide one or several blocks and then, she reveals her blocks at the suitable time to drop honestly generated blocks from the main chain. In this way, she can increase her ratio of blocks in the blockchain and thus also her reward compared to the reward she would obtain by following the honest protocol. In our UW protocol, a participant may earn reward being a writer, a winner or a verifier. A malicious writer cannot increase her incomes by withholding her block since a writer who does not provide her block is replaced by another. A winner is a worker who has already broadcasted his performed work and so a malicious one cannot increase incomes by hiding his computing effort. Moreover, the malicious worker cannot keep this work for later since an assigned problem that is unsolved is proposed for the next round with a seed in the code. As a verifier, an attacker cannot increase his reward since a non-participation to the verification process is not rewarded.

A.5.9 Nothing-at-stake attack

The nothing-at-stake attack is generally specific for PoS-based blockchain since the generation of blocks necessitates little computational effort. Indeed, an attacker may generate blocks on multiple chain of a fork. Thus, it may guarantee that one of them is chosen to be in the main chain and the attacker can recover the currencies invested in the discarded blocks. In our protocol, a worker who proposes false or empty results is detected during the Work verification phase where verifiers invalidate the result. Moreover, with the punishment mechanism a worker may lose her stake by sending false results.

A.5.10 Double spending attack

In a double spending attack, an attacker succeeds to execute two transactions that spend the same currencies. For example, the attacker may issue a transaction that sends an amount of coins to a recipient. She may succeed to confirm it and then she may spend the same coins in another transaction, i.e. the second transaction is also confirmed. We prevent this issue in the Writer election and validation step where a group of verifiers is selected to check the block generated by the writer. If a threshold

of the verifiers validates the blocks, i.e. there is not conflicting data, then the block is added into the blockchain.

A.6 Variants and discussion

In this paper, we propose a high-level construction of our Useful Work protocol to replace the hash puzzle of the Bitcoin's PoW mechanism by useful work. Obviously, at each step of our UW protocol, several variants are possible.

In Section A.3, we proposed some possible protocols for the `SELECTION`, `ELECTION` and `PROVE` algorithms. The different steps described in Section A.4 may also be done with other mechanisms. For example, the selection of the relevant problems may be done via an external consortium of experts. The distribution of problem may be replaced by a mechanism where each participant may choose their problem to execute. The choice of protocols may be based on the security properties according to the application or the trade-off with the performance.

Note that, for each choice done in the different steps, several processes may be adapted. For example, the reward distribution may be reviewed to remain fair, i.e. each entity is rewarded according to his work invested into to the blockchain. Moreover, a security analysis may be necessary to prevent attacks as described in Section A.5.

A.7 Conclusion and future works

We presented the Useful Work protocol which is a new consensus protocol for blockchain based on the Proof-of-Stake and Proof-of-Work processes where the computing work and the memory space are dedicated to useful works. In our UW protocol, the participants compete to win useful coins. To this end, they have to run the code of problems submitted by clients. Then, a PoS-based random election chooses one of them to win the coins. A participant is then elected to generate the new block of valid transactions along with the information related to the winner of useful coins. We also presented some new issues and showed that our UW protocol is resilient to these issues and the classical attacks on the consensus protocols and blockchain.

Several interesting topics are left for future works such as formal model and security analysis of our protocol, an implementation and tests to provide a security and performance trade-off, and a detailed analysis of the scalability requirement.

université
PARIS-SACLAY

Conception, analyse et implémentation d'algorithmes de chiffrement symétrique sur FPGA

Résumé:
Cette thèse a été réalisée au sein de l'Université de Versailles-Saint-Quentin-en-Yvelines, en partenariat avec HENSOLDT France SAS.
L'objectif pendant ces trois ans était de développer, pour HENSOLDT, deux solutions de chiffrement symétrique: une famille d'algorithmes de chiffrement par blocs personnalisables, associée à un mode d'opération propriétaire et une solution de chiffrement authentifiée légère. Ces algorithmes étant la propriété d'HENSOLDT, ils ne peuvent donc pas être publiés. Les travaux présentés dans cette thèse explorent alors différents aspects de construction d'algorithmes de chiffrement symétrique et reprennent les différentes techniques utilisées pour développer ces algorithmes sans les présenter dans le détail.
Les travaux portent sur le design et l'implémentation d'algorithmes de chiffrement par blocs dits légers, ainsi que sur les fonctions éponges permettant de réaliser du chiffrement authentifié. Le but recherché dans les deux notions est de définir des solutions permettant de garantir des bornes de sécurité similaires à celles des algorithmes standards de la littérature cryptographique tout en obtenant des performances et un ratio efficacité-occupation le meilleur possible. Le manuscrit se décompose en trois grandes parties. La première est une introduction, rappelant le contexte de recherche de ces travaux et présentant rapidement l'histoire de la cryptographie depuis l'antiquité à nos jours avec certains des challenges futurs pour la cryptographie et les menaces qui pourraient apparaitre dans le futur, notamment l'ordinateur quantique. La seconde partie présente l'état de l'art et différentes techniques utilisées pour construire des schémas de cryptographie symétrique. Cette partie présente également les standards et plusieurs techniques possibles pour implémenter ces solutions sur FPGA.
La troisième partie présente mes différentes contributions qui se décomposent en trois travaux différents. Le premier consiste en l'implémentation sur FPGA d'un nouvel algorithme de chiffrement par flot autosynchronisant sur FPGA. Cet algorithme est ensuite comparé à d'autres candidats pour comparer ses performances. Malgré une implémentation plus importante en termes de mémoire nécessaire, les résultats sont satisfaisants en termes de débit offert par cet algorithme. De plus, il offre le plus court temps de resynchronisation. Le deuxième article présente une attaque sur le mode d'opération SIV. Ce mode permet d'obtenir une solution de chiffrement authentifié en utilisant un algorithme de chiffrement par blocs tel que l'AES. La particularité de ce mode d'opération est qu'il est présenté comme possédant des bornes de sécurité similaires aux modes d'opération classiques mais sans recourir à l'usage de vecteur d'initialisation. Cette spécificité est particulièrement intéressante pour le développement de notre propre mode d'opération. L'attaque présentée ici présente un distingueur permettant de casser l'authentification de cette solution avec 2^{64} messages. Le dernier travail décrit dans le corps de ce manuscrit introduit une implémentation de trois algorithmes présentés lors du processus de standardisation d'algorithmes de chiffrement authentifié du NIST. Ces algorithmes, acceptés au second round de cette compétition cryptographique sont tous les trois basés sur des fonctions éponges, ce qui les rend très intéressants par rapport à notre étude. Les résultats de l'implémentation de ces algorithmes sur FPGA sont ainsi présentés et comparés avec trois autres algorithmes basés également sur des fonctions éponges.
J'inclus également en annexe un article, présentant une première réflexion concernant la création d'un protocole permettant de rétribuer les utilisateurs d'une blockchain qui auront contribué à celle-ci en réalisation un travail considéré comme utile.
Je conclus ces travaux en présentant de futurs axes de recherche possibles, notamment optimiser les implémentations de plusieurs algorithmes et l'utilisation de solutions permettant la résistance aux analyses par canaux auxiliaires de différents schémas basés sur les fonctions éponge.