VLSI Circuits for Cryptographic Authentication

Author(s):
Henzen, Luca

Publication Date:
2010

Permanent Link:

https://doi.org/10.3929/ethz-a-006299208 →

VLSI Circuits for Cryptographic Authentication

Diss. ETH No. 19351

VLSI Circuits for Cryptographic Authentication

A dissertation submitted to
ETH ZURICH

for the degree of
Doctor of Sciences

presented by

LUCA HENZEN
Dipl. El.-Ing. ETH (MSc ETH)
born March 30th, 1982
citizen of Blatten VS, Switzerland

accepted on the recommendation of

Prof. Dr. Wolfgang Fichtner, examiner
Prof. Dr. Willi Meier, co-examiner

2010

Abstract

Nowadays, digital communication protocols rely on cryptographic mechanisms to protect sensitive information from unauthorized access and modification. Cryptographic primitives, such as hash functions, block and stream ciphers, are key components of many information security applications, employed to provide privacy, authentication, and data integrity. Following the massive differentiation of modern communication technologies, cryptography must be able to provide highly-efficient algorithms that combine strong security with optimal implementability. It becomes therefore unlikely that a single cipher or hash function would be able to meet all the constraints imposed by the wide plethora of communication protocols. Researchers and designers are thus asked to develop application-specific algorithms that are jointly optimized for security and implementation performance.

This thesis is concerned with the design of very large scale integration (VLSI) circuits for cryptographic hash functions and block cipher authenticated encryption modes. We present two hash algorithms that have been submitted to the public hash competition organized by the U.S. National Institute of Standards and Technology (NIST). A complete design space exploration of their efficiency for application-specific integrated circuits (ASICs) is given. Due to our strong involvement in the competition, we further developed a uniform methodology for a fair comparison of the VLSI performance of the second round candidate algorithms. Our benchmarking framework is the result of three different research projects that culminated with the fabrication of three 0.18 µm and 90 nm ASICs, hosting the second round function cores.

In the second part of this thesis, we investigate high-speed field-programmable gate array (FPGA) designs of the Advanced Encryption Standard (AES) in the Galois/Counter Mode (GCM) of operation

for authenticated encryption. A multi-core AES-GCM architecture is optimized to fully support the recently-ratified IEEE 802.3ba Ethernet standard for 40G and 100G speeds. Moreover, a complete hardware platform for real-time 2G fibre channel (FC) encryption is described.

Finally, after a brief introduction into low-cost hardware implementations of lightweight cryptographic algorithms, we show the first FPGA-based implementation of specific cryptanalytic attacks on the stream cipher Grain-128 and we introduce the new hash function family QUARK, tailored for resource-limited applications, such as embedded systems and portable devices.

Sommario

Molteplici protocolli di comunicazione digitale fanno affidamento su meccanismi crittografici per proteggere informazioni considerate riservate. Componenti crittografiche primarie, quali funzioni hash, cifrari a blocco e a flusso, sono al centro di svariate applicazioni di sicurezza informatica e sono utilizzate principalmente per assicurare la riservatezza, l'autenticità, e l'integrità dei dati. A causa dell'eterogeneità che caratterizza le attuali tecnologie di comunicazione, la crittografia moderna deve così fornire algoritmi altamente performanti in grado di combinare al contempo sicurezza ed efficienza. Tuttavia, è improbabile che un unico cifrario o un'unica funzione hash siano capaci di soddisfare tutte le limitazioni imposte dai sistemi di comunicazione attualmente utilizzati. Ricercatori e sviluppatori sono pertanto chiamati a sviluppare nuovi algoritmi capaci ti ottimizzare unitamente aspetti di sicurezza e implementativi.

Questa tesi è principalmente incentrata sulla progettazione di circuiti VLSI per funzioni hash e per cifrari a blocco configurati nella modalità di cifratura autenticata. Due algoritmi di hash, iscritti alla competizione internazionale organizzata dall'istituto statunitense NIST, sono presentati nella prima parte. Una completa valutazione delle loro potenzialità implementative su silicio è riportata dettagliatamente. Grazie al nostro coinvolgimento nella competizione, abbiamo sviluppato un sistema di comparazione uniforme delle performance VLSI degli algoritmi promossi al secondo round. La nostra metodologia è il risultato di tre progetti di ricerca, nei quali le funzioni hash del secondo round sono state fabbricate in tre circuiti ASIC.

La seconda parte di questa tesi si concentra sull'implementazione per dispositivi FPGA di architetture high-speed per l'algoritmo AES nella modalità di operazione GCM. I circuiti sviluppati si basano su un'architettura multi-core, offrendo una completa compatibilità con

il nuovo standard IEEE 802.3ba per traffico Ethernet fino a 100 Gbit per secondo.

Nella parte finale, dopo aver introdotto il concetto di "crittografia leggera" (a basso costo implementativo), presentiamo la prima applicazione in FPGA di attacchi crittanalitici sul cifrario a flusso Grain-128. L'ultima sezione è dedicata alla descrizione di una nuova famiglia di funzione hash, chiamata QUARK, particolarmente appetibile per applicazioni a basso costo quali sistemi embedded e dispositivi portatili.

Contents

1 Introduction

It is assumed that the science of concealing the information has been introduced with the advent of writing. The earliest form of cryptography has been discovered in Egyptian stone inscriptions, while the first most famous cryptosystem was the substitution cipher of the Roman general Julius Caesar (see [1]). During World War II, the German army deployed the mechanical rotor machine "Enigma" to encrypt and decrypt their communications. The breaking of Enigma by the British Government was the first and most appealing example of the battle between codemakers and codebreakers. However, only the combination of the post-Shannon understanding and the drastic increase of computational power allowed the birth of modern cryptography, which can be identified with two major contributions. In 1970, the research efforts of Horst Feistel culminated with the development of the first widely used cipher approved by the U.S. Federal Government under the name of Data Encryption Standard (DES). Six years later, Whitfield Diffie and Martin Hellman introduced a new cryptographic scheme that was able to solve the crucial issue of the key distribution. Nowadays, military and government institutions, as well as industries

and private persons employ cryptographic schemes to exchange sensitive information in digital form.

1.1 Basic Concepts

Cryptology is the art of communicate in secure and usually secret form. It can be divided into the science of making codes and algorithms, i.e., *cryptography*, and the science of breaking codes or extracting the meaning, i.e., *cryptanalysis*. Interesting is the analogy suggested by Vaudenay in [2], where coding theory and cryptography are distinguished by the presence of random noise in the former and the malicious adversary in the latter.

1.1.1 Communications Model

The basic communication model used in cryptography is depicted in Figure 1.1. Two parties, in general referred to as Alice and Bob, want to share some information by transferring messages over an insecure channel. The channel is considered insecure, since the communications happens in the presence of a third party, or malicious adversary called Eve, whose objective is the defeat of the security services provided by Alice and Bob to secure the communication.

The first goal of Eve is indeed to detect the encrypted message, or *ciphertext*, and to recover the original message, or *plaintext*. In addition, Eve could also try to impersonate Alice and to communicate to Bob a false or modified message.

1.1.2 Security Goals

The major objectives that can be established in a communication are:

- *Privacy*, or *confidentiality*, is the process of keeping the information secret. Only authorized entities can have access on it.

- *Data integrity* assures that the information has not been manipulated by unauthorized entities. For manipulation, we intend insertion, deletion, and substitution.

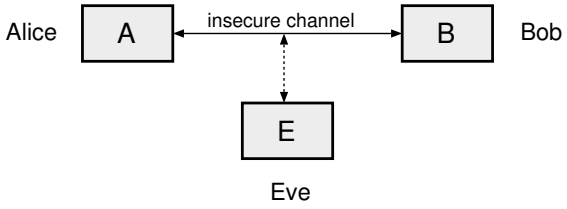

Figure 1.1: Basic communication model in cryptography.

- *Authentication*, or *authenticity*, is the general term for the process of corroborating the source of data (*data origin authentication*) and the identity of the parties (*entity authentication*).
- *Non-repudiation* is the service that prevents the denial of previous commitments or actions by an entity. In other words, the author of the message cannot deny crating or transmitting the message.

With the combination of these four cryptographic goals, it is possible to derive other security objectives as access control, validation, signature, or authorization. In general, cryptography deals with the design of algorithms and protocols that are able to guarantee one or more of these security goals.

1.2 Cryptography in Hardware

In the last decades, digital communication has drastically evolved. We have seen the widespread deployment of modern technologies in the form of wireless networks, cellular telephones, and optical fibres. Sensitive information is then transmitted instantaneously from a portable devices, such as a mobile phone, through large backbones switches to another device located somewhere in the world. Each networked entity must therefore have access to efficient cryptographic protocols, in order to implement a cryptographic layer, that secures the transmission. Popular information security protocols that rely on cryptographic schemes, are secure socket layer (SSL)/transport layer security (TLS) and internet protocol security (IPsec) for Internet

traffic and email encryption, wired equivalent privacy (WEP) and wi-fi protected access (WPA) for wireless local area network (WLAN) security, or the universal subscriber identity module (USIM) in the universal mobile telecommunications system (UMTS).

Software realizations of cryptographic protocols have the great advantage that they are portable to multiple platforms. In general, they have a fast time to market (TTM) factor. However, they can be applied in systems with limited traffic at low encryption rates. Moreover, software-based applications are not as power efficient as specialized hardware architectures. Speed and power are two major drawbacks that motivate the hardware design exploitation of cryptographic primitives.

1.2.1 High-Speed Encryption

Modern switches applied in wide area networks (WANs) or data centers process large amounts of data. With the diffusion of high-definition television (HDTV) broadcasting, video conferencing, and pay-TVs, transmission rates are growing increasingly. Real-time encryption is required by industries that prefer to concentrate the cryptographic applications in central network components to manage and to route the entire company's traffic. In this scenario, only high-speed encryption systems implemented in hardware infrastructures can handle the transmission bandwidth. System-on-chip solutions of cryptographic processors implemented in application-specific integrated circuits (ASICs) or programmed in field-programmable gate array (FPGA) devices are therefore of crucial relevance. The economical success of a modern company, involved in the information security market, is indeed related to the capability of offering in a short time reliable and highly-efficient encryption solutions for forthcoming network communication standards.

1.2.2 Portable Devices

Portable applications, such as smart cards and radio-frequency identification (RFID), are implemented on small chips that are often powered externally. The available resources in terms of silicon area, power, and processing time, are thus extremely low.

Smart cards differ from credit cards, since they host an additional embedded integrated circuit. Two categories exist: contact smart cards have a contact area, comprising gold-plated contact pads, and communicate with the reader by direct electrical connectivity; contactless smart cards are powered by the reader and communicate through radio-frequency (RF) induction. Smart cards are widely used in banking application and electronic cash systems, but they are also employed in satellite TV decoders, modern credit cards and identification systems. A sophisticated and highly secure cryptographic core is therefore implemented within the microprocessor of every smart cards.

RFID devices are microchips equipped with radio antennas. They were conceived as a way of tagging objects for automated identification. RFID tags can have a total area far below one square millimeter and can thus be easily incorporated (or hidden) in any object or person. They can also be read through barriers as wallets or pockets at a distance of up to several meters [3]. The RFID technology has predominantly been used for supply chain management, product tracking, and access control. In addition, the International Civil Aviation Organization (ICAO), a United Nations agency that defines the global passport standards, endorsed in 2003 the application of RFID in the new e-passports. The unauthorized reading of an RFID tag becomes immediately a threat that must be solved by means of specific cryptographic protocols. A good example is the challenge-response-based RFID authentication protocol (see [4]).

Highly-secure cryptographic schemes that can be efficiently implemented in low-cost very large scale integration (VLSI) architectures are therefore of paramount importance for pervasive computing applications as smart cards and RFID tags. Security, performance, and implementation costs of selected algorithms must then be coped and optimized in order to fit the limitation imposed by the target technologies [5].

1.2.3 Implementation Security

Each cryptographic system in real applications is prone to deliver additional unwanted information about its behavior. Cryptanalytic techniques that exploit specific properties of the implementation and

operating environments, are called *side-channel attacks*. Such attacks utilize information leaked during the algorithm's executions (e.g., operational time, power consumption, emitted electromagnetic radiation, etc.), rather than its theoretical and mathematical properties. Specific, and often resource-expensive, countermeasures must be implemented only where side-channel attacks are a plausible threat[1].

The analysis of side-channel resistant architectures is beyond the scope of this thesis, which principally focuses on efficient hardware designs and targets application-specific performance. We address therefore the investigation of efficient countermeasures and the immunization to side-channel analysis of our implementation as open research topic.

1.3 Contributions of this Thesis

This thesis deals principally with the design of efficient VLSI architectures of cryptographic algorithms for data integrity and authentication. We propose application-specific ASIC and FPGA implementations that target a wide range of different applications.

In particular, we focus on the VLSI design exploration of new cryptographic hash functions. With the gathered knowledge, we define a framework to compare the ASIC implementation performance of competitor algorithms. We further investigate new FPGA architectures of state-of-the-art algorithms for combined encryption and authentication, suitable for modern high-throughput multicast network security systems. Additionally, this thesis discusses low-cost cryptographic protocols. To this end, energy-efficient low-complexity VLSI architectures of new RFID-oriented cryptographic functions are presented.

The contributions of this thesis can be summarized in the following five points:

[1] Consider the case of smart cards where the measurement of the power dissipation by an external untrusted source is a realistic scenario. On the other hand, a device, such as a workstation or a network switch, placed in a secure location, is not penalized by releasing side-channel information, since it is assumed that unauthorized persons do not have access.

- *Design and analysis of new cryptographic hash functions*
  Hash functions are cryptographic primitives used principally to ensure data integrity. Due to their extreme flexibility, they can also provide generic authentication when combined with a secret key. Former hash standards, notably the MD5 function and the secure hash standard (SHA) families, have been designed and optimized for software applications. Promoted by the National Institute of Standards and Technology (NIST), a public hash competition has started in 2008, which encouraged researchers to create and present new candidate algorithms.

  Within this thesis, we present our contribution in the development of two candidate algorithms. Our main tasks has been the evaluation of their hardware efficiency, providing useful information for optimizations in this direction. At present, one algorithm is still competing to become the next worldwide-recognized hash function standard.

- *Development of a hardware evaluation method for cryptographic algorithms*
  Pushed by our involvement into the analysis of the VLSI performance of two hash candidates, we decided to extend our evaluation to all fourteen algorithms, accepted to the second round of the NIST hash competition. Within three student projects, we designed several architectures that were fabricated in three distinct ASICs. With the achieved experience, we describe here a fair and uniform methodology to compare the VLSI performance of selected cryptographic algorithms. We strongly advise this benchmark approach to the final round of the hash competition, where the number of candidates will be reduced to five algorithms.

- *FPGA implementation of high-speed architecture for authenticated encryption*
  Algorithms for authenticated encryption are the most suitable solution to reliably secure a network link. With the adoption of the new 40-100 Gigabit Ethernet protocol in the IEEE 802.3ba standard [6], speed rates of authenticated encryption algorithms must be extended beyond the limit of available field-programmable

hardware platforms. To this end, we present an efficient architecture for high-speed authenticated encryption, that fully supports the new Ethernet standard.

Furthermore, we describe the design and implementation of a complete 2G fibre channel (FC) communication system in FPGA. The network encryptor enhanced with special single-photon detectors for key distribution has been applied within the SwissQuantum project [7], in order to demonstrate the feasibility of quantum cryptography in practice.

- *FPGA implementation of high-dimensional cryptanalysis tools*
  The generic class of cryptanalytic attacks *cube testers* [8] can be used to detect nonrandomness or weak algebraic properties in cryptographic algorithms. Cube testers have the potential to be applied on broad classes of functions. A good example are lightweight algorithms designed for low-end systems, since they often build on low-degree equations.

  This thesis reports on the first implementation of high-dimensional cube testers in hardware. Our design exploits the intrinsic speed improvement of modern FPGA chips to improve the potentiality of the attacks on the stream cipher Grain-128. An extrapolation of our results suggests the cipher may not provide full protection when 128-bit security is required.

- *Design and evaluation of a lightweight hash function family*
  Lightweight cryptography deals with the design of algorithms and protocols tailored for low-resource applications as smart cards or RFID. Security robustness is optimized and balanced in accordance to performance and implementation costs of the algorithm. Particular attention is given to the reduction of the overall requirements in silicon area and power dissipation.

  The last part of this thesis is dedicated to the presentation of the new hash function family QUARK, composed of three distinct instances for different security/performance trade-offs. QUARK is a valuable solution to the need of lightweight cryptographic hash functions, repeatedly expressed by application designers. Implementation results in a 0.18 µm complementary metal-oxide semiconductor (CMOS) technology demonstrate the suitability

of the function to ultra-lightweight environments. To the best of our knowledge, the presented ASIC circuits are the smallest implemented hash functions at equal security.

1.4 Thesis Outline

The reminder of this thesis is divided into five main chapters.

Chapter 2 introduces the basic cryptographic algorithms and protocols. This chapter is purely conceived to give an insight into the field of modern cryptography. Selected hardware architectures for cryptographic applications are discussed in the next chapters.

Chapter 3 is entirely dedicated to cryptographic hash functions. In the first part, we illustrate the major hash constructions and review the main architectural approaches. An overview of state-of-the-art hash functions, which motivates the start of the NIST hash competition, is subsequently given. In the central part of this chapter, we presents two new hash functions (both submitted to the hash competition) that were designed to meet the needs of modern communication protocols. Finally, a novel methodology to fairly compare cryptographic algorithms, e.g., the second round candidates of the hash competition, is exposed.

Chapter 4 focuses on authenticated encryption algorithms for high-speed applications. We demonstrate how to achieve multi-gigabit speed ratios in modern FPGA devices, providing reference figures for throughput-optimized architectures of selected algorithms. We conclude this chapter with the description of a network encryption systems, supported by a series of measurements in a real 2G FC link.

In Chapter 5, the major problems related to lightweight cryptography are formulated. We further introduce an efficient cryptanalytic tool to evaluate the security of algorithms with a low-algebraic degree and illustrate its application to the hardware-oriented stream cipher Grain. Finally, we present a new hash function family with reduced hardware requirements.

Eventually, Chapter 6 gives a summary of the main contributions and draws the conclusions.

2 Fundamentals

This chapter introduces the basic algorithms used in cryptography and it lists their security properties with a short overview on their application fields. Security goals as confidentiality and data integrity are indeed achieved by means of generic ciphers and one-way functions. Public-key cryptography is also briefly mentioned. In particular, we explain the strong need of entity and message authentication in a general public-key communication scheme.

This thesis deals marginally with cryptanalysis and cryptanalytic attacks. A detailed definition of principles and methodologies for cryptanalysis is beyond the scope of this thesis. However, we present in Chapter 5 an approach to improve the efficacy of a specific class of attacks by exploiting hardware resources. To correctly consider the security of cryptographic primitives and to evaluate the effect of cryptanalytic attacks, some insights in the field of cryptanalysis are therefore needed.

Eventually, quantum cryptography and quantum key distribution (QKD) are presented. As complementary part of the hardware system for high-speed authenticated encryption described in Chapter 4, the

BB84 protocol for QKD is also briefly discussed. A final remark on the necessity of authentication in quantum cryptography protocols is presented.

2.1 Notation

Within this work, a standard notation for algorithms and mathematical computations is used. The cryptographic theory of this thesis as been shaped on two main sources, the "Handbook of Applied Cryptography" by Menezes et al. [4], and "Quantum Computation and Quantum Information" by Nielsen et al. [9].

With the term *variable*, a single mathematical entity or an array of smaller entities of w bits is defined. Generally, this entity, also called *word*, is the smallest computational unit on which the arithmetic of an algorithm is defined. We define the i-th word of the variable X as x_i with $X = (x_0, x_1, \ldots)$. Normally, the size w corresponds to 32, resp. 64-bits, or also to 8 bits, depending on the underlying function (in this case the term word is replaced by *byte*). In Table 2.1, the main operators and symbols are summarized.

2.2 Symmetric-Key Cryptography

Symmetric-key cryptosystems form the first category of cryptographic primitives. Here, the two parties involved in the communication (e.g., Alice and Bob of Figure 1.1) share a fixed-length stream of bits. If this bitstream is secret and considered as authentic, it could be used as secret key. Through symmetric-key algorithms and one-way functions, the two parties are able to transfer information that requires confidentiality, data integrity, and authentication.

Symmetric-key cryptography offers several advantages with respect to asymmetric schemes (see Section 2.4). In general, the major benefits are its high efficiency in terms of speed and the use of shorter keys. The drawback is however related to the key distribution. Since the parties need to share a common secret key, an additional channel

Table 2.1: Basic operators.

Symbol	Description
$\leftarrow$	variable assignement
$+$ or $\boxplus$	addition modulo 2^w
$-$	subtraction modulo 2^w
$\oplus$	bitwise exclusive-OR (XOR)
$\wedge$	bitwise logical AND
$\vee$	bitwise logical OR
$\neg$	bitwise negation
$\gg \cdot$	shift of $\cdot$ bits towards less significant bits
$\ll \cdot$	shift of $\cdot$ bits towards more significant bits
$\ggg \cdot$	rotation of $\cdot$ bits towards less significant bits
$\lll \cdot$	rotation of $\cdot$ bits towards more significant bits
$\lvert \cdot \rvert$	bit length of $\cdot$
$\lceil \cdot \rceil$	ceil: smallest integer value equal to or larger as $\cdot$
$\lfloor \cdot \rfloor$	floor: largest integer value equal to or smaller as $\cdot$
$\cdot \parallel \cdot$	concatenation of two varables

that is both secure and authenticated is required[1]. In large networks, each communication link needs therefore a defined symmetric key, hence increasing the complexity of the key management.

Symmetric-key cryptosystems rely on few basic primitives, which constitute the building blocks of more complex cryptographic schemes. In the next sections, we introduce the main principles of generic ciphers.

2.2.1 Block Ciphers

Block ciphers are the first class of symmetric-key algorithms. They are used to provide confidentiality. Due to their versatility they could

[1]Normally, the additional secure channel is significantly slower than the public channel and is typically accessible for short time periods.

also be used to generate other cryptographic primitives like stream ciphers or hash functions.

A generic block cipher is a function that maps a block of fixed length n into another block of the same length using a k-bit secret key. This process is called encryption and could be expressed as follows

$$E_K : \{0,1\}^n \times \{0,1\}^k \mapsto \{0,1\}^n, \tag{2.1}$$

where $E_K(P) = C$ is the invertible mapping of the plaintext block $P \in \{0,1\}^n$ into a ciphertext block $C \in \{0,1\}^n$ with a key $K \in \{0,1\}^k$. The inverse mapping corresponds to the decryption function, i.e., $E_K^{-1}(C) = P$. The ideal block cipher should behave as a pseudo-random permutation (PRP)[2].

In most cases, the plaintext often exceeds the block size n. It is then accordingly partitioned in n-bit blocks and processed by E_K under a specific *mode of operation*. The most common modes of operation for block ciphers are:

- **Electronic codebook (ECB)** Each plaintext block is passed to the encryption function and mapped to a ciphertext block. Decryption is applied inversely and independently to each ciphertext block in order to recover the original plaintext blocks.
- **Cipher-block chaining (CBC)** The input of the encryption function is the combination of the plaintext block with the previously computed ciphertext block. For decryption, the inputs are the ciphertext blocks, while the plaintext is recovered by XORing the previous ciphertext block with the output of the decryption function. The CBC requires an initialization vector, denoted IV.
- **Cipher feedback (CFB)** Ciphertext blocks are generated by combining the plaintext blocks with the sequence of outputs of the encryption function. The ciphertext block is the input of the next encryption function. The CFB mode requires an IV as initial input block.
- **Output feedback (OFB)** The encryption function is applied iteratively on its outputs (starting with the IV). The sequential

[2]Consult [10] for the formal definition of pseudorandomness.

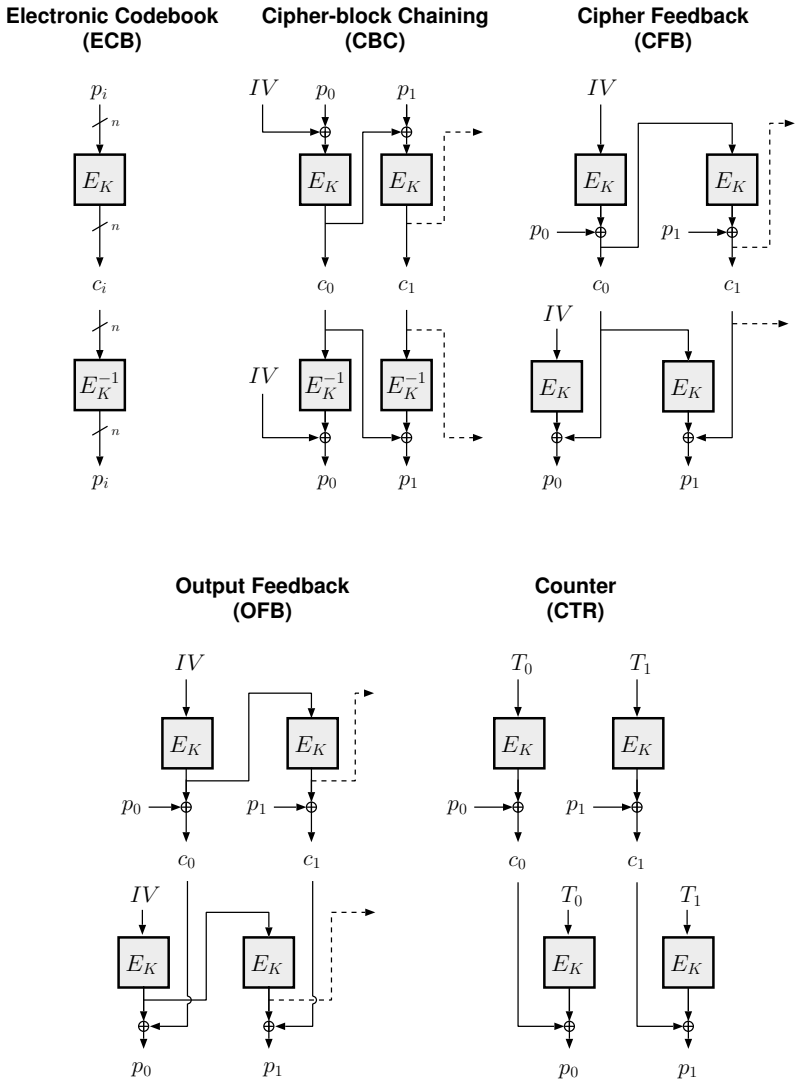

Figure 2.1: Block cipher mode of operations for encryption (top) and decryption (bottom). We assume that the plaintext message $P = p_0, p_1, p_2, \ldots$ is mapped into the ciphertext $C = c_o, c_1, c_2, \ldots$ with secret key K.

output blocks are XORed with the plaintext blocks to generate the ciphertext.

- **Counter (CTR)** A set of input blocks called counters is encrypted to produce a sequence of output blocks, which are then combined with the plaintext to generate the ciphertext. As fundamental property, the counters must never be repeated with the same key, i.e., the same key-counter pair must be used only once.

Figure 2.1 shows the schematics of the main modes of operation. The only modes that apply the inverse mapping E_K^{-1} in the decryption process are ECB and CBC. Note that ECB and CTR do not use any feedback chain for encryption and decryption (the blocks are processed independently). This means that the encryption of the next block could start before the end of the encryption of the previous block.

Typical values for the block size n are 64 or 128 bits. The size of the key instead defines the security of the encryption. In general, the key space should be large enough to prevent the exhaustive search as in brute-force attacks (see Section 2.5). The minimal key length is currently 80 bits, while modern block ciphers support keys up to 256 bits. DES and the Advanced Encryption Standard (AES) are two of the most used block ciphers.

2.2.2 Stream Ciphers

The second class of symmetric-key algorithms are *stream ciphers*. Similarly to block ciphers, they provide confidentiality. But instead of encrypting the plaintext in blocks, they encrypt individual bits one by one.

A stream cipher is also indicated as a *keystream generator*. An output bitstream $Z \in \{0,1\}^l$ is produced starting from the key $K \in \{0,1\}^k$ and an initial value $IV \in \{0,1\}^n$. The formal mapping of the keystream generation is defined as

$$S_K : \{0,1\}^n \times \{0,1\}^k \mapsto \{0,1\}^l, \tag{2.2}$$

where the keystream size l is often a large value, e.g., 2^{64}. The keystream generation should ideally be a pseudorandom generator (PRG), i.e., the keystream should be generated pseudorandomly.

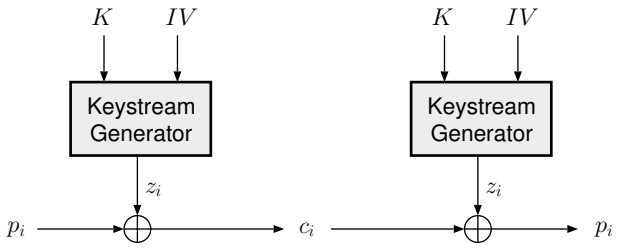

Figure 2.2: Encryption and decryption scheme of a stream cipher.

In stream ciphers, the ciphertext is obtained by combining the keystream with the bits of the plaintext, similarly to the CTR mode for block ciphers. Typically, this transformation is accomplished by a simple XOR function. The general model of stream ciphers is depicted in Figure 2.2.

When the single bits of the keystream Z are generated independently and randomly, i.e., the size of Z is equal to the size of K, the stream cipher is called *one-time pad*:

$$c_i = p_i \oplus k_i \qquad \text{for } i = 1, 2, 3, \ldots \tag{2.3}$$

This type of cipher is considered as unconditionally secure, since the generated ciphertext is statistically independent of the plaintext. However, having a perfectly random key that is as long as the message is almost impossible in practice. Therefore, functions that generate a keystream from a fixed-length key, as for stream ciphers, are required.

Stream ciphers and keystream generators are often built on feedback shift registers (FSRs), mainly because they are well-suited for hardware implementation and they easily generate sequences of large periods, e.g., linear feedback shift registers (LFSRs) with primitive polynomials as update function.

Stream ciphers are used in several telecommunication environments where resources and buffering capability are limited. For example, the GSM cellular telephone standard employs the A5/1 stream cipher to provide encrypted over-the-air communications. However,

the most widely-used stream cipher is probably RC4. Due to its simplicity, information protocols, like SSL for network communications or WEP and WPA for wireless cards, base their security on it.

Organized by the European Network of Excellence for Cryptology (ECRYPT), the eSTREAM Project was a multi-year effort running from 2004 to 2008, which identified a portfolio of promising stream ciphers. Currently, the eSTREAM portfolio contains four software-oriented ciphers and three hardware-oriented ciphers[3].

2.3 Cryptographic Hash Functions

In modern cryptography, *hash functions* are fundamental primitives. Unlike stream and block ciphers, hash functions provide data integrity. They can be described as the transformation of an arbitrary-length input (or message) into a fixed-length output, often called *hash-value*, or simply *hash*. Normally, the maximal input size is bound to a fixed number m, e.g., 2^{64}, while in typical applications $128 \leq n \leq 512$.

Since the size of the input message is typically larger than n, the hash function mapping

$$H : \cup_{i=1}^{m} \{0,1\}^i \mapsto \{0,1\}^n \tag{2.4}$$

is often considered as a compression. Most importantly, the generated hash-value is a compact and unique representation of the message that can be seen as a digital fingerprint or a message digest.

The hash mapping must literally be a *one-way* transformation. This means that the hash of a message is easy to compute, but it is significantly harder to reverse the process. The security requirements for hash functions are:

- *Preimage resistance:* for each element of the output domain it is computationally infeasible to recover the inputs, i.e, given a hash-value y, find x such that $H(x) = y$.
- *Second preimage resistance:* for a fixed input, it is computationally infeasible to find a second different input that generates the same output, i.e. given x with $y = H(x)$, find $x' \neq x$ such that $H(x') = y$.

[3] see `http://www.ecrypt.eu.org/stream/`.

Figure 2.3: General model of hash functions and keyed hash functions.

- *Collision resistance:* it is computationally infeasible to find two distinct inputs with the same hash-value, i.e., find x and x' with $H(x) = H(x')$.

Hash functions are used to detect modifications of the message. Combined with a key, they are able to provide data and entity authentication, or in general *authenticity*. In this case, we refer to *keyed hash functions*[4] with the mapping expressed as

$$H_K : \cup_{i=1}^{m} \{0,1\}^i \times \{0,1\}^k \mapsto \{0,1\}^n. \tag{2.5}$$

Keyed hash functions, which ideally should be pseudorandom functions (PRFs), are also called message authentication codes (MACs). The general model of unkeyed and keyed hash functions is depicted in Figure 2.3. The hash-based message authentication code (HMAC) construction is the most common solution to build a MAC from a unkeyed hash function (see [11]).

Hash functions are ubiquitous algorithms employed in most of today's information security protocols, notably digital signatures and authentication codes. Currently, the two most commonly used algorithms are MD5 and SHA-1. Due to their widespread use, hash functions are faced with fierce demands in terms of security, while in parallel, they must offer optimal speed performance at low implementation costs. Note that authentication protocols are broadly applied in bandwidth-intensive WANs, as well as, in resource-limited smart card systems.

[4] In the first case, i.e., when the hash function takes as input only the message, we speak of *unkeyed hash functions*.

2.4 Authentication in Public-Key Cryptography

In contrast to symmetric-key cryptography, the communicating entities in *public-key cryptography* are allowed to exchange keying material that is not secret. The encryption and decryption scheme is based on asymmetric-key algorithms that generate a related key pair: the public key and the secret private key. Public-key cryptosystems rest on the assumption that it should be computationally infeasible to recover the private key solely from the public key. In the standard situation, Bob generates a public key, which is passed through the insecure channel to Alice. Alice uses the public key to encrypt the message and sends the encrypted message to Bob. After receiving, Bob uses the private key to decrypt the message. In this way, public-key cryptography solves the problem of key management and distribution.

The most famous public-key algorithm is RSA, which takes the name from its inventors Rivest, Shamir, and Adleman [12]. RSA relies on the factorization problem of large integer numbers.

Public-key cryptography alone provides confidentiality. However, it strongly requires that the exchanging process of the public key is authenticated. In the eavesdropping model, an adversary placed between Alice and Bob could impersonate Bob by sending Alice his public key. The received message is then decrypted by the adversary with the related private key and further forwarded to Bob encrypted under Bob's public key. In this scenario, also called *man-in-the-middle* attack, the adversary eavesdrops the message without officially breaking the cryptographic system. This simple example demonstrates the necessity to authenticate the public key.

The basic difference in the generalized communication model between symmetric-key and public-key cryptography is illustrated more clearly in Figure 2.4. In the former, Alice and Bob have a secure and authenticated channel available. This channel is used to exchange the secret keys, while encrypted messages are transmitted over the insecure channel. In the latter, there is no secure channel that links Alice and Bob. They still require an authenticated channel to exchange the public key.

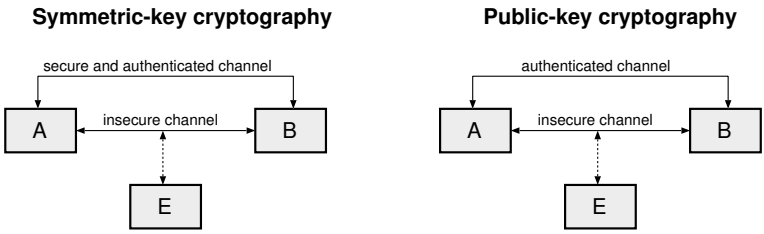

Figure 2.4: General communication model for symmetric-key and public-key cryptography [13].

2.5 Security of Cryptographic Functions

Within this thesis, cryptographic algorithms are often classified or compared using their security. A cryptographic primitive is considered as *unconditionally secure* (or *information-theoretically secure*) when its security statements are guaranteed even under the assumption that the adversary has complete access to the system and unlimited computational power (see one-time pad). However, unconditional security is conceived unattainable in practice; hence, most cryptographic models are *computationally secure*, i.e., the effort required to recover some information on the secret variables is equivalent to solving a computational problem that is considered as intractable, and even with the fastest known computers would take too much time. An example is the factorization of large integers (1024 to 2048 bits) in the RSA algorithm.

In the optimal case, the security of a cryptographic function is defined by the size of its specific security parameters. For generic ciphers, this coincides with the size of the secret key, e.g., the AES-128 (key size is 128 bits) has a security of 128 bits. Or, hash functions assure a security against preimage cryptanalysis equal to the hash-value length. The complexity of an attack becomes thus exponential with respect to this size. The basic attack on a cipher working with a k-bit key need indeed 2^{k-1} calls of the cipher to recover the key. Such an exhaustive search is identified as *brute-force attack* and involves the exploration of the entire key domain. Brute-force attacks define the bound under which another attack would effectively mine the

security claim of a cryptographic function. Consider the possibility of a hypothetical attack that is able to find the key of the AES-128 in 2^j attempts with $j < 128$. Under the assumption of acceptable memory requirements by the attack (see [14]), the AES-128 could no longer ensure 128-bit security.

Attacks on cryptographic primitives are enclosed within the field of cryptanalysis. Normally, cryptanalysis exploits some mathematical property of the algorithm to get an insight in its functioning. It is often defined as the science of developing methods to acquire some knowledge of the secret information in a cryptographic scheme.

2.6 Quantum Cryptography

When the physical information is enclosed in the state of a quantum system, we speak of quantum information. In the last years, the efficacy of quantum information against several well-known public-key cryptosystems has successfully been demonstrated (e.g., Shor's algorithm for quantum integer factorization). Nevertheless, the laws of quantum physics can also be exploited to achieve provable security[5]. Referred to as *quantum cryptography*, this particular branch of cryptography relies its security on the principles of quantum mechanics and, in particular, on the fact that the measure of a quantum signal perturbs the state of the signal.[6].

The major issue of symmetric-key encryption is the exchange of a common string of bits used as key between Alice and Bob. Somehow the key must be securely distributed through a secure channel, which can be established by quantum cryptography.

[5] The cryptographic system is provably secure, if the difficulty to be broken is similar to solving a hard problem. In the case of quantum cryptography, the hard problem is based on the laws of physics.

[6] In quantum information, a quantum signals (used to communicate a bit of information) is often referred to as *qubit*, or quantum bit. Qubits can be made by electrons, atoms, or, as in most applications, photons. For the latter, the polarization is its quantum-mechanical state.

2.6.1 Quantum Key Distribution (QKD)

Quantum cryptography is mainly employed to solve the key distribution problem, rather than the direct exchange of the message. In a QKD system, the private key is indeed transmitted through a public channel by quantum signals. The established key is then used by canonical symmetric-key algorithms, as the one-time pad, to achieve provable security, or with other ciphers.

The security of the quantum key agreement relies on the assumption that an eavesdropper can not gain any information on the key without disturbing the transmission of the key. Alice and Bob can therefore detect the presence of an eavesdropper by observing the error propagation rate of the quantum communication.

As previously mentioned, photons in different polarization states constitute a well-suited way to send quantum signals. They can indeed be transmitted over conventional optical fibres and measured with modern single photon detectors.

2.6.2 The BB84 Protocol

The first scheme for QKD has been proposed in 1984 by Charles H. Bennet and Gilles Brassard. It takes the name from its inventors, hence *the BB84 protocol*. BB84 has been applied in several quantum key encryption systems, and it is a key component of the high-speed encryption system presented in Chapter 4.

In the standard scenario where Alice and Bob want to establish a common secret key and Eve tries to gain some information on it, the BB84 protocol works as follows (cf. Table 2.2):

1. Alice selects a fixed number of bits and encodes each bit into quantum signals. The states of the quantum signals are randomly chosen between two non-orthogonal basis.

2. Alice sends the quantum signals to Bob through the public and insecure channel.

Table 2.2: Basic procedure of the BB84 protocol. The bits $\{0, 1\}$ are mapped into $\{\uparrow, \rightarrow\}$ for the vertical-horizontal basis "+", and $\{\nwarrow, \nearrow\}$ for the diagonal basis "×". Without eavesdropping, about half of the transmitted bits will be correct (see highlighted bits in the last row).

Alice:												
Bits	0	1	0	1	0	1	0	0	1	1	0	0
Basis	+	×	×	+	×	+	+	×	+	×	×	+
Q.Sig.	↑	↗	↖	→	↖	→	↑	↖	→	↗	↖	↑
Bob:												
Basis	+	×	+	+	+	×	×	+	+	×	×	×
Q.Sig.	↑	↗	→	→	↑	↖	↗	↑	→	↗	↖	↖
Bits	0	1	1	1	0	0	1	0	1	1	0	0

3. Bob measures the received signals by applying one of the two basis at random[7].

4. After receiving the confirmation that Bob has measured the signals, both send each other the sequence of the used basis. Half of the bits should be generated by Alice and measured by Bob using the same basis. The other half is discarded.

5. To check the presence of Eve, they publish and compare a subset of the remaining bits, which then are no more used to establish the secret key. If more than an acceptable number[8] disagree, they abort the protocol, otherwise they can use the remaining bits to generate the secret key.

The security proof of the last point is based on the fact that, during measurement, Eve loses the information on the state of the quantum signals. She is thus not able to replicate the state of the measured

[7] A basic rule of quantum physics states that it is impossible to simultaneously measure the state of a quantum signal in two different basis, e.g., measure a photon in the vertical-horizontal basis and simultaneously in the diagonal basis.

[8] The threshold value upon which the communication is considered eavesdropped is defined by the desired key length and the rules of information reconciliation [15, 16] and privacy amplification [15, 17].

quantum signals and to further propagate them intact to Bob. Unknown quantum states are indeed impossible to be duplicated. Doing so, Eve is forced to increase the error probability within the quantum communication. The close monitoring of the error rate in the shared subset of bits is therefore the only method to detect the intrusion of Eve.

2.6.3 Authentication in Quantum Cryptography

In general, QKD shows the same weakness against the man-in-the-middle attack as public-key cryptosystems. The main security flaw is that nothing prevents Eve to come between Alice and Bob and to interact with them by impersonating the other. In this scenario, Eve does not try at all to figure out the key; instead, she pretends to be Bob with Alice and Alice with Bob. She will then establish a completely secure key with Alice and another one with Bob, by means of QKD. Alice and Bob are indeed not able to determine if the received quantum signals or the information related to the basis are coming from the correct partner.

As for public-key cryptography, the only solution to the man-in-the-middle attack is the authentication of the parties. This is only possible under two basic assumptions: first, *a priori* Alice and Bob must share a common secret key; second, together with this key, they need classical authentication algorithms as MACs, to be sure to talk with the correct person [18]. The BB84 protocol becomes thus a sort of key expansion scheme, or more accurately *quantum key growing* scheme, where from an initial secret key it is possible to generate new provably secure keys.

3

Cryptographic Hash Functions

The operation modes of hash functions are based on a wide range of different constructions. In the first part of this chapter, we introduce the general model for iterated hash and discuss the main issues that arise in the VLSI implementation of a cryptographic hash algorithm. To this end, we provide the required metrics to evaluate and characterize the efficiency of hash circuits.

Within this thesis, we demonstrate that implementation aspects can not be overlooked during the design of a hash algorithm. Modern hash functions are indeed faced with an increasing demand in security (see [19]), while at the same time flexibility and optimal performance are essential properties when the algorithms are integrated in real systems. Security and efficiency are however design principles that tend to hinder each other.

In Section 3.2, we briefly describe the major motivations that led to the ongoing NIST hash competition. Our interest in the competition started with our involvement in the development of the candidate

algorithms BLAKE and EnRUPT. In Section 3.3.2 and Sections 3.4.2-3.4.3, we present a complete design space exploration of the two hash function families.

The last part of this chapter is devoted to the definition of a framework to evaluate the hardware performance of generic symmetric cryptographic algorithms. To demonstrate the validity of our methodology, we applied this framework within the second round of the NIST hash competition.

During the entire chapter, we mainly describe hardware implementations targeting CMOS designs for ASIC fabrication. The exploitation of specific macro blocks as bloc RAMs (BRAMs) in FPGA devices is therefore not considered.

3.1 Iterated Hash Functions

Unkeyed hash functions do not process simultaneously the whole input message. Hence, the message is split into fixed-length blocks that are processed sequentially by the *compression function* as iterative process. The compression occurs when the message block is "combined" with the internal state (or *intermediate chaining variable* h_i) to generate the next state.

In most cases, the size of the message is not a multiple of the message block length r. In the preprocessing phase of iterated hash functions, some bits are thus appended (*padding*) at the end of the last message block until the length becomes compatible.

3.1.1 The Merkle-Damgård Construction

With the iterated model, the major concerns are in the security of the resulting function. Is the iterated model resistant against collisions and preimage searches? The answer came simultaneously from Ralph Merkle and Ivan Damgård in 1989 [20, 21]. They proved that from a collision-resistant compression function F it is possible to build a hash function H with the same property by applying the

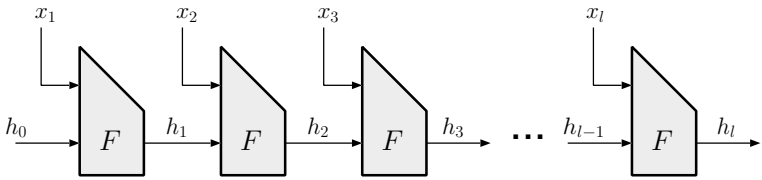

Figure 3.1: Merkle-Damgård iterated construction.

so-called Merkle-Damgård (MD) construction (see Figure 3.1). Using a collision-resistant mapping

$$F : \{0,1\}^n \times \{0,1\}^r \mapsto \{0,1\}^n, \tag{3.1}$$

the n-bit hash-value of the padded message $x = x_1 x_2 x_3 \dots x_l$ is computed as follows

$$H(x) = h_l \quad \text{with} \quad \begin{cases} h_0 = IV & i = 0 \\ h_i = F(h_{i-1}, x_i) & i = 1, 2, \dots, l \end{cases} \tag{3.2}$$

The result of $H(x)$ is directly returned from h_l, hence the internal state has the same size as the n-bit hash-value. In the particular case that the last block is additionally padded with the bitlength of the message (*MD strengthening*), the collision resistance (and also preimage-resistance) of the compression function is securely preserved in the hash function.

Almost all known iterated hash functions currently in use are derived from the MD model (e.g., MD5, the SHA-1 and SHA-2 families). In the last year, some security issues and limitations have been discovered (cf. [22] for an overview). After these findings, some important improvements to this construction have been presented (see Section 3.1.4).

Following the proof of Merkle and Damgård, the security of a hash function relies almost entirely on the compression function. The map F becomes therefore the fundamental component for the security and the implementation efficiency. Hardware architectures of MD-based hash functions can then be classified following some basic parameters related mainly to the compression function.

Before analyzing the hash function constructions and the design strategies, the next section introduces the main parameters needed to evaluate the hardware implementation of hash algorithms.

3.1.2 Hardware Specifications

Hash functions are employed in numerous applications with diametrically different requests, e.g., compactness, low power dissipation, high-speed, etc. It is extremely difficult that a single architecture can fulfill the requirements imposed by the entire application spectrum. We address this problem and deeply discuss the issues inherent in the flexibility of hardware designs in Section 3.5.

The various performances of a hash function hardware implementation are computed using a set of defined parameters. We list the most important ones, with relative metrics:

- **Circuit area** A**:** it indicates the amount of logic cells required to implement the hash function. In this thesis, we favor the expression of the circuit size in the *gate equivalent (GE)* count[1], instead of μm^2.

- **Clock frequency** f**:** the clock rate that drives the circuit. In synchronous designs, the maximum frequency is the inverse of the critical (i.e., longest) path of the circuit.

- **Latency** L**:** Total number of computation cycles required by the circuit to hash the message. Since the compression function is generally defined over a number of rounds R, the latency can be defined as R times the number of message blocks l.

- **Throughput** Θ**:** the rate at which the bits are hashed with respect to time. It is computed by multiplying the frequency with the quotient of message size and latency. The throughput is often expressed at the maximum value, i.e., with long messages, so that initialization and finalization procedures are omitted as well as discontinuities due to padding (see [23]).

[1]The GE area of the circuit is obtained by dividing the total area by the size of a two-input drive-one NAND gate in the target CMOS technology.

- **Hardware-efficiency:** the throughput to area ratio. The efficiency is exactly the inverse of the AT-product (see [24]).

- **Power consumption:** it corresponds to the amount of energy dissipated in a certain time span; in our case, the time required to hash the message. According to [24], the energy dissipation in CMOS designs is composed mainly by the (dis-)charging of the capacitive loads, driving of possible resistive loads, crossover and leakage currents.

- **Energy efficiency:** the value that indicates the energy required to hash a single bit. It is computed by dividing the power consumption by the throughput of the circuit and is given in Joules per bit [J/bit].

Several efforts in comparing different designs for cryptographic hash functions have been done in the literature. Most works focus on the evaluation of the circuit performance targeting two main constraints: maximum achievable throughput and lowest possible area; goals that are often incompatible and can not be achieved by a single IC design. Nevertheless, we consider a hash function as efficient or versatile in hardware, if it is able to reach these goals even with different architectures.

Maximization of the throughput

The crucial characteristics required by a hash algorithm to achieve high speed in hardware are a fast compression function (possibly parallelizable) and a small number of rounds per message block size ratio.

Hash algorithms based merely on Boolean functions show obviously better performance compared with algorithms with modular additions or substitution tables (S-boxes). The work presented in [25] compares the hardware performance of different hash function types; the algorithm based on Boolean transformations clearly outperforms (in throughput) the schemes based on additions and S-boxes.

Reduction of the circuit size

Compact implementations benefit mainly from a high modularity degree of the compression function. When the computations in F are similar and the flow dependencies between the internal variables show limited complexity, the circuit can indeed be reduced to some basic computational units that are shared within the rounds.

Nevertheless, a good modularity of the hash function is not the unique aspect that leads to compact architectures in hardware. The logic used to store the internal variables (internal state, message block, constants, etc.) constitutes, in most cases, the bottleneck of the area reduction. Taking into account that a single register (flip-flop standard cell) has a size of about 6 GE in United Microelectronics Corporation (UMC) technologies, the maximum storage capacity of a reference 10 kGE-circuit is far below 1600 bits.

In Section 3.4.3, we propose special-purpose semi-custom memories which reduce sensibly the area and power consumption of the storage logic. However, this specific design approach can be applied only in particular cases where only small parts of the internal variables are updated once per cycle, which is not the case for FSR-based algorithms.

More in general, cryptographic algorithms are faced with the *security vs. efficiency* problematic. Usually, the more secure a function is, the less efficiently can it be implemented. The broad fulfillment of all security criteria indeed drastically affects the performance of the function. In the ideal case, the algorithm meets (with a comfortable margin) the security requirements of a limited subset of similar applications and then optimizes the performance following the correspondent system constraints.

3.1.3 Design Strategies

Without considering the compression function, iterated hash functions offer limited architectural variations[2]. Due to the iterative structure

[2]The specification of a wide range of architectural transformations for digital IC designs is given in [24].

and the assumption that the message can be of variable length, hardware implementations are generally designed over a single compression. This is mainly motivated by the recursive procedure of the internal state, where the output of a compression is always used in the following compression.

Under this structural limitation, the major design optimizations target the compression function. Since F typically schedules several rounds, the internal state is updated (R times) by almost the same transformation. Hence, most architectures are split into the logic that computes a single round of F (*datapath*), plus some registers (*memory*) used to store the internal state h_i and, if needed, the message block. The message block is often used during the entire compression and needs therefore to be stored somewhere.

In this standard design, one round of the compression function is computed within a clock cycle and the message blocks are given to the circuit each R cycles. The maximal throughput can then be approximated as

$$\Theta = f\frac{m}{L} \cong f\frac{r}{R}. \tag{3.3}$$

It is mainly defined by the ratio between the message block size r and the latency per block R (m is the total bitsize of the message). Generally, the larger is the size of the input block, the more rounds are required. As example, the hash function Hamsi works on 32-bit blocks with $R = 3$, whereas the hash function Keccak has 1088-bit blocks and requires 24 rounds. The proportion between these two parameters is set by the security constraints of the function, but it also drastically affects the speed performance of hardware designs.

Throughput and HW-efficiency

Not many architectural transformations allow to increase the speed of a hash circuit. Techniques as pipelining or replication can not be considered due to the iterative nature of the hash process. Nevertheless, some algorithms can take benefit from *partially unrolled* architectures [26, 27].

In unrolled designs, multiple successive round units are implemented, instead of a single round unit (as is the case for the standard

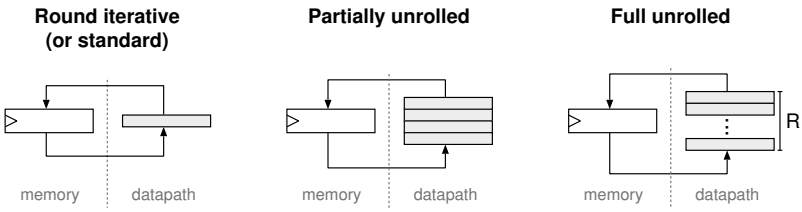

Figure 3.2: Design strategies for iterated hash functions. One grey module hosts the logic to compute one round of the compression function.

design). In a single cycle, two or more rounds are then executed (see Figure 3.2).

This solution leads to an increase in the circuit size and, at the same time, to a decrease of the maximum frequency. However, the principal benefits are possible logic optimizations during the synthesis process and the obvious reduction of the cycle count in the latency.

Figure 3.3 shows the area/speed trade-off generated by this design strategy. We synthesized five distinct architectures with different unrolling factors (1, 2, 4, 8, and 16) of a generic hash function that works with 16 rounds per compression. The graph on top illustrates the throughput increase for unrolled cores up to 60 % compared to the standard design (1 round). It is interesting that the optimum is reached with the 8-unrolled core. In the full unrolled design (16 rounds), the compiler is indeed not able to sufficiently optimize the deep datapath. The bottom graph clearly indicates that the area costs increase drastically. While the size of the memory part remains constant, the datapath size increases linearly. Furthermore, the dotted lines in the area vs. processing time graph identify equal-efficiency curves. The unrolled cores are faster but their hardware-efficiency is significantly lower.

In conclusion, depending on the system requirements, the suitable design should be applied. Generally, the standard iterative design is favored over unrolled designs for its higher hardware efficiency.

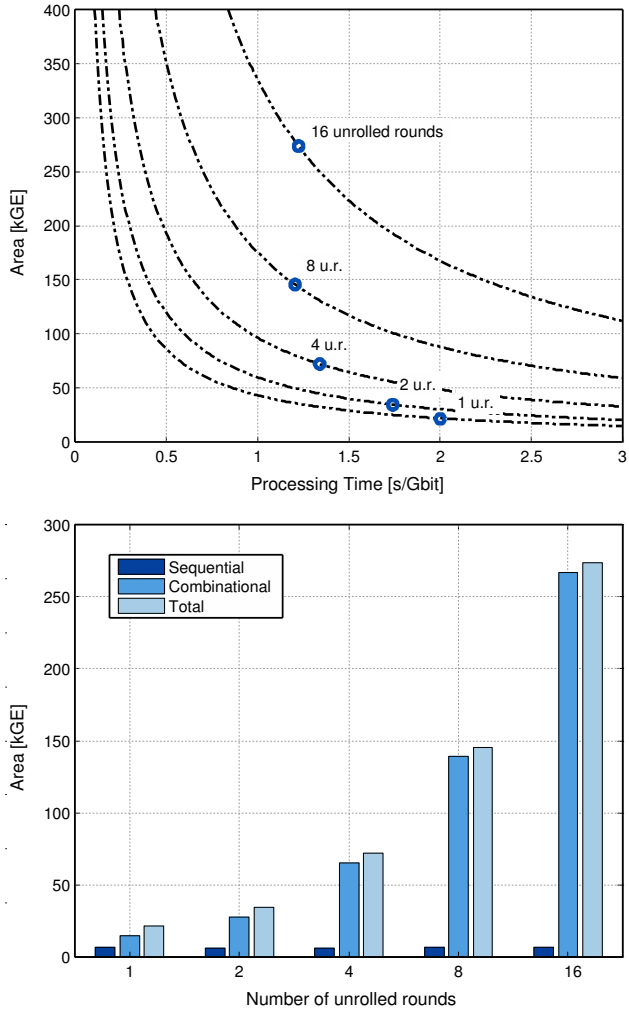

Figure 3.3: Area vs. processing time (top) and logic allocation (bottom) for different unrolling levels. The dotted lines indicate constant-efficiency area/speed curves.

However, if the silicon area is not critical, partially unrolled solutions can be exploited where speed becomes the foremost constraint.

3.1.4 Recent Hash Constructions

In the last decade, hash functions have been applied in countless protocols. In the mean time, their cryptanalysis has enhanced notably, and advanced tools to exploit the security of a function have been developed. This growing knowledge pointed out some security weaknesses in the classical MD construction. Improved hash constructions were thus proposed by the research community. We review some of the interesting models, which are used within this thesis, and briefly discuss their behavior in hardware.

Wide-Pipe

In the wide-pipe concept [28], the security of a n-bit hash function is improved by increasing the size of the internal state, i.e., $n_{\text{state}} > n$. Consequently, the hash function needs a finalization process that extracts n bits from the internal state to generate the hash-value. We speak of double-pipe when the state size is twice the hash-value size.

In hardware, the use of wide-pipe causes mainly an extended number of sequential cells required to store the internal state. In the case of double-pipe, if the state is split into two independent variables, pipeline interleaving becomes the most efficient approach to maximize the circuit efficiency. As example, Grøstl, a double-pipe hash function with independent state variables P and Q, increases its hardware-efficiency up to 50 % using internal pipeline-interleaved designs (see [29]).

Most modern hash functions feature a wide-pipe construction (e.g., ECHO, Fugue, Grøstl, etc.).

HAIFA

In contrast to the MD construction, the compression function of a HAIFA-based [30] hash function accepts two additional variables as input: the *counter* and the *salt*. The former contains the number of

bits hashed so far, while the latter is an optional input used for special applications such as *randomized hashing*[3].

The HAIFA model leads to similar VLSI performance as the MD construction. For comparison purposes, the salt is often omitted, since it is basically an added functionality of the algorithm. Instead, the counter is treated as an additional input fed to the circuit, rather than being computed internally.

New hash algorithms based on HAIFA are BLAKE and SHAvite-3.

Sponge

Instead of compressing the message with the internal state, a cryptographic hash sponge first "absorbs" the input block into the state and then applies a fixed-length permutation P. The sponge construction [33,34] has two distinct stages (see Figure 3.4): in the absorbing phase, the r-bit block is XORed with the last r bits of the state (the width of the sponge construction is the size of the internal state $r+c$), then the permutation P is applied; the squeezing phase returns r-bit of the state as hash-value block z_i interleaved with a P call.

In hardware, the sponge construction offers some relevant advantages over the classical model. Since the message block is directly XORed with the state, there is no need to store any bits of the message block inside the circuit, as it is often the case in the standard iterated design. In sponge-based functions, the message can indeed be combined with the state within a clock cycle without the need to store it. The major drawback is that the width of the sponge and, thus, the internal state are often larger in order to prevent preimage attacks (often $n_{\text{state}} > 2n$).

Two example of sponge-like hash functions are the Keccak algorithm and the QUARK family.

[3]Randomized hashing is a mode of operation for cryptographic hash functions intended for use with standard digital signatures [31]. The hash-value of the message is generated using an additional random value communicated with the signature (in the HAIFA construction this value is the salt). In this way, the signature scheme must no longer rely on a strong collision-resistant hash function [32].

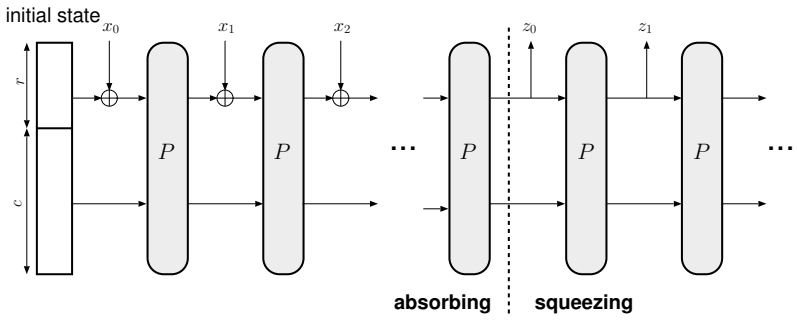

Figure 3.4: Hash procedure of a cryptographic sponge.

3.1.5 Block Cipher-based Constructions

A large class of iterated hash functions, especially used in the past, relies on block ciphers for the compression process. Under particular assumptions, the intermediate chaining variable h_{i-1} and the message block x_i can form the key-message pair of a block cipher. The choice to adapt a cipher as compression function is motivated by two aspects. First, an embedded system that provides confidentiality and data integrity can share the cipher implementation for encryption and hash. Second, the use of a trusted and well-established block cipher can give more confidence, if the security of the hash function is assumed by that of the cipher.

Among the block cipher constructions, the three main schemes are depicted in Figure 3.5.

Normally, the underlying block cipher E requires several rounds to process an incoming block. From Figure 3.5, we see that each scheme requires feedforward of at least one variable between x_i and h_{i-1}, hence increasing the storage demand in hardware, e.g., in the Davies-Meyer model, h_{i-1} mus be stored to be reused at the end of the encryption/compression rounds.

Currently, the worldwide recognized block cipher AES is the ideal candidate for block cipher-based hash functions. The hash function Whirlpool and its successor Maelstrom apply AES in the compression

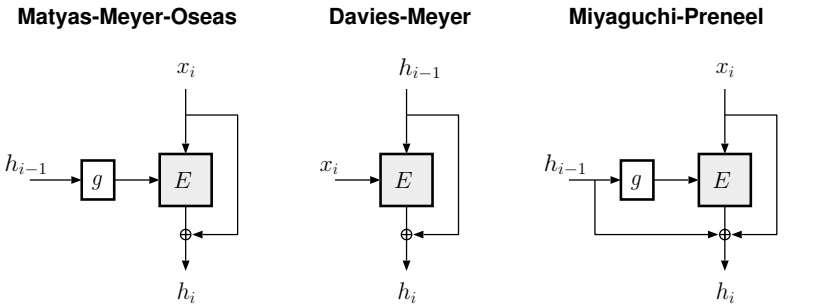

Figure 3.5: Three main schemes for block cipher-based hash. The g function maps the n-bit state to a k-bit key input, suitable for E.

function; the former uses the Miyaguchi-Preneel model, while the latter the Davies-Meyer. The widely deployed hash functions MD5, SHA-1, and SHA-2 do not build on a former block cipher, but adopt a compression scheme similar to the Davies-Meyer model.

3.2 The SHA-3 Competition

In 1991, Rivest presented MD5 (Message Digest algorithm 5) as 128-bit successor hash function of MD4. MD5 was the first cryptographic hash algorithm that was specifically designed for software applications and, due to its patent-freedom, it has been employed in numerous security protocols. After the discovery of non-fatal flaws in the design of MD5, NIST published in 1995 the second version of the Secure Hash Algorithm SHA-1, with a hash-value of 160 bits. Nowadays, MD5 and SHA-1 are still the most used hash functions.

In 2002, a new family of hash functions (for 224, 256, 384, and 512-bit outputs) under the name of SHA-2 [35] was advertised by NIST, as response to an increasing demand in security and the identification of mathematical weaknesses in the former standard SHA-1.

After the publication of important cryptanalytic attacks mining the collision resistance of MD5 and SHA-1 [36, 37], NIST was consequently forced to release in 2006 a public comment, which strongly advised a migration to the SHA-2 family. Following these attacks, Klima presented in [38] an optimized algorithm to find MD5 collisions within a minute. However, the most devastating (in terms of commercial impact) work has been presented at the 25th Chaos Communication Congress 2008, where the authors of [39] demonstrated a practical way to forge MD5-signed web certificates[4], hence "faking" the SSL certificate validity.

These results, combined with the structural similarity of MD5 with the SHA families, raised some concern on the security of the current hash standard SHA-2. Even if persuaded that practical attacks against SHA-2 are not imminent, NIST published in 2007 an open call for contributions to a public competition aiming at the development of a new standard for cryptographic hashing, which will be referred to as SHA-3 [40]. The proposed evaluation and selection process is similar to the competition that promoted the Rijndael block cipher as new AES in 2001. The cryptographic community was asked to propose new hash functions and to evaluate the security level of other candidates. In 2008, a total of 51 functions were accepted to the first round, while in July 2009, this number has been reduced to 14 second round candidates. The final decision, i.e., the proclamation of the winner algorithm, has been scheduled for 2012. To this end, the organizers are not only interested in the cryptographic strength of the candidates, but also in the evaluation of the performance of the algorithms implemented on different platforms. The new SHA-3 standard is indeed expected to provide at least the security of SHA-2 with significantly improved efficiency. Several applications, from multi-gigabit mass storage devices to RFID tags, are expected to utilize SHA-3. It is therefore crucial that the final SHA-3 function should be flexible enough to be used in both high-performance and resource-constrained environments. From a pure hardware point of view, the SHA-3 algorithm should provide good performance in terms of speed, area, and power.

[4]Certificate authorities (CAs) bind user identities with their public keys, and are thus a fundamental component in many public key infrastructure (PKI) schemes.

3.3 The Hash Function EnRUPT

The versatile cryptographic primitive EnRUPT has been presented by O'Neil [41] in 2008. EnRUPT is a simple and scalable word-based symmetric algorithm that allows to build efficient hash functions, stream and block ciphers. Our interest in EnRUPT was mainly due to its simplicity claim. According to the author, EnRUPT is indeed one of the simplest cryptographic primitives. Although top performance was not the principal goal, EnRUPT outperforms many block and stream ciphers on modern processors with significantly smaller code. Moreover, it was predictable that the intrinsic simplicity could be translated into low-cost straightforward circuits for hardware applications.

We started the analysis of EnRUPT with the goal to evaluate its efficiency when implemented in CMOS technologies. The motivation has been the not "well hidden" intent of the author to submit EnRUPT as SHA-3 candidate.

In 2007, the stream hashing mode of EnRUPT has been submitted and accepted in the first round of the NIST hash competition [42]. After the publication of the accepted candidates with the related algorithm specifications and implementation material, a method to find practical collisions for the seven EnRUPT variants (each with different hash-value length) has been developed and presented by Indesteege and Preneel [43, 44]. Due to this collision attack, the original submission of EnRUPT has rightly been considered broken and thus could not move to the second round of the competition.

In the next sections, we shortly give the specification of the EnRUPT hash function and present our contribution in the hardware analysis of the algorithm.

3.3.1 Specification of EnRUPT

The function has a set of parameters that identifies the different EnRUPT variants. The crucial parameter s defines the security robustness of the function (number of iterations). In the official submission document, the lower bound for general-purpose security was $s = 4$. With this condition, the authors were convinced to ensure a sufficiently high margin against all attack types. However, after

Table 3.1: Parameters of the original ($s = 4$) and tweaked ($s = 8$) EnRUPT-n variants.

s		P	H	rounds	state size [bits]
4	EnRUPT-256	1	8	8	512 + 64
	EnRUPT-512	1	16	8	1024 + 64
	EnRUPT-256	2	8	8	512 + 128
	EnRUPT-512	2	16	8	1024 + 128
	EnRUPT-256	4	8	8	512 + 256
	EnRUPT-512	4	16	8	1024 + 256
8	EnRUPT-256	1	8	16	512 + 64
	EnRUPT-512	1	16	16	1024 + 64
	EnRUPT-256	2	8	16	512 + 128
	EnRUPT-512	2	16	16	1024 + 128
	EnRUPT-256	4	8	16	512 + 256
	EnRUPT-512	4	16	16	1024 + 256

the aforementioned collision attacks, the minimal value for s has been increased to eight (i.e., twice the security of the previous specification). Despite this correction, the strengthened version of EnRUPT was not sufficiently convincing to be selected for the second round.

Also important are the internal state size H and the parallelization degree P. These parameters identify the number of w-bit words of the internal state x and the number of "delta accumulators" d_i, respectively. In Table 3.1, the parameters for the principal variants of EnRUPT are given.

The stream hashing mode of EnRUPT, formally referred to as ïrRUPT, is similar to the sponge construction. The message is absorbed into part of the combined state (x, d), and then a permutation is applied. In contrast to the sponge model, three (instead of two) phases are scheduled. Analyzing Algorithm 1, once the message is entirely absorbed (message processing), a finalization stage seals the internal state, before the function squeezes the hash-value words

Algorithm 1 EnRUPT stream hashing mode.

function EnRUPT(M)
set $m_0, \ldots, m_{l/w} \leftarrow M \| 1 \| 0^{w-((|M|+1) \bmod w)}$
  $x_0, \ldots, x_{H-1}, d_0, \ldots, d_{P-1}, r \leftarrow 0$
for $i = 0, \ldots, l/w$ **do**
  $(x, d, r) \leftarrow \text{round}(x, d, r, m_i)$ // Message processing
end for
$(x, d, r) \leftarrow \text{round}(x, d, r, n)$
for $i = 0, \ldots, H - 1/$ **do**
  $(x, d, r) \leftarrow \text{round}(x, d, r, 0)$ // Finalization
end for
for $i = 0, \ldots, n/w - 1$ **do**
  $(x, d, r) \leftarrow \text{round}(x, d, r, 0)$ // Output
  $y_i \leftarrow d_{P-1}$
end for
return $y_0 \| \cdots \| y_{n/w-1}$

(output). The permutation is performed by the round transformation. This function executes $2s$ rounds and updates x, d_i, and r as described in Algorithm 2.

A single iteration of the round function is also called ïr1. It is a combination of basic transformations (addition, XOR, rotation, and shift) executed on w-bit words. To achieve the best software efficiency, the size of the word w for the main variants listed in Table 3.1 has been fixed to 64 bits. In ïr1, an intermediate variable e is first computed

Algorithm 2 EnRUPT round function.

function round(x, d, r, m)
for $i = 0, \ldots, 2s - 1$ **do**
  $e \leftarrow (2x_{\lfloor r/P \rfloor P+((r+1) \bmod P)} \oplus x_{r+2P} \oplus d_{r \bmod P} \oplus r) \ggg w/4$
  $f \leftarrow (f \ll 3) + f$ // Multiplication with 9
  $x_{r+P} \leftarrow x_{r+P} \oplus f$
  $d_{r \bmod P} \leftarrow d_{r \bmod P} \oplus f \oplus x_{r+HP/2+P \wedge 1}$
  $r \leftarrow r + 1$
end for
$d_{P-1} \leftarrow d_{P-1} \oplus m$
return (x, d, r)

by XORing a combination of x and d_i words; the result e is then multiplied by nine to form f (addition of e with a shifted-by-3 copy of itself, cf. line 3 in Algorithm 2). Finally, one word of the state and one word within the accumulators are updated with f, while the round variable r is incremented. Only at the end of the $2s$ iterations of ïr1, the input message is absorbed into the delta accumulators.

3.3.2 EnRUPT Architectures

The first version of the EnRUPT stream hashing mode was proposed in [41]. After the public call for new hash functions in 2007, the author declared publicly the intention to submit EnRUPT to the NIST competition. The first version of the algorithm was specified to work exclusively with $P = 1$ and $s = 4$. This means that any parallelization attempt of the round function was possible. The parameters P was even not defined. We started therefore our analysis on this first version. The results have been presented in [45]; this work analyzes the performance in hardware of the stream hash construction applied in Algorithm 1.

In the first specification ($P = 1$, $s = 4$), the flow dependencies between the eight calls of the round function inhibited every parallelization effort, so the speed performance was considerably limited. Nevertheless, EnRUPT showed promising results especially in the low-area domain. The plain design, relying on a single modulo adder and few other gates, fitted in less than 6 kGE (see results presented in [45]). We were thus convinced that EnRUPT could be a front runner in the competition after increasing its parallelization capability.

After a careful evaluation of security and efficiency, the authors decided to introduce the parallelization parameter P, hence proposing three variants of EnRUPT characterized by different parallelization degrees $P \in \{1, 2, 4\}$. To avoid confusion, ïrRUPT with $P = 2$ was still favored, as it targets general-purpose hashing and offers optimal balance between constrained environments and large micro chips. In the hardware analysis, we concentrated our efforts on two design methodologies (see Figure 3.6):

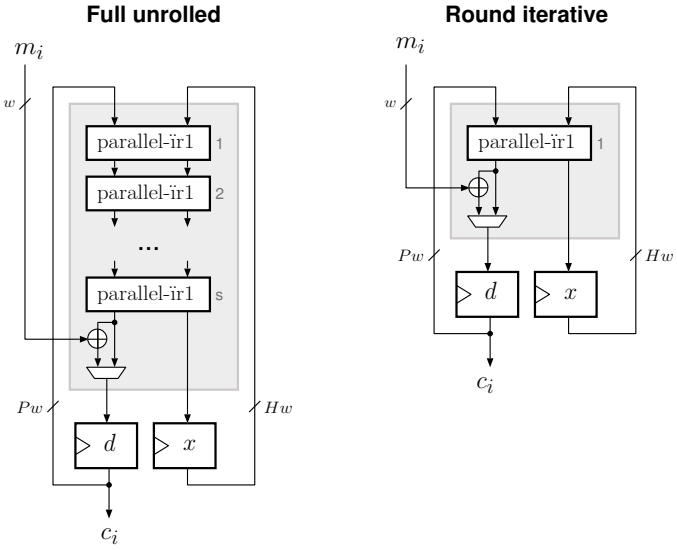

Figure 3.6: Block diagrams of the Ps-ïrRUPT (left) and P-ïrRUPT (right) architectures.

Full unrolled The first investigated architecture, i.e., Ps-ïrRUPT, is a sort of isomorphic hardware migration of the EnRUPT permutation, where a complete round function is executed within a cycle.

Due to the introduction of parallelization, in total P iterations of Algorithm 2 can be computed in parallel within the parallel-ïr1 module. As can be seen in the left design of Figure 3.6, the datapath consists of s parallel-ïr1 units. In addition, a register-based memory stores the updated x state and the delta accumulator d_i at the end of the cycle.

The aim of the Ps-ïrRUPT architecture is to achieve a high throughput, by maintaining the computational depth of the algorithm.

Round iterative The second architecture is based on the iterative decomposition of the round transformation. Only one parallel-ïr1 unit has been implemented in combination with the memory. In the P-ïrRUPT core, a total of s clock cycles are needed to complete a

round function call.

In both architectures, the parallel-ïr1 module consists in a progressive word-shift of the state x, combined with P ïr1 blocks. Using word-shift registers, the ïr1 modules take as inputs always the same x words. Consequently, every word is shifted by P positions each cycle, according to the algorithm. This shift operation is equivalent to the increment of the index variable r.

We deem the listing of the performance for each variant presented in Table 3.1 beyond the scope of this thesis. A comparison in 0.18 µm CMOS technology of the $(P = 2, s = 8)$-architectures with other SHA-3 second round candidates is proposed in Section 3.4.2. Furthermore, the detailed design space exploration for ASIC and FPGA devices is given in [23] and in the official submission document of EnRUPT [42].

3.4 The Hash Function BLAKE

After the publication of the hash function LAKE [46] in 2008, we were asked by the authors to evaluate the hardware efficiency of their newest algorithm. We then published a comparative analysis [25] between hash functions based on a sponge-like construction, on the MD construction with S-boxes, and on the MD construction with only additions, XORs, and rotations. This work clearly showed some deficiencies in the third hash model, which LAKE uses, compared to the first two. In order to fully analyze and improve the hardware efficiency, we participated in the development of the new/successor hash function, called BLAKE, which we submitted to the NIST hash competition. At the end of 2008, BLAKE was accepted as first round candidate.

BLAKE has not been designed to be the "best" algorithm in one particular domain as security, speed, or simplicity. The goal was rather to propose a versatile hash function that behaves well in every situation and with respect to all evaluation criteria, as will probably do the final SHA-3.

The three components that characterize BLAKE are:

- The HAIFA iteration mode. In this way, randomized hashing can benefit from the salt variable.

- The internal structure that doubles the size of the state with respect to the intermediate chaining variable, a sort of "local wide-pipe".

- The compression function based on the stream cipher ChaCha[5].

BLAKE relies therefore on previously studied structures, rather than on new design ideas.

In 2009, BLAKE has advanced to the second stage of the competition and was thus selected as one of the 14 second round candidates. So far, none of the published cryptanalyses mines the security claims of BLAKE.

In the next sections, after the specification of the algorithm, we explain how to maximize the throughput in VLSI architectures. To this end, a comparison with competitor designs of all second round candidates is reported. The architectures proposed here and in [50] improve the work done in the submission document [51]. We extend the VLSI analysis by including the design and test process of a manufactured ASIC, hosting a compact low-power 0.18 µm CMOS BLAKE implementation.

3.4.1 Specification of BLAKE

BLAKE has two main versions: BLAKE-32 and BLAKE-64. We first start with a brief specification of BLAKE-32. In fact, BLAKE-64 is similar with few modifications. A complete specification can be found in the official submission document [51].

BLAKE-32

The BLAKE-32 algorithm operates on 32-bit words and returns a 256-bit hash-value. It is based on the iteration of the compression function that takes four values as input:

- a chaining value $h = h_0, \ldots, h_7$.

- a message block $m = m_0, \ldots, m_{15}$.

[5]ChaCha [47] is the successor stream cipher of Salsa20 [48, 49], which is one of the selected algorithm for the final software portfolio of the eSTREAM Project.

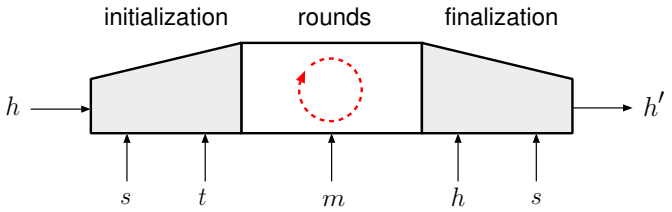

Figure 3.7: The local wide-pipe construction of BLAKE's compression function.

- a salt $s = s_0, \ldots, s_3$.
- a counter $t = t_0, t_1$.

These inputs represent 30 words in total (i.e., 960 bits). The output of the compression function is a new chaining value $h' = h'_0, \ldots, h'_7$ of eight words (i.e., 256 bits). We write the compression of h, m, s, t to h' as

$$h' := F(h, m, s, t). \tag{3.4}$$

F can be further decomposed into three main steps (see Figure 3.7):

Initialization A 16-word internal state $v_0, \ldots, v_{15}$ is initialized such that different inputs produce different initial states. This state is represented as a 4×4 matrix and filled as follows

$$\begin{pmatrix} v_0 & v_1 & v_2 & v_3 \\ v_4 & v_5 & v_6 & v_7 \\ v_8 & v_9 & v_{10} & v_{11} \\ v_{12} & v_{13} & v_{14} & v_{15} \end{pmatrix} \leftarrow \begin{pmatrix} h_0 & h_1 & h_2 & h_3 \\ h_4 & h_5 & h_6 & h_7 \\ s_0 \oplus c_0 & s_1 \oplus c_1 & s_2 \oplus c_2 & s_3 \oplus c_3 \\ t_0 \oplus c_4 & t_0 \oplus c_5 & t_1 \oplus c_6 & t_1 \oplus c_7 \end{pmatrix} \tag{3.5}$$

where $c_0, \ldots, c_{15}$ are predefined word constants.

Round Function Once the state v is initialized, the compression function iterates a series of ten rounds. A round is a transformation of the state that computes

$$\begin{array}{ll} \mathsf{G}_0(v_0, v_4, v_8, v_{12}) & \mathsf{G}_1(v_1, v_5, v_9, v_{13}) \\ \mathsf{G}_2(v_2, v_6, v_{10}, v_{14}) & \mathsf{G}_3(v_3, v_7, v_{11}, v_{15}) \end{array} \tag{3.6}$$

and then

$$\begin{array}{ll} \mathsf{G}_4(v_0\ ,v_5\ ,v_{10},v_{15}) & \mathsf{G}_5(v_1\ ,v_6\ ,v_{11},v_{12}) \\ \mathsf{G}_6(v_2\ ,v_7\ ,v_8\ ,v_{13}) & \mathsf{G}_7(v_3\ ,v_4\ ,v_9\ ,v_{14}) \end{array} \tag{3.7}$$

where, at round r, $\mathsf{G}_i(a,b,c,d)$ sets

$$\begin{array}{rcl} a & \leftarrow & a+b+(m_{\sigma_r(2i)} \oplus c_{\sigma_r(2i+1)}) \\ d & \leftarrow & (d \oplus a) \ggg 16 \\ c & \leftarrow & c+d \\ b & \leftarrow & (b \oplus c) \ggg 12 \\ a & \leftarrow & a+b+(m_{\sigma_r(2i+1)} \oplus c_{\sigma_r(2i)}) \\ d & \leftarrow & (d \oplus a) \ggg 8 \\ c & \leftarrow & c+d \\ b & \leftarrow & (b \oplus c) \ggg 7 \end{array} \tag{3.8}$$

The G function[6] uses ten permutations of $\{0,\ldots,15\}$, written $\sigma_0,\ldots,\sigma_9$, which are fixed by the design. G also uses the constants $c_0,\ldots,c_{15}$.

Note that the first four calls $\mathsf{G}_0,\ldots,\mathsf{G}_3$ in (3.6) can be computed in parallel, because each updates a distinct column of the state. The sequence $\mathsf{G}_0,\ldots,\mathsf{G}_3$ is called a *column step*. Similarly, the last four calls $\mathsf{G}_4,\ldots,\mathsf{G}_7$ in (3.7) update distinct diagonals and are called a *diagonal step* (see Figure 3.8).

Finalization After the sequence of rounds, the new chaining value h' is extracted from the state $v_0,\ldots,v_{15}$ with input of the initial chaining value h and the salt s:

$$\begin{array}{rcl} h'_0 & \leftarrow & h_0 \oplus s_0 \oplus v_0 \oplus v_8 \\ h'_1 & \leftarrow & h_1 \oplus s_1 \oplus v_1 \oplus v_9 \\ h'_2 & \leftarrow & h_2 \oplus s_2 \oplus v_2 \oplus v_{10} \\ h'_3 & \leftarrow & h_3 \oplus s_3 \oplus v_3 \oplus v_{11} \\ h'_4 & \leftarrow & h_4 \oplus s_0 \oplus v_4 \oplus v_{12} \\ h'_5 & \leftarrow & h_5 \oplus s_1 \oplus v_5 \oplus v_{13} \\ h'_6 & \leftarrow & h_6 \oplus s_2 \oplus v_6 \oplus v_{14} \\ h'_7 & \leftarrow & h_7 \oplus s_3 \oplus v_7 \oplus v_{15} \end{array} \tag{3.9}$$

[6] In the following, for statements that do not depend on the index i we shall omit the subscript and write simply G.

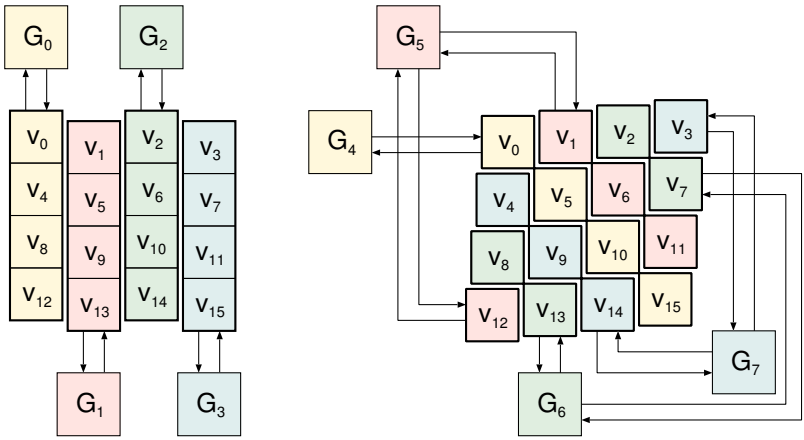

Figure 3.8: BLAKE's column step and diagonal step.

BLAKE-64

BLAKE-64 operates on 64-bit words and returns a 512-bit hash-value. All lengths of variables are doubled compared to BLAKE-32: chaining values are 512-bit, message blocks are 1024-bit, salt is 256-bit, counter is 128-bit.

The compression function of BLAKE-64 is similar to that of BLAKE-32, except that it makes 14 rounds instead of ten, and that $\mathsf{G}_i(a, b, c, d)$ computes

$$\begin{aligned} a &\leftarrow a + b + (m_{\sigma_r(2i)} \oplus c_{\sigma_r(2i+1)}) \\ d &\leftarrow (d \oplus a) \ggg 32 \\ c &\leftarrow c + d \\ b &\leftarrow (b \oplus c) \ggg 25 \\ a &\leftarrow a + b + (m_{\sigma_r(2i+1)} \oplus c_{\sigma_r(2i)}) \\ d &\leftarrow (d \oplus a) \ggg 16 \\ c &\leftarrow c + d \\ b &\leftarrow (b \oplus c) \ggg 11 \end{aligned} \tag{3.10}$$

After ten rounds, the round function uses the permutations $\sigma_0, \ldots, \sigma_3$ for the last four rounds.

3.4.2 High-Speed Architectures

In this section we investigate high-speed implementations of BLAKE, with an iterative decomposition of the round process.

Different architectures are made possible by varying the number of integrated G modules. Modern high-speed communication systems for which space is not a fierce constraint can take advantage of architectures with eight G modules or even with a full unrolled circuit. Opposite, by scaling the number of G modules, the design becomes slower but decreases in size (see design proposals of [51] or Figure 3.3).

Besides the round computation, BLAKE requires some circuitry to perform initialization and finalization; for instance, 32 w-bit XORs are required to compute (3.5) and (3.9), where $w = 32$ for BLAKE-32 and $w = 64$ for BLAKE-64. Furthermore, the complete execution of initialization and finalization can be performed in the same clock cycle, when the new message block is available. Like most hash functions, BLAKE uses some constant values, which are

- the initial value IV_i (eight w-bit words);
- the 16 round constants c_i;
- the ten permutations σ_i (in total 640 bits).

These values are used mainly by the G function; the best solution is to hard-code them without using special macro blocks for storage. Since BLAKE iterates a series of rounds over an internal state, additional sequential components are required to store the following 44 values:

- the 8-word chaining value h;
- the 16-word internal state v;
- the 4-word salt value s;
- the 16-word message block m.

The two words of the counter t need not be stored. In high-speed architectures, the initialization process (the only phase where the counter is used) is indeed executed in a single clock cycle. Moreover, we decided to take the counter as external input, together with the message block. This choice is motivated by the fact that the counter during the last call of the compression function "knows" the number of padded bits inside the last message block. It is thus natural to treat it like a normal input. The sequential logic is thus made up by $44 \times w$ registers (i.e., 1408 for BLAKE-32, 2816 for BLAKE-64) plus some additional registers for the control unit.

To exploit the full parallelizability of BLAKE, two types of designs have been coded in VHDL. Referring to [52, 51], the first is called [8G], which corresponds to a straightforward round-iterative implementation with eight G modules computing the column and diagonal steps; and the second, called [4G], where only four parallel G modules concurrently compute the two steps. Outside the round module, the sequential part (register memories), and the components for initialization and finalization, we added a control unit, based on a simple finite-state machine, which computes the round increment and starts/terminates the hashing process. Figure 3.9 shows a block diagram of the [8G]- and [4G]-BLAKE cores. During the round iteration, only the state memory and the [8G], respectively [4G], module are mainly involved.

Round Rescheduling

The G function of BLAKE is a modified version of the core function of the stream cipher ChaCha [47]. Speed limits for plain designs implementing several architectures of ChaCha have been reported in [52]. The introduction of the addition with the message/constant (MC) -pair in the G function leads to an increment of the propagation delay. If in the core function (similar to G) the maximum delay is given by the total delay of four XORs and four modular adders (rotation is a simple re-routing of the word without effective propagation delay), the slightly modified G function inserts an addition with the MC-pair. Accordingly, the maximum frequency of analogous BLAKE architectures (cf. [51]) is slightly lower than those obtained for the stream cipher ChaCha. However, with a rescheduling of the G computation, it is

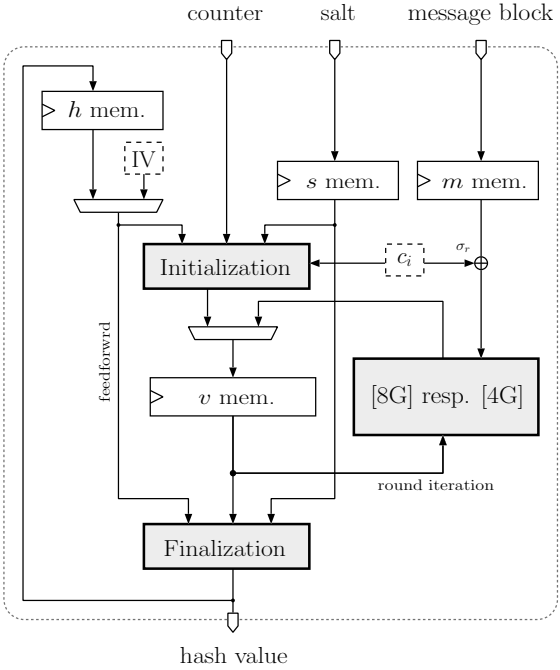

Figure 3.9: The main architecture of the [8G]- and [4G]-BLAKE cores.

possible to recover the original longest path of ChaCha (four XORs and four adders), hence decreasing the overall propagation delay of the core function. Observing the flow dependencies in (3.8), it is clear that the addition with the MC-pair is independent (message word and constant are unrelated to the state v) and can be computed in parallel to the other computations. If in a single call of G, similarly to the core function of ChaCha, each update of the state has been conceived to operate sequentially, the MC-pair addition can be shifted within the

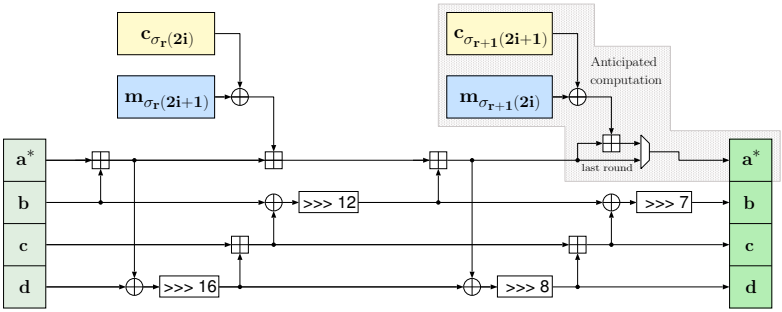

Figure 3.10: Block diagram of the rescheduled G function. Note: the round index of the second message/constant pair is increased by one.

computations. It is thus possible to anticipate it, reducing the critical path of G. The rescheduled $\mathsf{G}_i(a^*, b, c, d)$ computes

$$\begin{aligned} a &\leftarrow a^* + b \\ d &\leftarrow (d \oplus a) \ggg r_0 \\ c &\leftarrow c + d \\ b &\leftarrow (b \oplus c) \ggg r_1 \\ a &\leftarrow a + b + (m_{\sigma_r(2i+1)} \oplus c_{\sigma_r(2i)}) \\ d &\leftarrow (d \oplus a) \ggg r_2 \\ c &\leftarrow c + d \\ b &\leftarrow (b \oplus c) \ggg r_3 \\ a* &\leftarrow a + (m_{\sigma_{r+1}(2i)} \oplus c_{\sigma_{r+1}(2i+1)}) \end{aligned} \tag{3.11}$$

where r_i are the rotation indices for BLAKE-32 and BLAKE-64, and a^* corresponds to the modified first input/output variable after the MC addition. Figure 3.10 shows the block diagram of the modified G function. To keep the correct functional behavior, a 2-input MUX must be inserted before the sequential logic, hence allowing the record of a instead of a^* in the last round. This is the main reason why the rescheduling optimization can not be carried out automatically by the synthesis tool and must be instantiated at code level.

Performance Analysis

To evaluate the speed-up provided by the G rescheduling, we coded the [8G] and [4G] architectures in VHDL and we synthesized them for BLAKE-32 and BLAKE-64 with the Synopsys Compiler. Our results refer to fully autonomous designs, which take as input salt, counter, and message blocks and generate the final hash value. Moreover, to obtain an exhaustive analysis of the BLAKE hash cores, the designs have been synthesized in three different UMC technologies: 0.18 µm, 0.13 µm, and 90 nm.

Tables 3.2 to 3.4 present a detailed performance comparison with the current standard SHA-2, and with other second round candidates in the NIST hash competition for which performance figures were available. We also added the performance of the EnRUPT cores for $s = 8$ and $P = 2$, presented in Section 3.3.2. Each entry refers to a post-synthesis implementation, and the last column reports the hardware efficiency. Only for 0.18 µm we were able to provide a full comparison of the 14 candidates. This was possible thanks to the results provided in [53]. Figure 3.11 illustrates the trade-off between area and processing time for the 256-bit versions of the candidate functions plus SHA-256 and EnRUPT-256. Note that our designs for BLAKE-32 support salted hashing which is not the case in [53]. Although very similar, the design presented in [53] inserts a pipeline register at the output of the permutation m_{σ_r}. This technique has an effect analogous to the round rescheduling, i.e., reduction of the critical path, adding however an additional 512-bit register bank.

Compared to the architectures presented in [51], we obtain a 20 % speed-up, due to the delay reduction of the round rescheduling process described in the previous section. However, we must take into account an area increase caused by the integration of the register-based memories for message block, chaining value, and salt; note that the previous designs of [51] represent only the compression function.

Comparing the proposed BLAKE cores with the SHA-2 family, we observe a substantial throughput gain. This improvement comes at the cost of an area increase, which can also be a side effect of the alleged security improvement. Comparing with the other candidates, BLAKE is faster than about half of them. If we take into account that the function Blue Midnight Wish requires a large area to achieve the

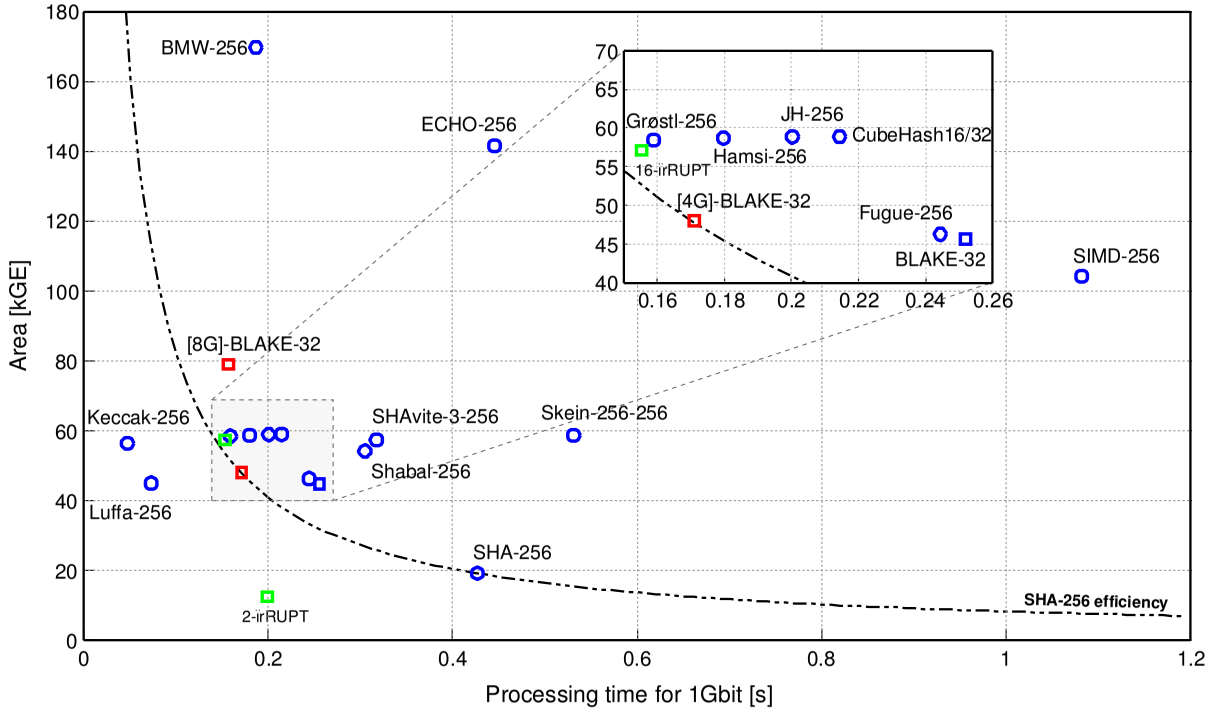

Figure 3.11: Processing time for 1 Gbit of data versus total area of the 14 second round candidates and EnRUPT (0.18 µm CMOS technology). The blue points refer to the implementations of [53]. The cores presented here are in red (BLAKE-32) and green (EnRUPT).

Table 3.2: Performance comparison for a 0.18 µm CMOS technology.

Algorithm	**Area** [kGE]	**Cycles**	**Freq.** [MHz]	**Thr.** [Gbps]	**HW-Eff.** [kbps/GE]
[4G]-BLAKE-32	48	21	240	5.847	123
[4G]-BLAKE-64	98	29	204	7.192	74
[8G]-BLAKE-32	79	11	137	6.376	81
[8G]-BLAKE-64	147	15	106	7.216	49
16-ïrRUPT [23]	58	1	99	6.305	109
2-ïrRUPT [23]	13	8	621	4.969	390
BLAKE-32[a] [53]	46	22	171	3.971	87
BMW-256 [53]	170	1	10	5.385	32
CH16/32(-256)[b] [53]	59	8	146	4.665	79
ECHO-256 [53]	141	97	142	2.246	16
Fugue-256 [53]	46	2	256	4.092	88
Grøstl-256 [53]	58	22	270	6.290	108
Grøstl-512 [54]	340	14	85	6.225	18
Hamsi-256 [53]	59	1	174	5.565	95
JH-256 [53]	59	39	380	4.992	85
Keccak(-256) [53]	56	25	488	21.229	377
Luffa-256 [53]	45	9	483	13.741	306
Shabal-256 [53]	54	50	321	3.282	61
SHAvite-3-256 [53]	57	37	228	3.152	55
SIMD-256 [53]	104	36	65	0.924	9
Skein-256-256 [53]	59	10	74	1.882	32
Skein-512-512 [53]	102	10	49	2.205	22
SHA-256 [53]	19	66	302	2.344	122
SHA-512 [55]	31	88	169	1.969	64

[a] Salt support is omitted.
[b] We refer to the CubeHash candidate.

same speed, we can assert that our architectures improve the results of [53], outperforming in efficiency a set of four (second round) candidate algorithms with similar throughput performances, i.e., Grøstl, Hamsi, JH, and CubeHash (see Figure 3.11). With the application of the round rescheduling, we can indeed increase the hardware efficiency up to the value achieved by SHA-256 in 0.18 µm.

Table 3.3: Performance comparison for a 0.13 µm CMOS technology.

Algorithm	**Area** [kGE]	**Cycles**	**Freq.** [MHz]	**Thr.** [Gbps]	**HW-Eff.** [kbps/GE]
[4G]-BLAKE-32	43	21	330	8.047	187
[4G]-BLAKE-64	92	29	291	10.265	111
[8G]-BLAKE-32	67	11	201	9.365	140
[8G]-BLAKE-64	139	15	158	10.802	78
CH16/32 [56]	34	16	578	9.248	269
ECHO-256 [57]	521	9	87	14.850	29
ECHO-512 [57]	517	11	83	7.750	15
Hamsi-256 [58]	22	7	1 080	4.937	224
Hamsi-512 [58]	50	13	820	4.036	81
Keccak [59]	48	18	526	29.900	623
Luffa-256 [60]	27	9	444	12.642	471
Luffa-512 [60]	44	8	444	12.642	286
Shabal [56]	41	52	645	6.351	154
SHA-256 [61]	22	68	794	5.975	271
SHA-512 [61]	43	84	746	9.096	210

Table 3.4: Performance comparison for a 90 nm CMOS technology.

Algorithm	**Area** [kGE]	**Cycles**	**Freq.** [MHz]	**Thr.** [Gbps]	**HW-Eff.** [kbps/GE]
[4G]-BLAKE-32	38	21	621	15.143	396
[4G]-BLAKE-64	79	29	532	18.782	237
[8G]-BLAKE-32	65	11	376	17.498	269
[8G]-BLAKE-64	128	15	298	20.317	158
Fugue-256 [62]	110	2	870	13.913	127

The functions Keccak and Luffa outperform every candidate in maximum achievable speed, requiring, at the same time, limited-area hardware. This mainly follows from their sole use of Boolean operators, rather than modular additions. Note however, that such optimization for hardware comes at a price of low software performance (where the function cannot benefit of CPU arithmetic instructions).

Moreover, previous cryptanalysis results suggest that such designs may have structural flaws [63, 64].

A final note on EnRUPT suggests that the round iterative architecture 2-ïrRUPT achieves the best results in hardware efficiency with a suitable throughput of 6.3 Gbps in 0.18 µm. The size of the circuit of only 13 kGE points out the extremely high flexibility of this function even with improved security $s = 8$. Due to the simplicity of the EnRUPT round, also the full unrolled design 16-ïrRUPT compares well against other second round candidates. Nevertheless, these merits have not been a sufficient argument against the affected reliability of EnRUPT's security strength.

3.4.3 Silicon Implementation of a Compact BLAKE-32 Core

We designed a compact architecture of BLAKE-32 in order to satisfy the stringent restrictions of resource-constrained environments. Besides an area reduction, the cryptographic core must also keep the energy dissipation at minimal values. Following these two design principles, we concentrated our efforts in the reduction of the round circuit and in the implementation of efficient memory modules.

As previously noted, BLAKE relies on eight calls of the G function within the column and diagonal steps. Inside the G function, the computation that requires most of the area resources is the modular addition. Instead of implementing four G modules with six independent 32-bit adders, we opted for a single adder, where the G function is iteratively decomposed in ten steps. This causes an increase of the *per-message block* processing time, but contributes to a small overall size. Figure 3.12 shows the block diagram of the proposed compact architecture. For the G computation, two 32-bit XOR gates and a rotation selector (r_i defines the different rotation numbers) are implemented in conjunction with the 32-bit adder (cf. ② in Figure 3.12). Each variable required by the hashing process is stored in optimized two-port memories. In total, five memory elements are needed, while an intermediate 32-bit register allows the extraction of temporary state words. This architecture leads to a total latency in clock cycles of 816 for a 512-bit message block. In addition to the $10\times$ 8×10 cycles to complete the round function, 16 cycles are indeed needed for the

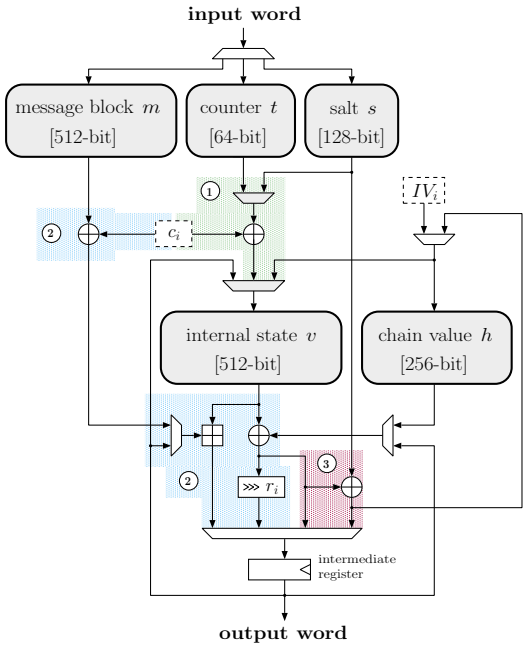

Figure 3.12: Block diagram of the implemented lightweight BLAKE-32. All connections are 32-bit wide.

initialization process. Moreover, the initialization is started while the update of the chaining value (finalization) is still ongoing. Here, after sorting the selected state word v_i (required for the h' computation, cf. ③ in Figure 3.12), the free memory slot is filled with the new chaining value or with the result coming from ①, respectively.

The output of the architecture depicted in Figure 3.12 is the 32-bit value stored in the intermediate register, while the input is a 32-bit word which is consequently routed to the memories for message block, salt, or block counter.

Memory Architecture

The VLSI implementation of BLAKE-32 needs memory to store 16 words of internal state and eight words of chaining value, plus additional registers to store the salt (four words), the counter (two words), and the message block (16 words), i.e., in total 1472 bits of memory. The counter is used during four clock cycles and needs thus to be stored. Compared to the minimum circuit needed to implement the compression function (initialization, rounds, and finalization), the memory units are the main contribution in terms of area and energy consumption. It is thus of primary interest to design special-purpose register elements in order to decrease the global resource requirements of the hash core. We introduced in the compact architecture of BLAKE-32 semi-custom memories based on clock-gated latch arrays, able to store at most one word per cycle. In general, depending on the word number of the target value to be stored, these memories replace the standard flip-flop gates by latch gates. The latches are organized in 32-bit banks, so that each bank stores a single word and is triggered by a dedicated gated clock [24].

Our example architecture in Figure 3.13 depicts a 4-word latch array to store the salt value. In the address decoder, the different one-hot enable signals are activated, depending on the write address. To prevent timing loops inside the logic, caused by the transparent behavior of latches, an input flip-flop bank is added. This bank is in turn driven by a gated clock generated by the write enable signal, while the outputs of the flip-flops are connected to the inputs of all latch banks. When a write enable occurs, the input word is first stored inside the flip-flop bank and subsequently passed to the activated latch bank.

To keep the functionality similar to a normal flip-flop-based memory, the outputs of the latch array are collected by a large multiplexer driven by the read address signal. This allows an instantaneous response of the memory.

An area comparison of the proposed latch-array and standard flip-flop-based memories is shown in Table 3.5. By decreasing the number of words, the two memories get closer in size. However, we still achieve a slight area saving, even with the smallest used memory size (counter). As can be seen in Figure 3.12, the compact BLAKE-32

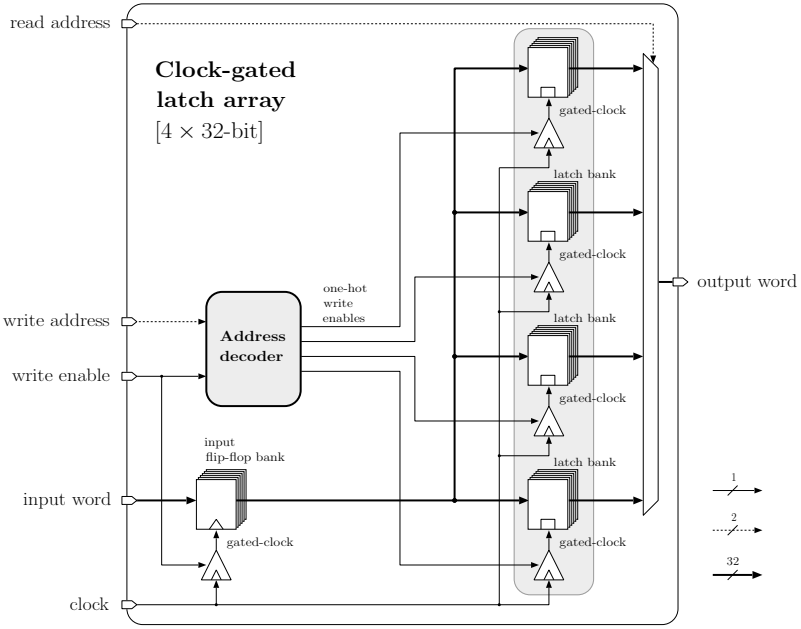

Figure 3.13: Overview of a 4-word memory unit implemented as clock-gated latch array.

architecture works with five memories. This leads to an overall area reduction of about 34 % for the memory components and 28 % for the complete design. In order to compare the power consumption, we designed the layout of two equal BLAKE-32 cores, using different memory strategies. With the aid of post place and route power simulations, a 60 % reduction in the mean energy dissipation of the five latch-based memory components has been measured, leading to a global energy decrease of 36 % for the complete design.

Memory access times for write and read operations have further been analyzed. Due to the flip-flop bank at the input, the write time is kept similar to a standard flip-flop-based memory. At the output, the read time is only affected by the size of the multiplexer, i.e.,

Table 3.5: Size comparison in GE (1.0 GE is 9.3744 µm^2) of the memory elements in the 0.18 µm UMC technology (post layout results for 200 MHz).

Word number	2	4	8	16
Standard FF	585	1234	2370	5434
Latch array	550	926	1681	3376
Area gain [%]	6	25	29	39

the number of words. Also here, the timing is equivalent in both architectures.

ASIC Constraints

The compact BLAKE-32 architecture was coded in VHDL and synthesized with the Synopsys Compiler using the UMC 1P/6M 0.18 µm technology. The Cadence SoC Encounter System has then been used to place and route the final layout of the ASIC. The chip layout and the die photo are presented in Figure 3.14. The BLAKE-32 core fills 0.127 mm^2, which is only a small fraction of the total chip (size 1.565 mm×1.565 mm, i.e., 2.450 mm^2). Table 3.6 gives an overview of the area partitioning of the hash core. In the remaining space, additional projects unrelated from BLAKE have been integrated.

Measurements and Performance Comparison

To test the correct functional behavior, the fabricated chip has been stimulated using a HP83000-F660 digital tester under different setups and stimuli vectors. The characteristic period vs. supply voltage shmoo plot is presented in Figure 3.15. The evident aspect is that the maximum working frequency strongly depends on the supply voltage.

To reach 200 MHz, the chip must be supplied with the technology's nominal voltage of 1.8 V. With these parameters, post layout power simulations have been performed, in order to evaluate the individual energy contributions of the chip components (cf. last column of Table 3.6). Memory modules, sparsely used during the compression

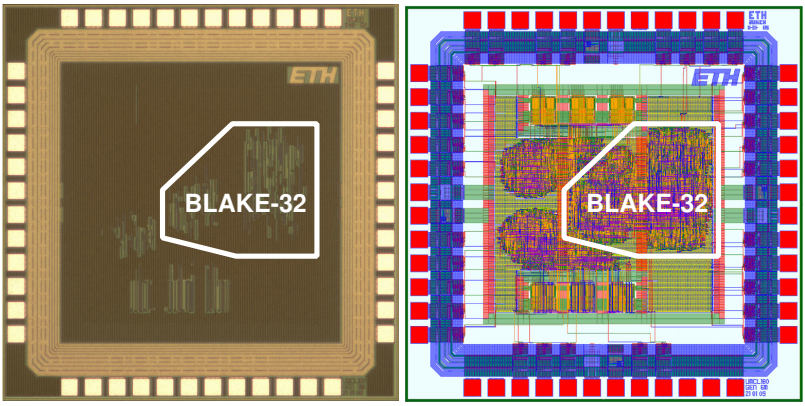

Figure 3.14: Die photo (left) and layout (right) of the compact BLAKE-32 implementation in 0.18 µm CMOS technology. Note that the ASIC hosts additional circuitry for the compact low-power Whirlpool cores of [65].

process, consume less energy independently of their size. This is the primary goal of the proposed memory architecture. The m memory, which is one of the largest memory units, but updated only once per compression, dissipates indeed the same amount of power as the half-sized h memory. This leads to a minor global contribution by the storing elements, which consume globally only 33.5 % of the total power, even if they fill more than 70 % of the chip area.

In resource-constrained environments like RFID systems or smart cards, power is often limited, as is the total silicon size. Decreasing the supply voltage becomes an efficient solution to reduce the overall consumption. As can be seen from Figure 3.15, this causes a proportional slowdown of the working frequency. It becomes thus important that in low-voltage regimes the frequency still satisfies the speed requirements of the target communication protocol. For the case of the RFID standards ISO 18000, ISO 14443, or ISO 15693, working in high-frequency (HF) and low-frequency (LF) domains, the operating frequency can reach the 13.56 MHz [66, 67]. By selecting

Table 3.6: Detailed chip area and power contribution of the compact BLAKE-32 core.

Component	**Area**		**Power**[a]
	[GE]	[%]	[%]
m mem. (16 w)	3295	24.3	3.4
v mem. (16 w)	3457	25.5	26.4
h mem. (8 w)	1681	12.4	3.1
s mem. (4 w)	926	6.8	0.4
t mem. (2 w)	550	4.1	0.2
Controller	776	5.7	6.6
Round	2890	21.3	60.0
Total	13 575	100.0	100.0

[a]The power consumption values of the individual modules are extracted from a post-layout simulation-based power analysis.

a correct functional region from the shmoo plot, we can decrease the supply voltage to 0.65 V, ensuring a correct behavior of the BLAKE-32 core up to 18 MHz.

Real power measurements of the core energy dissipation have been performed using a long randomized message as input. The mean power consumption, measured during the compression process, indicates that the chip dissipates 22.32 mW in nominal condition at the working frequency of 200 MHz. For the case of 13.56 MHz, i.e., the maximum frequency of HF RFID applications, the core dissipates 130 µW at 0.65 V. This value is far below the predictions given in [72] (< 500 µW). In RFID systems, the voltage is often affected by the distance between reader and tag [73]. Despite reliability issues due to voltage-scaling, the ability to work at lower voltage regimes becomes thus fundamental.

To meet the restrictive constraints given in [74] (mean current below 10 µA), the frequency must be further scaled to 100 kHz (see [75]). At this speed the chip requires only 0.55 V to generate correct output data.

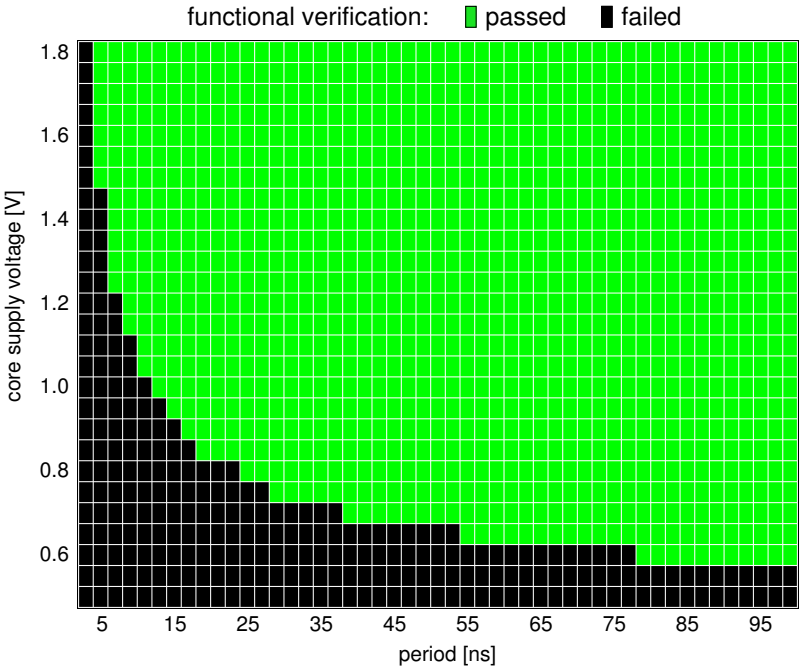

Figure 3.15: Period-voltage shmoo plot for the compact BLAKE-32 design.

Table 3.7 gives a comparison with other cryptographic protocols (not necessarily hash functions, and of different security levels), e.g., AES-128, at 100 kHz frequency. Although it has the largest area, the BLAKE core turns out to be the most efficient circuit in terms of mean current. Note, however, a difference in the CMOS fabrication processes. Nonetheless, the area demand of the proposed implementation can be further reduced by removing the message block memory and the salt support. For the first case we can suppose the presence of an external tamper-resistant memory, that stores the secret message, for the second case we simply omit an added functionality of the BLAKE algorithm. We designed the layout of a modified version

Table 3.7: Performance comparison of cryptographic primitives.

Algorithm	**Area** [kGE]	**Latency** [cycles]	$\mathbf{I}_{\text{mean}}$ [µA]	**Tech.** [µm]
BLAKE-32	13.575	816	0.7	0.18
SHA-1 [68]	6.122	344	7.7	0.18
SHA-256 [69]	10.868	1128	3.2	0.35
Whirlpool [65]	7.822	1650	5.0	0.18
MD5 [69]	8.001	712	3.2	0.35
AES-128 [70]	3.400	1032	3.0	0.35
ECC-163 [71]	11.904	306 000	5.7	0.18

of the compact BLAKE core. The size of this reduced BLAKE-32 version requires only 8.802 kGE. In Table 3.8, a comparison with other compact implementations of second round candidates of the SHA-3

Table 3.8: Overview of low-area architectures of SHA-3 candidates.

Algorithm	**Area** [kGE]	**Freq.** [MHz]	**Thr.** [Mbps]	**Tech.** [µm]
BLAKE-32	13.575	215	135.0	0.18
BLAKE-32[a]	8.602	100	62.7	0.18
BLAKE-32 [76]	25.569	31	15.4	0.35
CH16/32 [56]	7.630	100	2.0	0.13
Grøstl [76]	14.622	56	145.9	0.35
Keccak[b] [59]	5.000	200	52.9	0.13
Luffa-256 [60]	10.157	100	28.7	0.13
Skein-256 [76]	12.890	80	19.8	0.35

[a] This compact core uses an external memory to hold the message block and does not provide salted hashing.

[b] This implementation uses external memory to hold 1600-bit intermediate values during the hashing of a message.

competition is given. The results demonstrate a fair trade-off between area and speed for our compact BLAKE designs, which are well-suited for area-limited embedded systems.

Moreover, in [56] we applied the same design approach for memory components in a compact implementation of the CubeHash algorithm. We were then able to reduce the circuit size to 7.63 kGE (see Table 3.8). To the best of our knowledge this is, so far, the smallest published VLSI architecture of a complete SHA-3 second round candidate.

3.5 Development of a Hardware Evaluation Method for the SHA-3 Candidates

When the SHA-3 competition entered the second phase, we started a VLSI characterization of several candidates within three separate student projects. The resulting designs were manufactured in three different ASICs, each containing a dedicated interface for input/output (I/O) communication and the selected algorithms. At that time, we had implemented twelve out of fourteen candidate algorithms (all apart from ECHO and SIMD). We then decided to extend the analysis to all candidate algorithms.

Due to the gained experience, we deemed therefore that we had a sufficient knowledge to propose an efficient method to reasonably compare the hardware efficiency of symmetric cryptographic algorithms. We were not pushed by the intent of identifying the overall "best" function, but rather by the ambition to compensate the lack of concrete hardware requirements in the cryptographic community, e.g., in the official call for the NIST hash competition [40].

In the following sections, we describe the effort of developing a, in our opinion, optimal comparison methodology for VLSI designs of cryptographic hash functions; in this specific case, the 14 SHA-3 second round candidates. This work has been presented in [77]. Further on, the material (e.g., code, scripts, etc.) has been published on [78].

3.5.1 Evaluation Methodology

In this work we will attempt to make a fair comparison between VLSI implementations of a set of algorithms with similar function but very different structures. The main difficulty in this particular evaluation is the lack of concrete hardware specifications for the secure hash function candidates.

In practice, the specifications of the hardware are determined by the application. The hardware designers can then make several well-known trade-offs to come up with a design that offers the best compromise between, the required silicon area, the amount of energy required for the operation and the throughput/latency of the operation. For this study the requirements state *efficient hardware implementation* without being specific[7].

In some cases, such as telecommunication algorithms which have to fulfill requirements of certain well-defined standards, the application field alone sets sufficient constraints on the system. However cryptographic functions, like the SHA-3 hash function candidates, are used for a very wide range of applications with different requirements. This makes it difficult to determine which of the performance parameters is more important. A hash function that is part of a battery operated wireless transmitter would probably be optimized for energy consumption, while the same algorithm when implemented in a telecommunication base station would most likely favor a high-throughput realization.

For comparative studies, if concrete specifications are not present, the authors will usually determine one parameter to be more important (i.e., throughput) [53, 79, 80], or will come up with aggregate performance metrics such as hardware efficiency [76, 81, 82]. Both approaches have their problems. Focusing on one parameter will favor algorithms which are strong on one parameter (i.e. throughput), but will not merit algorithms which perform better in other scenarios. Aggregate performance metrics on the other hand, may end up hiding the absolute performance of an implementation, impractical design

[7]This should not necessarily be understood as criticism for the NIST specifications [40]. However, lack of concrete specifications make a fair comparison more difficult.

corners (i.e., very large area, very low throughput) may perturb the results.

In the following subsection we will first define the performance metrics that we will consider in this evaluation. We selected three crucial metrics from the list presented in Section 3.1.2. The next step will be to define specifications that will set limits on these performance metrics.

Performance Metrics

The most common metrics for hardware include operation speed, circuit area and power consumption. For this analysis we have decided to use the following three main metrics for performance:

- **Circuit Area** Generally speaking, the cost of an ASIC implementation of a function for a particular technology directly depends on the area required to realize the function[8]. In this evaluation, we will use the net circuit area of a placed and routed design, including the overhead for power routing and clock trees. The area will be reported in kGEs. This metric covers the evaluation criterion *4.B.ii Memory requirements* in the NIST specification [40].

- **Throughput** We need a measure to determine how fast the implementation is. To this end, we use the throughput. Furthermore, we assume that the hash function has been properly initialized, and the message sizes are matched to individual candidate functions for best case performance. This metric covers the evaluation criterion *4.B.i Computational Efficiency* in the NIST specification [40].

- **Energy Consumption** Power and energy metrics have gained importance in recent years. On one hand there are power density limits the circuits have to comply for sub 100 nm technologies, and on the other hand, for systems with scarce energy resources (handheld devices, smart cards, RFID devices, etc.), reduced

[8] This is only true if the area is within a certain range. Extremely large circuits will have yield penalties, while very small circuits will not be able to justify the overhead associated with manufacturing.

energy consumption equals to increased functionality or longer operating time. In this evaluation, we will consider the energy consumption as our metric and will calculate the energy per bit of input information processed by the hash function. This metric partly covers the evaluation criterion *4.C.i.b Flexibility* in the NIST specification [40] as the energy efficiency is a deciding factor for implementation in constrained environments.

SHA-3 parameters

The SHA-3 minimum acceptability requirements state that all algorithms should support hash-value sizes of 224, 256, 384, and 512 bits, and support a maximum message length of at least $2^{64} - 1$ bits. All candidates are based on the iterated construction and, hence process the message in blocks. The message block size differs from algorithm to algorithm. In addition several submissions have included a salt input (see HAIFA model in Section 3.1.4). In our evaluation we have chosen:

- **Hash-value size of 256**
  Several algorithms use (slightly) different architectures for different output lengths. Additional circuitry is then required to support all possible hash sizes. By selecting a single length, we aim to focus on the core algorithm which also simplifies certain architectural decisions. Out of the four required sizes, we have eliminated 224 and 384 as they are not a power of two (always a disadvantage in hardware design). We have settled on 256 as it will usually result in smaller hardware and faster implementations.

- **Use the largest message block size available**
  For each algorithm we have used the largest message block size and we have assumed that the message has already been padded (i.e., the length of the padded message is an exact multiple of the message block size). For throughput computation we always give the maximum achievable values, e.g., very long message for algorithms that have an initialization procedure.

- **No salt inputs**
  Since not all algorithms provide such an input, we have not

included any salt inputs. For algorithms that provide a salt, the inputs are set to their default values according to the specification, and these constants have been propagated during synthesis to allow further optimizations whenever possible.

Defining Specifications

As mentioned earlier, the main difficulty in this evaluation is the lack of precise specifications that the candidate algorithms have to fulfill. Hardware design is based on finding a compromise between competing parameters that determine circuit performance. For example, there are several architectural transformations that allow to increase the throughput at the expense of the circuit area (see [24]). Without guiding specifications, it is difficult to determine which of the circuit metrics is more important for a design.

In summary, the NIST specifications in [40] require that the candidate algorithms to be computationally efficient (4.B.i), have limited memory requirements (4.B.ii), to be flexible (4.C.i) and simple (4.C.ii)[9].

The classical way to perform this analysis would be to concentrate on only the throughput metric and try to find out which algorithms are the fastest. In the last year, several groups presented comparative works and, almost certainly, others will be publishing new results to this effect. However, if only the maximum throughput requirement is investigated the flexibility of candidate algorithms may not be visible. Therefore, we suggest to use two separate specifications: an aggressive **high-throughput** target and a **moderate-throughput** target.

The high throughput target has been chosen to be beyond the expected performance of most algorithms, and would therefore still be able to rank the algorithms in their maximum throughput capability. Our observation has been that even with older fabrication technologies, such as 180 nm CMOS, several candidate algorithms are able to reach throughputs of multiple Gigabits/s.

[9]Note that computational efficiency could be interpreted in different ways, however, in the NIST specification it is stated that the "*computational efficiency essentially refers to the speed of the algorithm*". Similarly the memory requirements refer to the circuit area in hardware implementations

There are certainly applications which could make use of such throughputs. Such data rates, however, are way beyond the requirements for many applications. For the moderate throughput requirement we have decided to determine a throughput which is at least two orders of magnitude lower than that used in the first case.

Fixing one of the performance metrics allows us to make a fairer comparison between the remaining performance metrics (area and energy), and by considering two distinct throughput targets, we hope to uncover the flexibility of the candidate algorithms for different operational requirements. In particular, we will be interested in the circuit area for our high-throughput target, while we will be more interested in the energy consumption for our moderate-throughput target.

The maximum achievable throughput by a circuit implementing a cryptographic algorithm depends on the specific technology into which the circuit will be mapped. A throughput value that is easily achieved in 65 nm process may not be feasible at all when using a 180 nm process. Therefore, the specifications for our two scenarios have to be chosen while considering the capabilities of our target process.

We have decided to use the 90 nm CMOS process by UMC with the free libraries from Faraday Technology Corporation, mainly because we already had experience in designing ASICs with this technology and it was readily available within our design environment at the time of this study.

Our experiences from designing the three ASICs (one of which was manufactured using this target technology) have given us a good estimation for the expected performance of all algorithms in the 90 nm process. We have decided to use 20 Gigabits/s for our high throughput target and 0.2 Gigabits/s for our moderate performance specifications. In the high-speed mode, almost all designs should be pushed to their speed limit, while with the latter we can evaluate the scalability and therefore the flexibility of each candidate algorithm.

ASIC Realizations

During this work (Master and Semester Theses [83, 29]), twelve out of the fourteen second round SHA-3 candidates (some with several architectural variations) were fabricated in three different ASICs as

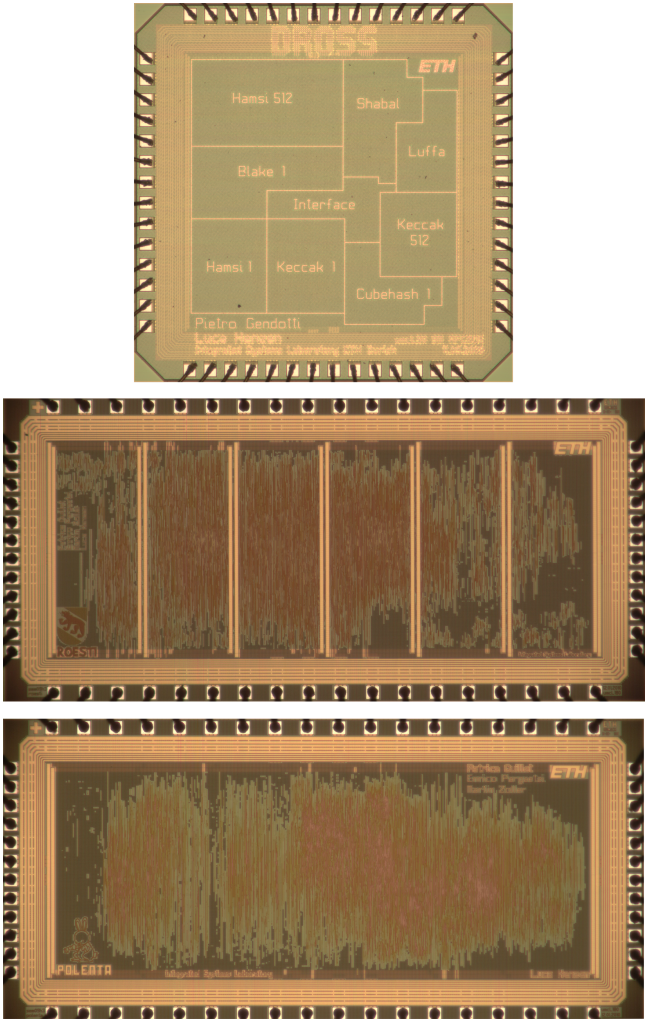

Figure 3.16: Photograph of the fabricated 90 nm chip implementing BLAKE, CubeHash, Hamsi, Keccak, Luffa and Shabal (top). Photograph of the two 180 nm chips implementing Fugue, Grøstl, JH, and SHAvite (middle), and BMW and Skein (bottom).

shown in Figure 3.16. Table 3.9 shows a list of algorithms that were implemented and their performances measured on the manufactured chips.

Table 3.9: Measured results of the algorithms implemented in the student projects.

Algorithm	**Area** [kGE]	**Throughput** [Gbps]	**Energy** [mJ/Gbit]	**Technology** [nm]
BLAKE-32	33.55	7.314	15.291	UMC 90
BMW-256	95.00	3.527	31.407	UMC 180
CubeHash16/32-256	39.69	8.000	20.700	UMC 90
Fugue-256	26.00	2.806	122.506	UMC 180
Grøstl-256	65.00	4.064	73.075	UMC 180
Hamsi-256	32.25	7.467	23.624	UMC 90
Hamsi-512	68.66	7.467	46.605	UMC 90
JH-256	44.00	2.371	72.885	UMC 180
Keccak-256[a]	27.85	39.822	5.726	UMC 90
Keccak-512[a]	26.94	19.911	11.933	UMC 90
Luffa-256	29.70	22.400	9.482	UMC 90
Shabal-256	35.99	4.923	30.713	UMC 90
SHAvite-3_{256}	48.00	2.452	93.764	UMC 180
Skein-256-256	27.00	1.917	44.329	UMC 180

[a]First round specification.

A real silicon implementation is the best way to validate a design and to determine its true potential. However, during this work we have realized that several practical factors have affected these results. The maximum available silicon area (that can be afforded for this project), the total number of I/O pins, the capabilities of the test infrastructure that is available for the test of the ASIC have all set limits on the implementations.

Since none of the designs was large enough to merit its own ASIC, each ASIC comprised of several independent modules. All modules shared a common interface which provided the inputs and collected the

outputs from individual hash function realizing cores. For practical reasons, cores with similar clock frequencies were grouped together and were optimized using common constraints. In many cases compromises had to be made to allow two or more cores to be optimized at the same time. All of these had non-negligible influence on the outcome.

Practical considerations for testing the systems has brought even more constraints. The necessity to include test structures (scan chains) adds some overhead, but more importantly, the maximum achievable clock rate greatly depends on the capabilities of the ASIC test infrastructure available. Designs with a high clock frequency (more than 500 MHz for 90 nm designs) put yet other constraints. When compared to designs running at lower frequencies, these designs suffer more from clock and power distribution problems, and are difficult to test at speed.

When designing these three ASICs, we were forced to make many design decisions (i.e., blocks running faster than 700 MHz were deemed to be impractical within our environment) based on practical constraints which had its influence on the results. Scheduling constraints have also played a role in the choice of technology used to implement the designs. For the last two ASICs, there were no feasible 90 nm multi project wafer (MPW) runs available. Consequently we had to submit these designs to a 180 nm run, which in turn made direct comparisons more difficult.

For this reason we have taken the design experience from the actual implementation of the individual cores, and have decided to re-implement all cores without considering these practical limitations. In particular we have decided:

- **No limits on the clock frequency**
  In this study we will not set any artificial limits on the clock rate. Obviously designs with high clock rates will still face the penalties for clock distribution, but we will not deal with practical considerations such as test, crosstalk and I/O limitations.

- **No test structures**
  Testing is an essential part of IC design. The exact overhead for testing depends on many factors, such as the desired test

quality, and a one-size fits all solution is difficult to find[10]. Since the designs in this study will not be manufactured directly we chose not to include any test specific structures into the designs to have a fair comparison.

- **Assumed an ideal interface**
  The candidate algorithms differ in the number of I/Os they require. We have assumed that these core will eventually be part of a larger system which has an adequate I/O interface matching the requirements of each core. In this way, every function can express its maximum potentiality without suffering from any external limitation. However, we made no assumptions about how long the inputs stayed valid, all required inputs were sampled by the cores at the beginning of the operation. In other words, we implemented an internal message block memory for designs that require the input to be stable for more than one clock cycle.

- **No macro blocks**
  We have not used any macro blocks to realize look-up tables (LUTs) or register files for portability reasons. All LUTs and memory blocks were realized by standard cells.

3.5.2 Implementation

Design flow

The same design procedure was used for all candidate algorithms. We have first developed a "golden model" based on the *Known Answer Tests* provided by the submission package. This golden model was then used to generate the stimuli vectors and expected responses that we have used to verify the register transfer level (RTL) description of the algorithm written in VHDL.

[10]Simply using a full-scan methodology for example would not ensure that all designs have the same test coverage. Furthermore certain designs could be partially tested using functional vectors, or would be more amenable to built-in self-test (BIST) structures.

We have then used Synopsys Design Vision-2009.06 to map the RTL description to the UMC 90 nm technology using the RVT standard cell library fsd0a_a_2009Q2v2.0 from Faraday Technology Corporation. All outputs are assumed to have a capacitive loading of 50 fF (equivalent to the input capacitance of about 9 medium strength buffers), and the input drive strength is assumed to be that of a medium strength buffer (BUFX8).

We used the worst case condition (1.08 V, 125 °C) characterization of the standard cell libraries. We have decided to use worst case characterized libraries in order to guarantee that we can meet the specifications. Table 3.10 is given as a reference to be able to compare the three characterizations that are commonly available (worst, typical, best) for one of the candidate algorithms.

Table 3.10: Comparison of different characterizations, synthesis results for the ECHO algorithm.

	Worst Case	Typical Case	Best Case
Supply Voltage	1.08 V	1.2 V	1.32 V
Temperature	125 °C	27 °C	-40 °C
Critical Path	3.49 ns	2.24 ns	1.59 ns
Throughput	13.75 Gbps	21.42 Gbps	30.19 Gbps
Relative Performance	64.2 %	100 %	140.9 %

Depending on the throughput requirements, we tried different architectural transformations such as parallelization, pipelining to come up with an architecture that meets (or comes closest to meeting) the requirements. We then used the Cadence Design Systems Velocity-9.1 tool for the back-end design. The technology of this evaluation uses 8 metal layers (metallization option 8m026), out of which the top-most two are double pitch (wider and thicker). A square floorplan is generated, leaving 30 µm space around the core for the power connections. For all designs we have used a 85 % utilization of the core area. In other words, we have left 15 % of the area for post-layout optimization and power and ground distribution overhead. For power routing we have used a power grid at Metal-7 and Metal-8.

Subsequently, the design was placed, a clock tree was synthesized, and subsequently the design was routed. After every step, the timing was checked, and if necessary a timing optimization was performed. At the end, if a valid layout without any design rule check (DRC) violations were found, the total core area was reported as the area of the system. The total core area excludes the 30 µm space reserved for power rings, but includes all the available area that the placement and routing tool can use for the design. By default, all designs start with a 15 % overhead for post-layout optimizations. Depending on the design, some amount of this overhead is used during various optimization phases during the back-end design. However, it is difficult to quantify the minimum required overhead for every design reliably. We have decided to start all designs with the same initial placement density, and we verified that the final design was not overly-congested. In a congested design, the routing solution includes many detours which adversely affect timing. For these designs, the initial row utilization would have been reduced by 5 %, increasing the overhead. This was not necessary for any designs in this study[11]. In some designs, the routing resources are sparsely utilized. Such designs would benefit from a higher initial row utilization, which would result in a slightly smaller circuit without noticeable timing penalties. As mentioned earlier, it is not trivial to make sure that two designs have exactly the same amount of overhead. Therefore, we have not considered changing the default row utilization, unless there was a noticeable problem.

The timing results were taken from the finalized design. First, the Velocity tool was used to extract the post-layout parasitics and an SDF file containing the delays of all interconnections and instances was generated. The final netlist and the SDF files were read by the Mentor Graphics Modelsim-6.5a simulator and the functionality of the design was verified. At the same time, a value change dump (VCD) file that records the switching activity of all the nodes during the simulation was produced. To have more realistic results, the start of the VCD file is chosen after the circuit has been properly initialized. This VCD file was then read back into the Velocity tool

[11] Note that the initial density strongly depends on the technology options such as used metal layers. We have used 85 % as a result of our previous experience with this particular technology.

and a statistical power analysis was performed. The "Total Power" number was used to determine the energy consumption of the system.

Algorithms

For a given candidate algorithm, there are several well-known architectural transformations such as parallelization, pipelining, loop-unrolling, etc. that will allow different trade-offs between circuit size and throughput. In addition, within the submission document, the authors often suggest different computational methods to perform a specific transformation of their candidate function. A good example are the frequently used S-boxes. They can be implemented as LUTs, or they can be realized as a circuit that computes the underlying function mathematically. To make matters worse, the exact trade-off between alternative realizations may only be visible after placement and routing. All these aspects broaden the spectrum of the possible hardware architectures. For a single candidate, there is often a large set of circuits with different trade-offs between size and speed. To identify the "best" design among many possibilities is not a trivial task. Despite all attempts to formalize architectural exploration, our experience has been that optimizing the circuit still remains a manual task, that relies on the skill and experience of the designer.

In this work, for each candidate algorithm we have selected what we believe was the most appropriate architecture that was able to reach the target throughput (20 and 0.2 Gbps) with minimal resources. For every candidate we designed and implemented two different architectures. The specifications of the single designs used within this work, is given in Appendix A. We make no claims that any of the architectures we have reported in this paper is the optimal possible architecture for a given candidate algorithm. In our opinion, it is not possible to make such a claim, and the exact implementations should be open to public scrutiny and review. For this purpose, we have made all the source code that was used for this evaluation public.

3.5.3 Results

In this section, we present the performance of the circuits implemented for high and moderate speed environments. The comparison between

these two scenarios gives a further overview of the efficiency and flexibility of the candidate algorithms. We will refrain from concluding remarks about the performance of the algorithms, as we do not consider the results complete without public scrutiny.

For each architecture we report two operating frequencies. The "Maximum Clock Frequency" is the maximum achievable clock frequency of the given architecture. When operating at this clock frequency the circuit achieves the given "Maximum Achievable Throughput". In most cases, this throughput is not exactly the same as the required throughput (either 20 or 0.2 Gbps). The second clock frequency states the clock rate required to reach the target throughput. The final value in the tables is a relative indicator of how close the architecture is in achieving the target clock frequency. A number lower than one means that the architecture failed to achieve the target throughput. One can take this as a ratio of how closely we were able to optimize the circuit to the given target performance.

High Throughput Scenario

As expected, not all the circuits optimized for high-speed were able to reach the target throughput. Only two algorithms, Keccak and Luffa, were able to achieve the constraint. Table 3.11 lists the main performance figures for all architectures. In this scenario, both area and energy were sacrificed to achieve high-throughput. The corresponding layouts can be seen in Figure 3.17. The scale is given in the lower right corner of the figure. Circuits with a higher congestion rate (e.g., BMW or SIMD) require indeed the entire core for routing, and would probably reach a faster throughput with more core area, i.e., a lower row utilization. Particularly interesting is also the local congestion for the 8-bit LUT-based S-boxes which makes them easily identifiable within ECHO, Grøstl, Fugue, and partly SHAvite-3.

Medium Throughput Scenario

The moderate-throughput circuits match the target throughput of 0.2 Gbps without difficulty. As can be seen in Table 3.12, the maximum achievable clock rate always exceeds the clock frequency required for 0.2 Gbps operation. To some extent, the additional speed can

Table 3.11: Post-layout performances of all candidate algorithms for a target throughput of 20 Gbps in the UMC 90 nm process.

Algorithm	**Area** [kGE]	**Energy** [mJ/Gbit]	**Maximum Achievable Throughput** [Gbps]	**Maximum Clock Frequency** [MHz]	**Clock Freq. for 20 Gbps Throughput** [MHz]	**Max. / Target Frequency Ratio**
BLAKE-32	47.5	11.00	9.752	400	820	0.49
BMW-256	150.0	16.86	8.486	298	703	0.42
CubeHash16/32-256	42.5	13.71	10.667	667	1250	0.53
ECHO-256	260.0	43.41	13.966	291	417	0.70
Fugue-256	55.0	15.60	8.815	551	1250	0.44
Grøstl-256	135.0	14.13	16.254	667	820	0.81
Hamsi-256	45.0	15.90	8.686	814	1876	0.43
JH-256	80.0	17.54	10.807	760	1406	0.54
Keccak-256	50.0	2.42	43.011	949	441	2.15
Luffa-256	55.0	6.92	23.256	727	625	1.16
Shabal-256	45.0	14.83	6.819	693	2033	0.34
SHAvite-3_{256}	75.0	19.21	7.999	562	1406	0.40
SIMD-256	135.0	35.66	5.177	364	1406	0.26
Skein-256-256	50.0	30.47	3.558	264	1484	0.18

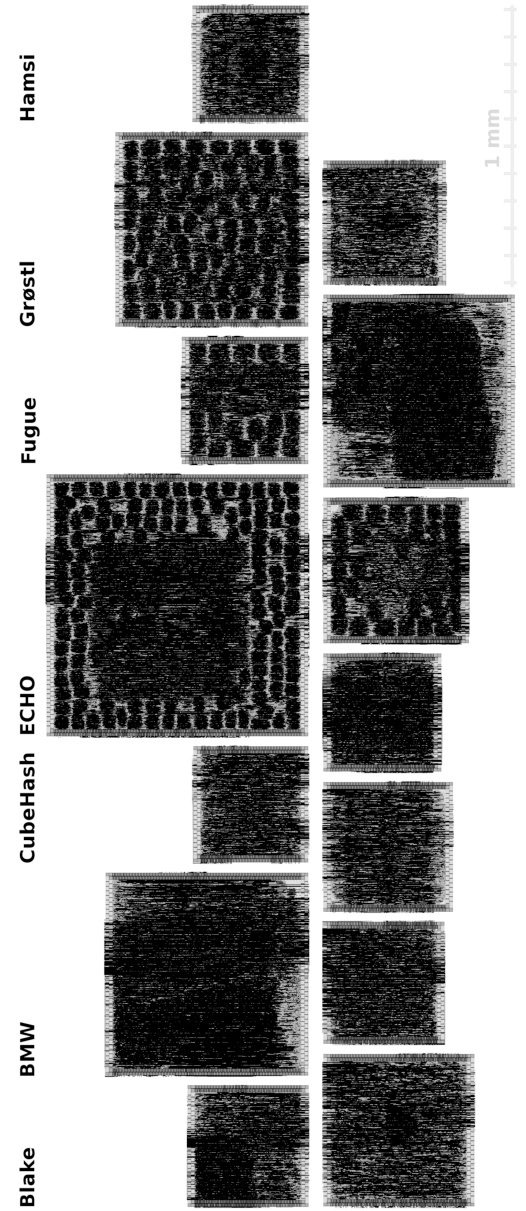

Figure 3.17: The final layouts of all candidate algorithms for a target throughput of 20 Gbps.

Table 3.12: Post-layout performances of all candidate algorithms for a target throughput of 0.2 Gbps in the UMC 90 nm process.

Algorithm	**Area** [kGE]	**Energy** [mJ/Gbit]	**Maximum Achievable Throughput** [Gbps]	**Maximum Clock Frequency** [MHz]	**Clock Freq. for 0.2 Gbps Throughput** [MHz]	**Max. / Target Frequency Ratio**
BLAKE-32	16.0	13.00	0.463	73.282	31.646	2.32
BMW-256	85.0	14.04	1.845	64.876	7.031	9.23
CubeHash16/32-256	16.0	10.50	1.741	217.581	25.000	8.70
ECHO-256	60.0	59.44	0.204	137.061	134.771	1.02
Fugue-256	19.0	9.02	1.828	114.260	12.500	9.14
Grøstl-256	25.0	22.28	0.412	128.750	62.500	2.06
Hamsi-256	15.0	35.12	0.200	150.083	149.925	1.00
JH-256	37.5	13.03	1.909	134.228	14.063	9.54
Keccak-256	27.5	5.50	6.767	149.276	4.412	33.83
Luffa-256	22.0	21.79	1.265	118.624	18.751	6.33
Shabal-256	25.0	26.57	0.399	128.634	64.475	2.00
SHAvite-3_{256}	25.0	11.43	1.871	131.527	14.063	9.35
SIMD-256	90.0	32.49	0.943	66.295	14.063	4.71
Skein-256-256	19.0	32.67	0.200	118.765	118.765	1.00

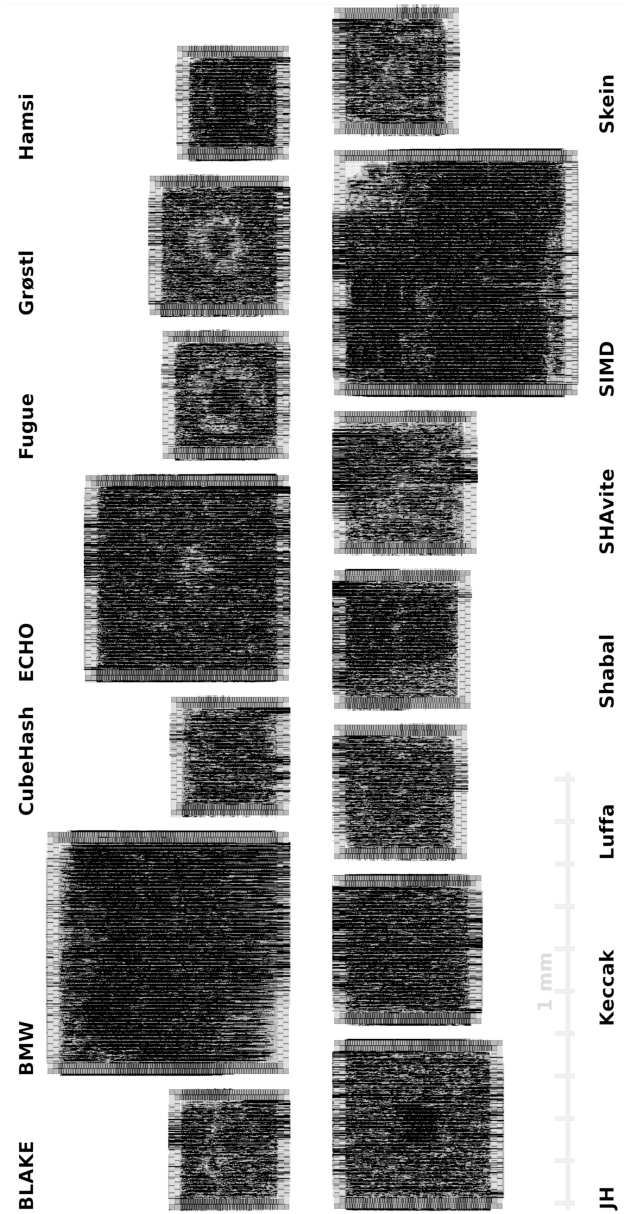

Figure 3.18: The final layouts of all candidate algorithms for a target throughput of 0.2 Gbps.

be traded to reduce the overall energy consumption, by lowering the supply voltage. It must be noted that there is a lower limit for the supply voltage (around 0.5 V for this process). Such voltage scaling techniques were not considered in this comparison, all results are listed for 1.2 V supply voltage.

In this scenario, timing was quite relaxed and the main figure of merit becomes the area and the energy dissipation. The layouts of all fourteen architectures are compared in Figure 3.18, with the scale indicated on the bottom left.

The most interesting result is that a smaller area (or indeed throughput) does not always equal lower energy consumption (see Hamsi or Skein compared to BMW or SIMD). It must be noted that, no special precautions were taken for a low-power design (e.g., proper clock-gating, input-silencing). In addition some architectural decisions resulted in increased number of operations and/or increased circuit activity which affected the energy consumption differently for separate algorithms. We believe that there is much room for improvement in terms of low-power performance of the architectures. We must conclude that the present specifications do not necessarily result in low-power realizations in the medium-throughput corner. In a next step, the design methodology could be extended to provide a low-power scenario.

Sources of Error

Although we have tried our best to ensure a fair comparison, there are many factors that might have influenced the results. In this section we try to outline the possible sources of error in our results, and outline what we have done to address them.

- **Conflict of interest**
  Our interest in implementing the SHA-3 candidate algorithms has started by investigating optimal hardware implementations of BLAKE. We have tried to be as impartial as possible when implementing other candidate algorithms. However, it is true that we are more familiar with this algorithm than with any other.

- **Designer experience**
  The algorithms have been implemented by a group of students over a period of several months. Different designers may have more or less success in optimizing a given design. We have confidence in our team, but it is possible that for some algorithms we have inadvertently missed a possible optimization while for the others we were more successful. In addition, over time the designers naturally gain more experience and are more successful with the designs.

  We believe that the most important aspect of a fair comparison is openness. For this reason we have made the source code and run scripts for the electronic design automation (EDA) tools used to implement all designs presented in this paper available [78]. In this way, other groups can replicate our results, and can find and correct any mistakes we might have made in the process.

- **Accuracy of numbers**
  The numbers delivered by synthesis and analysis tools rely on the library files provided by the manufacturer. The values in the libraries are essentially statistical entities and sometimes have large uncertainties associated with it. In addition, most of the design process involves heuristic algorithms which, depending on a vast number of parameters, can return different results. Our experience with synthesis tools suggest that the results have around $\pm 5\%$ variation. We therefore consider results that are within 10% of each other to be comparable.

  In an effort to be more accurate we have chosen to report post-layout area numbers that include clock and power distribution overhead. We have designed all circuits with the same overhead. For some circuits this overhead is adequate, for others it is too much, and for others is insufficient. We made sure that there is an acceptable solution for all cases.

- **Bias through specification**
  We have chosen two design corners in our applications. These specifications have helped us to have a common base for comparing all 14 algorithms. Regardless of how these specifications

are chosen, it is possible that they benefit some algorithms more than the others. We hope that similar studies by other groups which use different specifications will help to give a clearer picture.

- **Simplification due to assumptions**
  All our assumptions, the specific choices we made for SHA-3 parameters and the practical choices we made in the design flow will have some effect on the results. For example, we have decided not take IR-drop or crosstalk effects into account. As a result, the cores that achieve their reported performance by using very high clock frequencies will be more difficult to realize in practice. The assumptions in the design flow are a practical necessity and were designed to create a methodology in which the same solution could be used for all designs.

3.5.4 Final Remarks

We have presented a methodology to compare the SHA-3 candidate algorithms. Our previous experiences in designing ASIC implementations of candidate algorithms (cf. Table 3.9) has been instrumental in developing what we believe is a fair set of specifications. Rather than targeting outright performance, we have set limits for one performance metric (throughput) and re-implemented all algorithms to meet two distinct throughput requirements. This enabled us to compare the flexibility of the algorithms (cf. Tables 3.11-3.12).

A public selection process, such as SHA-3, invariably attracts a large number of submissions with many different algorithms. In early stages of the selection process, the sheer number of algorithms (51 in the first round) makes it impractical to employ a detailed analysis for hardware suitability. Our experience has shown that even with the 14 second round candidates, it is difficult to present an authoritative and fair evaluation of all candidates. We believe that for the final round of evaluations, a similar approach to what we have demonstrated in this paper should be utilized: Clear constraints should be set for the implementations, preferably more than one performance corner should be targeted, the evaluation process should be well documented and the errors in the evaluation process should be openly discussed. We would

also suggest the addition of a low-power corner that also considers voltage scaling for low-power operation to our methodology.

In many parts, we have extensively commented on limitations of our methodology, and have included a whole subsection on sources of error. We strongly believe that any such comparison must be thorough with its analysis of error sources and clear with its performance metrics.

4

High-Speed Authenticated Encryption

Together with confidentiality, data origin authentication is a fundamental property to establish a secure channel. In Chapter 2, we have seen that advanced cryptographic schemes based on public-key or quantum cryptography still require an authentication protocol in order to avoid basic attacks. To this end, authenticated encryption combines both confidentiality and authenticity in a unique scheme. Although several emerging standard were already published, authenticated encryption was formally defined in 2000 by Bellare and Namprempre in [84].

In this chapter, we introduce the concept of authenticated encryption and explain the major application fields. We mostly concentrate on dedicated algorithms that define a specific mode of operation for block ciphers. In particular, we propose two application-optimized FPGA architectures of the Galois/Counter Mode (GCM) algorithm in combination with the AES block cipher. Indeed, GCM is so far the

fastest and most hardware-friendly authenticated encryption mode approved by NIST.

The second AES-GCM core presented in this chapter has been further implemented in a complete embedded system for authenticated encryption of 2G FC nodes. Enhanced with QKD, the underlying link encryptor has been deployed for several months within the SwissQuantum network.

4.1 Background

Each days, billion of messages in form of internet protocol (IP) packets are transmitted by the Internet. The paramount goal of cryptography is to implement a *secure channel* between two end points that is used to exchange information. A network link is considered secure when the confidentiality of the communication is warranted, as well as, data integrity and generic authenticity. We have seen that confidentiality can be achieved by means of standard encryption schemes (e.g., block ciphers in CBC mode), whereas MACs provide data integrity and origin authentication.

The combination of these symmetric-key schemes (cipher and MAC) is known as the *generic-composition* approach for confidentiality and authenticity [85]. However, the fusion of a secure cipher with a secure MAC is often *ad hoc* and could still generates an insecure scheme. Only in the last decade, researches have started to develop single cryptographic models that provide both security goals. We refer to these methods as *authenticated encryption* modes.

The principal advantage of dedicated authenticated encryption over the generic-composition approach is the clear specification on how to achieve privacy and authentication at the same time and the lack of accidental errors generated by combining MACs with encryption in an insecure fashion.

4.1.1 Different Classes

In the standard authenticated encryption process, Alice first encrypts her message (plaintext) and generates a fixed-length bitstring (authentication tag) from the message. She sends then the encrypted message

(ciphertext) with the tag to Bob, who in turn decrypts the message and generates his own tag. The decrypted message is considered valid, or authentic, only if Bob's tag is equal to the received tag. Obviously, Alice and Bob still need to share a common secret key.

Authenticated encryption schemes can be classified into two distinct families. If the algorithm is capable to uniquely authenticate the encrypted message, we use the term of authenticated-encryption (AE). Rogaway, however, defined a further class in [86], i.e., authenticated-encryption with associated-data (AEAD), where additional data can be authenticated without being encrypted. AEAD-based algorithms are much more flexible, since plain MACs can be constructed simply by zeroing the length of the plaintext when only integrity/authenticity of the message is required.

The cryptographic community has developed a wide range of different algorithms for authenticated encryption[1]. Nevertheless, not all schemes have been recognized and standardized. Currently, only the Counter Mode with CBC-MAC (CCM) and the GCM (both AEAD) have been approved for use by the U.S. federal government.

Normally, authenticated encryption schemes are algorithms that specify the operation of a block cipher. For this reason, they are classified within the domain of block cipher modes of operation.

4.1.2 Applications

Many internet communication protocols have been developed for confidential authenticated data transfer. The three most popular are TLS (and its predecessor SSL), secure shell (SSH), and IPsec. The first two protocols operate on the upper layer of the open systems interconnection (OSI) model, while the latter is used in the network layer and is therefore not limited to a single application.

A further branch of security systems are the so-called *link encryptors*. They operates on the lowest layers, usually layer 1 or 2 of the OSI model, and are designed to interact transparently with the network. Link encryptors offer an improved flexibility with respect to the target application and in general are able to maintain the network performance (maximum bandwidth and low latency).

[1] See the NIST's modes development page `http://csrc.nist.gov/groups/ST/toolkit/BCM/modes_development.html`.

Typically, they are implemented in separate hardware modules[2], used predominantly to secure storage area networks (SANs) or multi-gigabit network backbones. Due to their broad range of application, link encryptors must support numerous standardized communication mechanisms, such as Ethernet, FC, synchronous optical networking (SONET), and synchronous digital hierarchy (SDH).

In general, authenticated encryption systems rely on public-key schemes to exchange the secret key. However, quantum cryptography began to be deployed within practical security applications since the commercialization of the first single photon detectors. Quantum cryptography-based systems are able to combine general symmetric-key encryption with the robustness of QKD to build a provably secure communication mechanism.

In spite of the theoretical excellence of QKD, its application in practice is often far from perfect. Errors could indeed creep in owing to mundane environmental noise. Two leading companies in the field of practical quantum cryptography that commercialize QKD-based security systems are MagiQ Technologies and ID Quantique (IDQ). Interestingly, several products released by IDQ have been the focus of practical attacks [87, 88], demonstrating the feebleness of current available QKD-based systems and the ineluctable gap between theory and practice in quantum physics.

4.2 The Advanced Encryption Standard

The first recognized and thus widely used block cipher has been the DES algorithm. DES has been deployed for about 20 years until its key size (and therefore its security) was no longer enough to be considered computationally secure. At the end of the twenty century, exhaustive search on the key space of DES required less than a day[3].

DES has formally been replaced in 2001 by the AES algorithm. Through a public competition, NIST selected the Rijndael algorithm as new official block cipher. Due to the improved long-term security

[2] Link encryptors are often implemented on digital signal processor (DSP) or FPGA devices.

[3] The size of a DES key is 56 bits.

(key size up to 256 bits), AES has been immediately applied in most security applications requiring confidentiality. At present, AES is considered by industry and government environments as the essential scheme to protect sensitive information.

For over a decade, the cryptanalytic effort on AES has been relatively slow. Only in 2009, a surge of new cryptanalyses [89, 90, 91] improved the knowledge on the cipher notably, culminating in the attack described in [92] against the full version with 256-bit key. The introduction of these recent works started a fervent and still ongoing discussion on the practicality of such cryptanalytic attacks. Nevertheless, the vast majority of the cryptographic community is firmly convinced of the cipher security and it still believes that AES is a good PRP.

4.2.1 Algorithm Specifications

The AES algorithm is specified to work on 128-bit blocks of data. A single block is divided to form a 4×4 matrix of bytes, called *state*. The three standardized versions AES-128, AES-192, AES-256 work with different key lengths (128, 192, and 256 bits) and are specified with a different number of rounds (10, 12, and 14).

The AES round function schedules four distinct transformations in a fixed order (see Figure 4.1)[4]:

1. `SubBytes` is a non-linear transformation that substitutes the bytes of the state independently, using a S-box built over two steps: computation of the multiplicative inverse in the Galois field GF(2^8) followed by an affine transformation over GF(2).

2. `ShiftRows` cyclically shifts the bytes in the last three rows of the state over an increasing offset.

3. `MixColumns` multiplies in GF(2^8) each column of the of the state with a fixed polynomial modulo $x^4 + 1$.

4. `AddRoundKey` adds a *round key* to the state in GF(2).

[4]For more details consult [93].

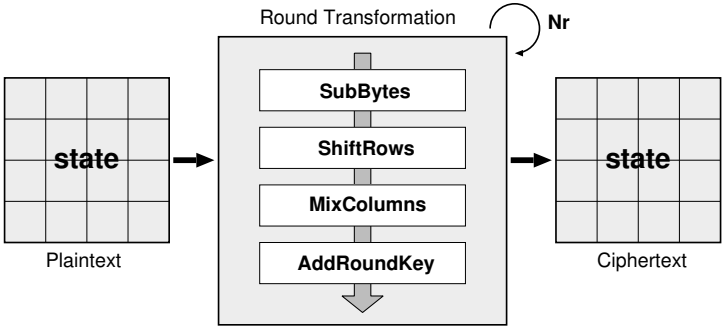

Figure 4.1: AES encryption of a single data block.

The AES round is completed with the key expansion routine, which generates the round keys from the original key. This expansion is defined on 4-bytes words and mainly reuses the aforementioned transformations of the round. In the AES specification, a single round key is made up by four words and is updated to the next round key using a combination of the `SubWord` (S-box over a 4-byte input) and `RotWord` (word shift) functions.

The AES decryption is exactly the inverse mapping and computes the original plaintext block from an encrypted ciphertext. The AES round is thus executed backwards, using the inverse of the single transformations, i.e., `InvSubBytes`, `InvShiftRows`, and `InvMixColumns`. The `AddRoundKey` function remains obviously the same, while the round keys are computed in the reverse order within the key expansion process.

4.2.2 Hardware Architectures

After the standardization of AES, numerous research contributions focusing on hardware implementations of the algorithm have been presented. Nowadays, almost each possible AES architecture, covering the entire design space (both in ASIC and FPGA), has been investigated. Companies and federal organizations have plenty of choice in selecting the suitable design.

We list the major components and strategies that characterize the AES hardware design.

Datapath The AES is defined over a 128-bit state with 8-bit operations. The designer has then the possibility to adapt or shrink the datapath of his architecture according to the system specification. Typical datapath widths are 128, 64, 32, 16, or 8 bits.

The datapath size affects directly the area occupation and the final throughput of the AES core. In practice, it represents a sort of indicative area/speed trade-off. In [94, 95] the authors analyze thoroughly different-datapath implementations for FPGA and ASIC technologies, respectively.

In-round The `SubBytes` transformation is the most costly (both in size and propagation delay) component of the AES round. The `ShiftRows` operation can be implemented as a straightforward rerouting of the signals without any specific hardware component, while the most efficient strategy to realize `MixColumns` is with combinational logic.

Several approaches to design the `SubBytes` have been developed and implemented. We describe briefly three main solutions:

- **Random-access memory (RAM)**
  The S-box is implemented as a LUT and stored in a dedicated memory. This approach is particularly appealing in modern FPGAs, where two 2048-bit S-boxes can be stored inside a dual-port BRAM.

- **Composite field**
  In [96], Rijmen suggests an efficient approach to compute the multiplicative inverse by considering GF(2^8) as the quadratic extension of GF(2^4). This reduction to composite fields brings significant improvement in flexibility and in the area costs of `SubBytes` (see [97]).

- **FPGA LUT**
  The third approach is applicable only in FPGA devices. As proposed in [98], the S-box can be directly instantiated in LUTs

located in the FPGA logic elements. As example, the basic logic elements of Xilinx FPGAs are called slices and each slice comprises a different number of LUTs depending on the selected device. In a Xilinx Virtex-5 chip a single 2048-bit S-box fits in 32 6-input LUTs.

Out-round The mode of operation of the block cipher influences remarkably the performance of the implementation. Feedback modes as CBC, CFB, and OFB lead to standard designs similarly to the iterative construction of hash functions. Here, the realization of a single round turns out to be the most efficient solution.

Full unrolled architectures combined with pipelining can be exploited only by the feedback-free modes ECB and CTR, favoring the latter for its security strength. Although the latency grows proportionally, unrolled designs are able to achieve 20-30 Gbps, simply by putting a pipeline stage between the rounds. To further increase the speed, pipelining can also be implemented inside the round and the `SubBytes` operation, which must then be realized with the composite field approach [99, 100].

Figure 4.2 shows the performance of eight full unrolled AES-128 cores in CTR mode for different levels of pipelining. Maximum hardware-efficiency is reached with five stages per round (in total 50 pipeline stages). Interestingly, with five stages the area is almost balanced between sequential and combinational logic.

4.3 The Galois/Counter Mode

The block cipher GCM has been standardized by NIST in order to provide an efficient authenticated encryption mode that can support modern multi-gigabit networks and is free of intellectual property limitations. In the last years, GCM has been applied in numerous standards, such as the IETF RFC 4106 for IPsec encapsulating security payload (ESP) or the IEEE P1619.1 project for authenticated encryption with length expansion for storage devices. Moreover, the combination of GCM with AES has been recommended in the IEEE 802.1AE standard for link security.

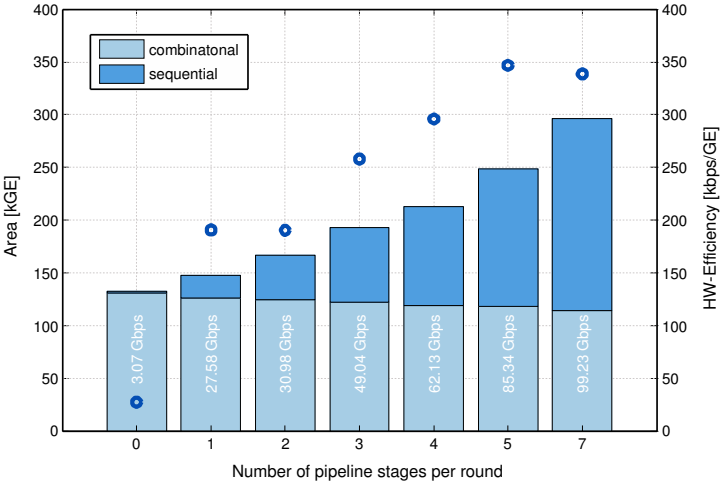

Figure 4.2: Area occupation (left) with corresponding hardware-efficiency (right) and throughput (white values) for different pipelined full unrolled AES-128 architectures. Post-synthesis results in 0.18 µm CMOS [99].

GCM is a AEAD scheme and can act as a stand-alone MAC. Upon request the message can indeed be authenticated without being encrypted. This particular mode is defined as Galois message authentication code (GMAC) [101].

GCM combines the CTR mode of operation for block ciphers with a MAC based on *universal hashing*[5]. The underlying universal hash function is based on a polynomial multiplication in GF(2^{128}). The selected finite field allows to match the size of the hash input with the cipher block length. Moreover, binary GF multiplication is

[5]A universal hash function family [102, 103] is a set $\mathcal{H}$ of hash functions h mapping a universe of keys $\mathcal{U}$ to $\{0, 1, \ldots, m-1\}$ with the following property: if $x, y \in \mathcal{U}$ and $x \neq y$ then the number of functions in $\mathcal{H}$ with $h(x) = h(y)$ is precisely $|\mathcal{H}|/m$. In other words, the parties select randomly a common hash function that has a collision probability of $1/m$ (m is the size of the hash-value domain). Polynomial hashing as in GCM is a class of universal hash functions.

particularly suitable for hardware designs and leaves space for parallel implementations (see below).

4.3.1 Algorithm Specifications

The GCM algorithm takes as input a plaintext P (or input message), split into 128-bit blocks $P_1, P_2, \ldots, P_n^*$, an initialization vector IV, some additional authenticated data $A = (A_1, A_2, \ldots, A_m^*)$, and the secret key[6] K. The size in bits of the final blocks P_n^* and A_m^* is defined by u and v, with $1 \leq u, v \leq 128$.

As outputs, the ciphertext $C = (C_1, C_2, \ldots, C_n^*)$ and the t-bit authentication tag T are generated.

The following equation defines the authenticated encryption of GCM:

$$\begin{aligned} H &= E_K(0^{128}) \\ Y_0 &= \begin{cases} IV||0^{31}1 & \text{if } \mathrm{len}(IV) = 96 \\ \mathrm{GHASH}(H, \{\}, IV) & \text{otherwise.} \end{cases} \\ Y_i &= \mathrm{incr}(Y_{i-1}) \qquad i = 1, 2, \ldots, n \\ C_i &= P_i \oplus E_K(Y_i) \qquad i = 1, 2, \ldots, n-1 \\ C_n^* &= P_n^* \oplus \mathrm{MSB}_u(E_K(Y_n)) \\ T &= \mathrm{MSB}_t(\mathrm{GHASH}(H, A, C) \oplus E_K(Y_0)). \end{aligned} \tag{4.1}$$

The successive counter needed for the CTR mode of the underlying block cipher are generated with the incr() function (increment modulo 2^{32} of the right most bits). The encryption process is depicted in Figure 4.3.

The GHASH() function in the last lines of (4.1) takes the $(m + n + 1)$-block input composed by H, A, and C, and compresses it to a single 128-bit block. The compression is performed by the sequential multiplication of A and C with the pre-computed term H over GF(2^{128}) with irreducible polynomial equal to

$$g(x) = x^{128} + x^7 + x^2 + x + 1. \tag{4.2}$$

[6] In GCM, the encryption process and the authentication code rest upon the same key K.

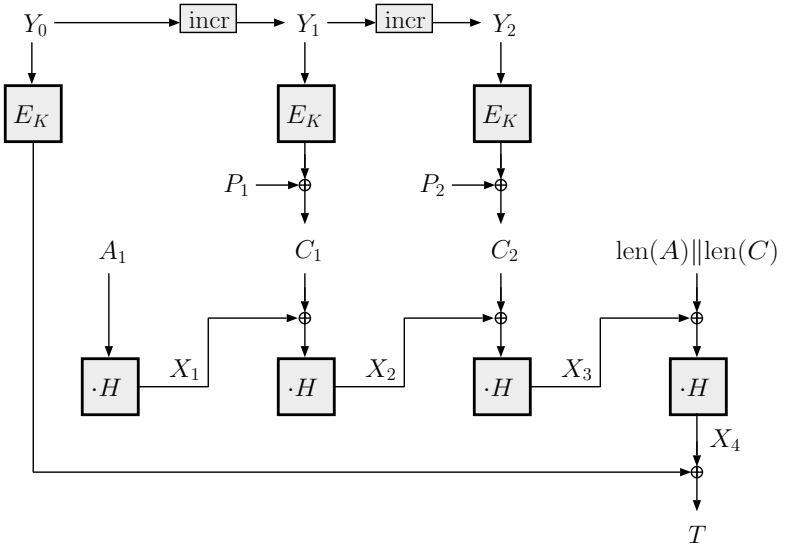

Figure 4.3: GCM authenticated encryption of a 256-bit plaintext P with 128-bit of additional authenticated data A.

The tag T consists in the first t bits ($64 \leq t \leq 128$) of the XOR between the encryption of the IV and the output $X_{m+n+1} = \text{GHASH}(H, A, C)$, where

$$X_i = \begin{cases} 0 & i = 0 \\ (X_{i-1} \oplus A_i) \cdot H & i = 1, \ldots, m-1 \\ (X_{m-1} \oplus (A_m^* || 0^{128-v})) \cdot H & i = m \\ (X_{i-1} \oplus C_{i-m}) \cdot H & i = m+1, \ldots, m+n-1 \\ (X_{m+n-1} \oplus (C_n^* || 0^{128-u})) \cdot H & i = m+n \\ (X_{m+n} \oplus (\text{len}(A) || \text{len}(C))) \cdot H & i = m+n+1 \end{cases} \tag{4.3}$$

The last multiplication is computed using the concatenation of the 64-bit sizes of A and C.

Since the block cipher operates in CTR mode, the authenticated decryption is simply computed by exchanging the input of the GHASH function with the incoming data, in this case the received ciphertext. The new generated tag T' is then compared with the received tag T. In case of mismatch, the decrypted plaintext is completely discarded and considered as modified or not authentic.

Parallel Multiplication

The parallelization process of the multiply operation in (4.3) was originally advised for hardware implementations in [85,101]. The goal is to divide the sequential *multiplication-addition* steps into several parallel computations that generate the same final result. By defining a parallelization degree constant q, the final X block can indeed be expressed as the sum of q sub-terms Q_i:

$$X_{m+n+1} = Q_q \oplus Q_{q-1} \oplus \ldots \oplus Q_1, \tag{4.4}$$

where

$$\begin{array}{lcl} Q_q & = & (((I_1 H^q \oplus I_{q+1}) H^q \oplus I_{2q+1}) H^q \oplus \ldots) H^q \\ Q_{q-1} & = & (((I_2 H^q \oplus I_{q+2}) H^q \oplus I_{2q+2}) H^q \oplus \ldots) H^{q-1} \\ & \vdots & \\ Q_2 & = & (((I_{q-1} H^q \oplus I_{2q-1}) H^q \oplus I_{3q-1}) H^q \oplus \ldots) H^2 \\ Q_1 & = & (((I_q H^q \oplus I_{2q}) H^q \oplus I_{3q}) H^q \ldots) H, \end{array} \tag{4.5}$$

and

$$\begin{array}{c} (I_1, I_2, \ldots, I_{m+n+1}) = \\ (A_1, \ldots, A_m^* || 0^{128-v}, C_1, \ldots, C_n^* || 0^{128-u}, \mathrm{len}(A) || \mathrm{len}(C)). \end{array} \tag{4.6}$$

If the length of the input blocks $m + n + 1$ is not a multiple of q, the last q multiplications are accordingly shifted through the Q_i terms. More important is that the different Q_i can be computed separately and then XORed only at the end of the authentication process.

4.4 FPGA Parallel-Pipelined AES-GCM Core for 100G Ethernet Applications

In July 2010, the IEEE announced the ratification of the IEEE 802.3ba standard for 40 Gbps and 100 Gbps Ethernet [6]. The project started in 2007 as response to the growing diffusion of bandwidth-intensive technologies. Nowadays, networked storage, video-on-demand, or social networking are emerging applications, contributing to an increased bandwidth requirement in the enterprise computing environment. Aggregating multi-line protocols working with the former 10 Gbps Ethernet requires indeed a 100 Gbps Ethernet interface for switch-to-switch interconnection in modern data centers.

The capability to introduce into the market an advanced solution for data link encryption at the speed rates of the new Ethernet standard becomes therefore crucial for network security companies. In this section, we report on an efficient AES-GCM architecture for FPGA target applications, that is able to overcome the 100 Gbps throughput and is thus able to fully support the 802.3ba protocol (see [104]).

4.4.1 Hardware Design

The achievement of throughput rates up to 100 Gbps in state-of-the-art FPGA devices is almost impossible with a single AES core in combination with a bit-parallel multiplier computing the tag. Even with modern FPGAs the maximal speed of a standard architecture is limited to 40-50 Gbps [105,106]. In order to support the new Ethernet standard IEEE 802.3ba, we then decided to exploit parallelization and pipelining both in the AES and the GHASH function.

Multi-core AES Design

A block cipher in CTR mode of operation does not require to feed the output of the previous block back to compute the next one. Aiming at speed, this feature allows the insertion of pipeline stages in unrolled implementations. In order to support the three key sizes, the AES architecture relies on 14 unrolled rounds separated by pipeline registers.

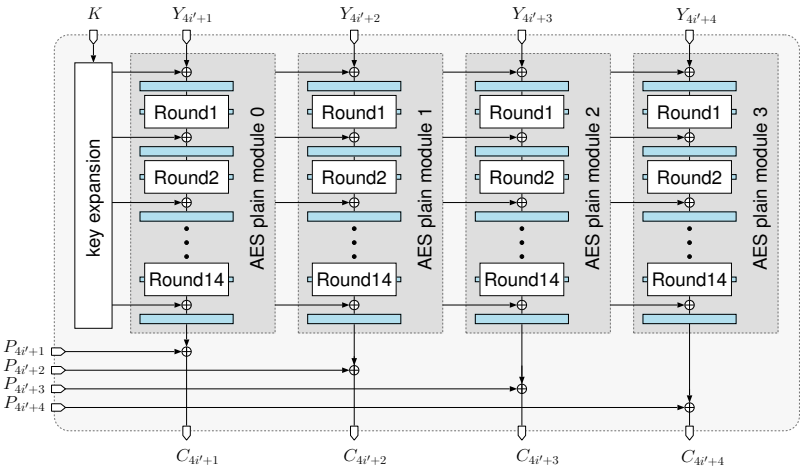

Figure 4.4: Block diagram of the multi-core AES. The blue blocks identify pipeline stages. Each round has an additional internal pipeline stage. Connections are 128-bit wide, $i' = 0, 1, \ldots, \lceil \frac{n}{4} \rceil - 1$.

The strategies proposed in Section 4.2.2 for the realization of the `SubBytes` operation have been all exploited. Furthermore, we decided to add an *in-round* pipeline stage into the composite and the LUT-based approaches, since the use of BRAMs introduces an additional cycle of latency.

Nevertheless, a single pipelined AES is not able to achieve 100 Gbps. This limitation forces the application of a multi-core construction. We instantiated four parallel AES cores sharing the same circuit to perform the key expansion transformation. The four AES plain modules work consequently using the same round keys. Figure 4.4 shows a schematic overview of the main components.

The resulting multi-core design is thus able to process a 512-bit block (4×128 bits) of plaintext at each clock cycle, since the ciphertext is generated by directly XORing the four 128-bit blocks $P_{4i'+1}$, $P_{4i'+2}$, $P_{4i'+3}$, and $P_{4i'+4}$ with the output strings of the four plain modules ($i' = 0, 1, \ldots, \lceil \frac{n}{4} \rceil - 1$). Particular attention has to be taken with the

generation of these outputs. As pointed out in [107], the same counter must not be used twice with the same key. This means that the input counter must not be equal for the four AES modules. This can easily be avoided by fixing with different values two bits of the four input counters $Y_{4i'+1}$, $Y_{4i'+2}$, $Y_{4i'+3}$, and $Y_{4i'+4}$.

The Parallel Pipelined GHASH Design

To combine the authentication core with the multi-core AES, a design solution based on four parallel binary-field multipliers has been investigated. Due to the high-speed requirement, we were forced to further insert pipelining into each multiplier. This is due to the fact that the speed of a plain multiplier is mainly defined by the size of the binary field, in this work the large GF(2^{128}). We adopted the 2-step Karatsuba-Ofman (KO) algorithm and designed a 4-stage pipelined architecture similar to [108]. The overview of the implemented multiplier is depicted in Figure 4.5.

More precisely, the single step KO algorithm splits two m-bit inputs A and B into four terms A_h, A_l, B_h, and B_l, where the index identifies the highest or lowest $\frac{m}{2}$ bits (*split* phase). The result R is the combination of the outputs of three multiplications between these four terms:

$$\begin{aligned} R_l &= A_l B_l \\ R_{hl} &= (A_h + A_l)(B_h + B_l) \\ R_h &= A_h B_h \\ R &= R_h x^m + x^{\frac{m}{2}}(R_h + R_{hl} + R_l) + R_l. \end{aligned} \tag{4.7}$$

The last line computes the final result of the multiplication between A and B, by aligning the intermediates results R_l, R_{hl}, and R_h (*align* phase).

We applied this approach twice recursively in order to reduce a large 128-bit multiplier into nine 32-bit multipliers (see the "2-s mult" region in Figure 4.5). The basic bit-parallel multiplier has then be implemented to carry out the 32-bit multiplication. In this way, the overall complexity of the computation can be decreased allowing the insertion of registers to shorten the longest path. As can be seen in Figure 4.5, a single KO multiplier hosts four pipeline stages. The

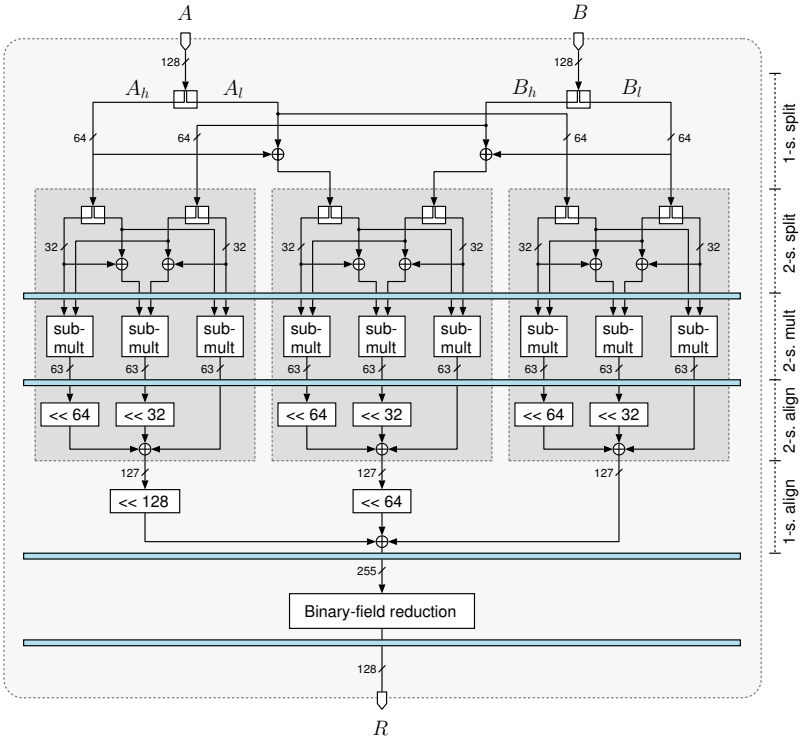

Figure 4.5: Architecture overview of the pipelined 2-step Karatsuba-Ofman multiplier. The blue lines represent the four pipeline stages.

first comes after the *split* phases, the second after the multipliers, while the last two stages isolate the binary-field reduction from the two *align* phases.

To preserve the correct tag computation, the parallelization degree q has then been set to 16 (4 parallelization $\times$ 4 pipelining). This

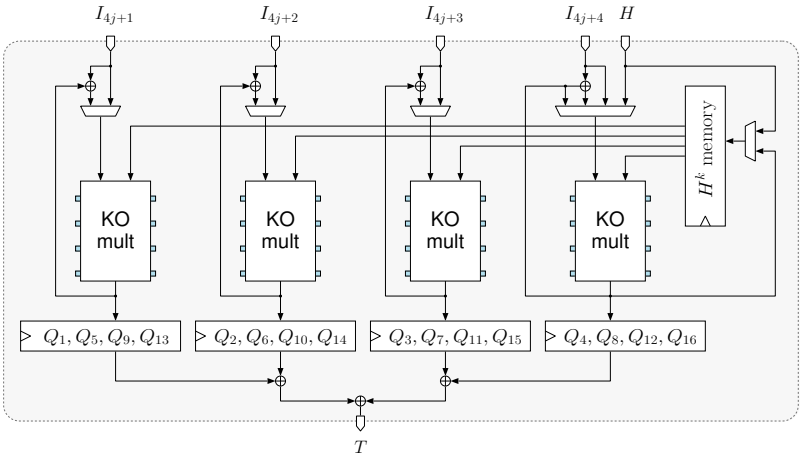

Figure 4.6: Architecture of the parallel authentication core. Connections are 128-bit wide, $j = 0, 1, \ldots, \lceil \frac{m+n+1}{4} \rceil - 1$.

means that the input blocks I_i are processed into 16 independent accumulators:

$$\begin{aligned} Q_{16} &= (((I_1 H^{16} \oplus I_{17}) H^{16} \oplus I_{33}) H^{16} \oplus \ldots) H^{16} \\ Q_{15} &= (((I_2 H^{16} \oplus I_{18}) H^{16} \oplus I_{34}) H^{16} \oplus \ldots) H^{16-1} \\ &\vdots \\ Q_1 &= (((I_{16} H^{16} \oplus I_{32}) H^{16} \oplus I_{48}) H^{16} \ldots) H. \end{aligned} \tag{4.8}$$

The block diagram of the resulting GHASH module is illustrated in Figure 4.6. Each input multiplexer selects the first operand of the multiplication, while the second, i.e., the terms H^k with $1 \leq k \leq 16$, comes from a dedicated memory module. The 16 accumulators Q_i are also stored in register-based memories. The authentication tag T is obtained by XORing the 16 outputs of these memory registers.

Table 4.1: Input pairs of the four multipliers. The flow corresponds to a 288 bytes plaintext with 48 bytes of authenticated data.

Clk	I_{4j+1}	H^k	I_{4j+2}	H^k	I_{4j+3}	H^k	I_{4j+4}	H^k
1	A_1	H^{16}	A_2	H^{16}	A_3	H^{16}	P_1	H^{16}
2	P_2	H^{16}	P_3	H^{16}	P_4	H^{16}	P_5	H^{15}
3	P_6	H^{14}	P_7	H^{13}	P_8	H^{12}	P_9	H^{11}
4	P_{10}	H^{10}	P_{11}	H^9	P_{12}	H^8	P_{13}	H^7
5	P_{14}	H^6	P_{15}	H^5	P_{16}	H^4	P_{17}	H^3
6	P_{18}	H^2	len[a]	H	-	-	-	-

[a] $\text{len}(A)||\text{len}(C)$.

GCM Design

The overall GCM architecture is based on the combination of the multi-core AES and the parallel-pipelined authentication core. The powers of H are computed whenever the key is updated. After the encryption of the 128-bit zeros term, 17 cycles are indeed needed to compute and store inside a dedicated memory the 16 H^k terms. After this task the AES-GCM core is ready to encrypt messages with the new key.

In (4.5), the last multiplication of the accumulators Q_i is done using scaling power terms. This involves the *a priori* knowledge of the total message length by the GCM core. While most of the blocks are multiplied with H^{16}, the last 14 blocks must be multiplied with lower power terms. In Table 4.1, the input pairs of the four multipliers are given for a 18 128-bit blocks plaintext with three 128-bit blocks of authenticated data. Note that the sum of $m+n+1$, i.e., $3+18+1 = 22$, is expressly not a multiple of $q = 16$.

In order to configure the correct H^k inputs, the GCM controller needs to know already at the second cycle that the block P_5 must be multiplied with H^{15}, scaling the powers of H in the next four cycles. This problem is solved by adding a 4-stage buffer at the input of the AES-GCM. The grey module in Figure 4.7 acts like a shift register with controlled output. Thanks to this component, the AES-GCM

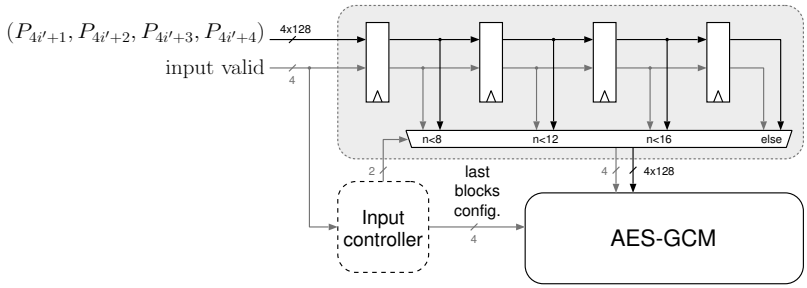

Figure 4.7: Input buffer architecture, $i' = 0, 1, \ldots, \lceil \frac{n}{4} \rceil - 1$.

is informed in advance of the configuration and total length of the message. The output multiplexer is used in case of small messages, i.e., $n < 16$. The input controller selects indeed the correct buffer output, depending on the computed size of the plaintext. In spite of an increase of the total system latency by four cycles, the AES-GCM is able to process data sequentially without having any prior information on the message size.

Because of the pipelined architecture applied in the GHASH core, the correct authentication tag T appears four cycles after the insertion of the last message blocks. In Table 4.1, T would then be generated at the 11th cycle.

4.4.2 Results and Comparison

The parallel-pipelined AES-GCM core has been coded in functional VHDL in three architectures differing by their `SubBytes` implementation (see Section 4.2.2). The three designs have then been synthesized using Synplify Pro, while place and route has been done with the Xilinx ISE Design Suite. Target FPGAs were two Xilinx Virtex-5 chips, i.e., the XC5VLX220 with speed grade -2 for the cores with composite and LUT-based `SubBytes`, and the XC5VSX240T (-2) for the BRAM `SubBytes`. The choice of different FPGAs is motivated by the large amount of required BRAMs in the last design. The 36 Kb

Table 4.2: FPGA performance comparison of the GCM cores. FPGA family is the Xilinx Virtex-5 with speed grade -2.

Ref.	`SubBytes`	Area		Freq.	Thr'put	FPGA
		[Slices]	[BRAM]	[MHz]	[Gbps]	Type
Ours	LUT	14'799	0	233	119.30	LX220
Ours	Comp.	18'505	0	233	119.30	LX220
Ours	BRAM	9'561	450	233	119.30	SX240T
[106]	LUT	5'961	0	296	37.89	LX85
[106]	Comp.	8'077	0	305	39.04	LX85
[106]	BRAM	4'115	59	287	36.74	LX85

BRAMs are indeed dual-port memories that can store only two S-boxes. In total, the multi-core AES requires 450 BRAMs (4×14×16 S-boxes for the rounds and four for the key expansion). Such large amount of BRAMs is only available in bigger Virtex-5 FPGAs like the XC5VSX240T chip. Table 4.2 summarizes the performances and proposes a comparison with the results of [106]. Note that their implementations are based on a single-core pipelined AES-GCM, but are currently the fastest published FPGA design.

We optimized the area of the AES-GCM cores for the same maximal frequency of 233 MHz. The three designs locate indeed their critical path inside the key expansion module of the AES core. This frequency is suitably enough to guarantee the speed requirements imposed by the forthcoming 100G Ethernet standard. Although the design using BRAMs for the `SubBytes` operation consumes less logic, it is penalized by the huge memory demand to store the S-boxes. We point out the core, using the LUT approach. This core fits in 14.8 kslices, about 43 % of the total space available in the XC5VLX220 chip, without requiring additional storing capacity.

Eventually, the composite solution (18.5 kslices) allows the insertion of supplementary pipeline registers in the AES round to further increase the speed of the block cipher. Nevertheless, we implemented a single *in-round* stage since this was enough to reach the target data rate.

Figure 4.8: Two Centauris^2 modules developed in collaboration with IDQ.

4.5 2G Fibre Channel Link Encryptor

The aim of this work was the implementation of a complete bidirectional 2 Gbps FC link encryptor hosting two area-optimized AES-GCM cores for concurrent authenticated encryption and decryption. Measurements in a working network link pointed out that per-packet authentication results in a speed decrease up to 20 % of the channel capacity for a reference frame length of 256 bits. Two methods of frame encryption are therefore investigated to reduce the required GCM data overhead and to exploit different network configurations.

The design of the link encryptor was presented in [109], while the entire system was applied as point-to-point secure communication system within the SwissQuantum network. Carried out in 2009, the SwissQuantum project [7] aimed at the long-term demonstration of QKD and its application in real telecommunication environments. The QKD enhanced FC encryptor, under the name of *Centauris^2* (see Figure 4.8), has been successfully installed and tested in a 3 km node during several months of operation.

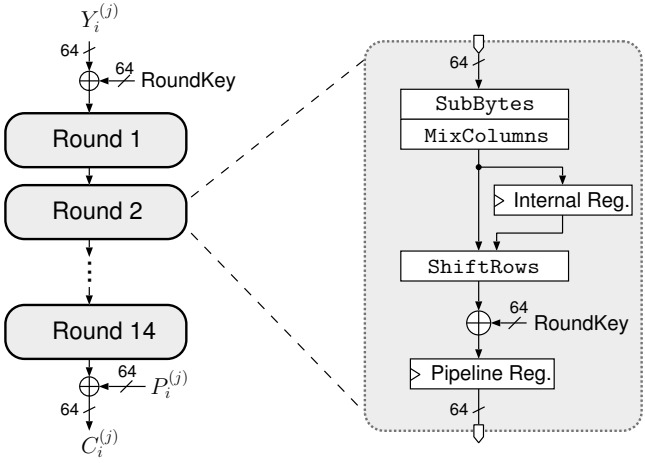

Figure 4.9: Rounds schedule for the pipelined 64-bit datapath AES implementation. Variables are $\{Y_i^{(j)}, P_i^{(j)}, C_{i-m}^{(j)}\}$ with $j = 0$ for the first 64-bit block and $j = 1$ for the last 64-bit block.

4.5.1 AES-GCM Hardware Architecture

The combination of GCM with a pipelined architecture of the AES cipher makes possible the achievement of the gigabit throughput. The hardware implementation hosted into the link encryptor is evaluated and designed to satisfy the maximal rates of the 2G FC standard at minimum hardware costs.

AES Architecture

The demand of a block cipher for high-speed encryption leads to a fully pipelined design of the AES algorithm. The architecture is composed by 14 independent instantiations of the round function block (see Figure 4.9). Each round block includes the succession of the `SubBytes`, `MixColumns`, and `ShiftRows` functions. Although an output block is computed every clock cycle, the fully pipelined AES requires an excessive amount of hardware resources. To reduce the

overall dimension of the encryption core, a 64-bit datapath architecture was designed instead of a conventional 128-bit datapath AES. In this way, a complete round block is calculated within two clock cycles. The shortened datapath halves the hardware utilization: 8 instead of 16 `SubBytes` and two instead of four `MixColumns` are needed.

Since in the target FPGA dual-port memory elements are not a scarce resource, the S-box is stored as LUT in 58 BRAMs. The resulting `SubBytes` function is a fast circuit using no further FPGA resources.

GCM Architecture

The critical computation in the authentication process of GCM is the binary multiplication over the finite field GF(2^{128}). All previously computed values X_{i-1} are XORed with the ciphertext C_i and at the end multiplied by the value H. According to the 64-bit datapath implementation of the AES core, the binary multiplication in the GHASH function processes 64-bit input blocks. Thus, a basic bit-parallel structure of the multiplier has been replaced by an adaptation for the GF(2^{128}) of the bit-parallel word-serial (BPWS) multiplier proposed in [110]. The BPWS multiplier considerably reduces the hardware complexity of the GHASH architecture, without decreasing the speed of the GCM core. The implemented multiplier, illustrated in Figure 4.10, calculates the final value in two clock cycles.

4.5.2 FPGA Implementation

With the aid of high-speed multi-gigabit transceiver (MGT) modules[7], part of the physical layer (PHY) of a FC interface is directly integrated into the FPGA. The MGTs provide 8B/10B encoding scheme[8] and

[7]Xilinx MGTs are special macro blocks integrated in modern Virtex chips. They consists of the physical media attachment (PMA) and physical conding sublayer (PCS) for several high-speed communication protocols. The PMA contains the serializer/deserializer (SERDES), TX and RX I/O buffers, clock generator, and clock recovery circuitry. The PCS contains the 8B/10B encoder/decoder and the ring buffer supporting channel bonding and clock correction.

[8]8B/10B is a line code for the transport digital bits of data across carrier waves. It maps 8-bit symbols in 10-bit symbols to achieve DC-balance and bounded disparity, hence providing enough state change to allow reasonable clock recovery.

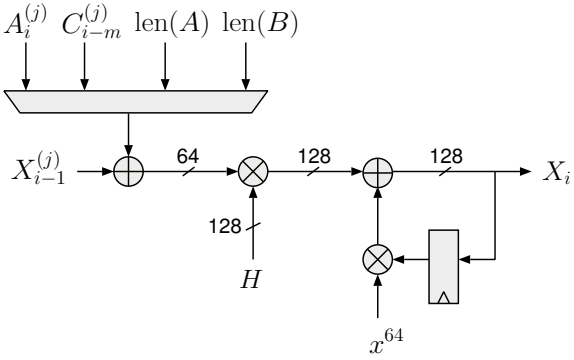

Figure 4.10: BPWS multiplier of the GHASH function. Variables are $\{A_i^{(j)}, C_{i-m}^{(j)}, X_{i-1}^{(j)}\}$ with $j = 0$ for the first 64-bit block and $j = 1$ for the last 64-bit block.

translate serial data from the FC connector to 32-bit parallel data (FC words) inside the FPGA. Transmission words are indeed the lowest level of control in FC, they are defined as four contiguous 8B/10B encoded transmission characters and, most importantly, they are treated as a unique entity (see Figure 4.11).

The interface region of the link encryptor depicted in Figure 4.12 elaborates 32-bit parallelized data at 53.125 MHz to reach the specified 2G FC throughput of 1.7 Gbps, before 8B/10B encoding[9]. The cryptographic core hosting two AES-GCM units for encryption and decryption, is driven by the double frequency of 106.25 MHz in order to allow the additional processing of authentication at full rate.

In the FC standard user data are encapsulated in frames (cf. framing protocol FC-2 [111]). Special delimiters are start of frame (SOF) and end of frame (EOF) words. However, to keep the nodes synchronized, a large number of primitive words and primitive sequences is continuously transmitted through the link. The protocol signals (primitive words and sequences) do not contain any relevant information that needs to be secured, but only information concerning

[9]The 2G FC transmission line speed is indeed 2125 Mbps. The corresponding data rate limit with 8B/10B encoding is therefore 1700 Mbps.

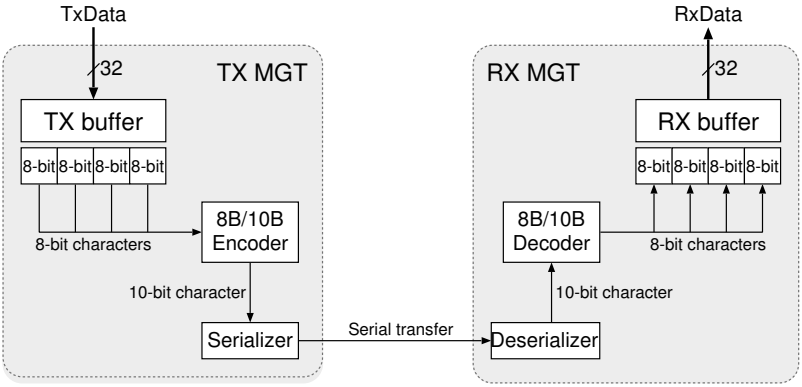

Figure 4.11: SERDES process inside transmitter and receiver MGT.

the status of the nodes. They are transmitted between frames, but are not delimited with SOF and EOF. To work correctly, the nodes of a network must exchange these protocol signals without interruptions. Moreover, for primitive sequences the order of every 32-bit block must be preserved.

In the implemented link encryptor primitive words and primitive sequences are propagated unencrypted (see Figure 4.12). This way the control function of the nodes is able to guarantee the correct link synchronization.

4.5.3 Frame Encryption

Since user data are transferred in the payload field of frames, the designed link encryptor identifies SOF and EOF delimiters in the interface region and propagates the 32-bit blocks of the frame into the cryptographic core.

The first encryption method provides the entire frame as plaintext for the encryption core. The initialization vector IV, the encrypted frame, i.e., the resulting ciphertext, and the tag T are encapsulated by internal SOF and EOF to compose the internal frame (see Figure 4.13A). The assembled frame is then transmitted to the

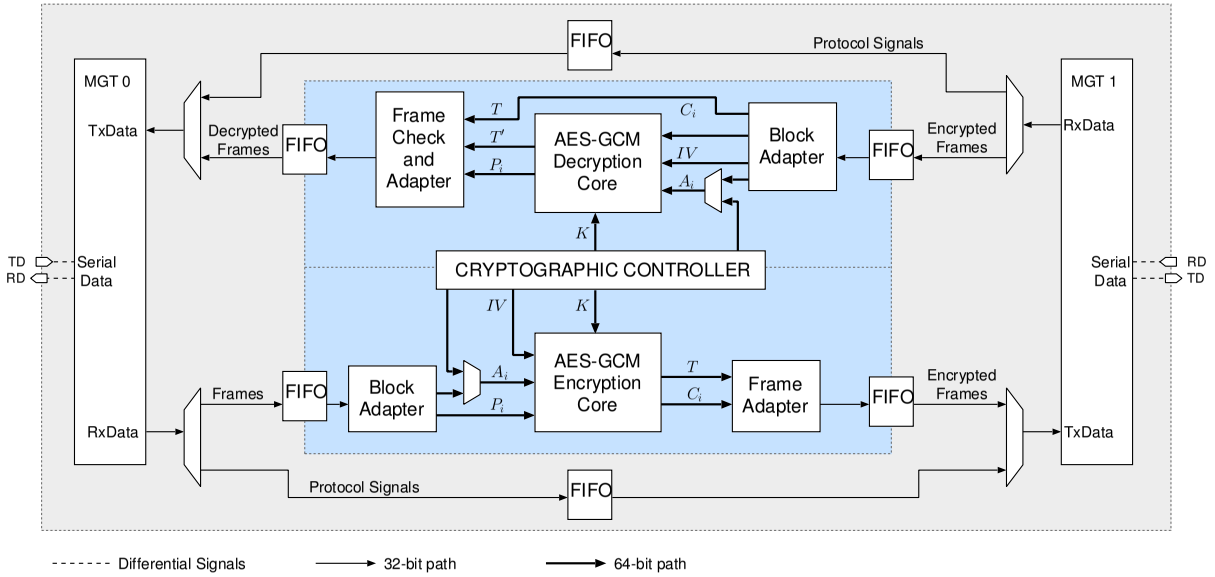

Figure 4.12: Block diagram of the implemented FC link encryptor. The encryption path (bottom) detects incoming frames and sends secured frames through the FC link. The decryption path (top) decrypts the incoming secured frames and transmits the recovered frames to the destination network node. The grey area is the interface region, while the blue corresponds to the cryptographic core.

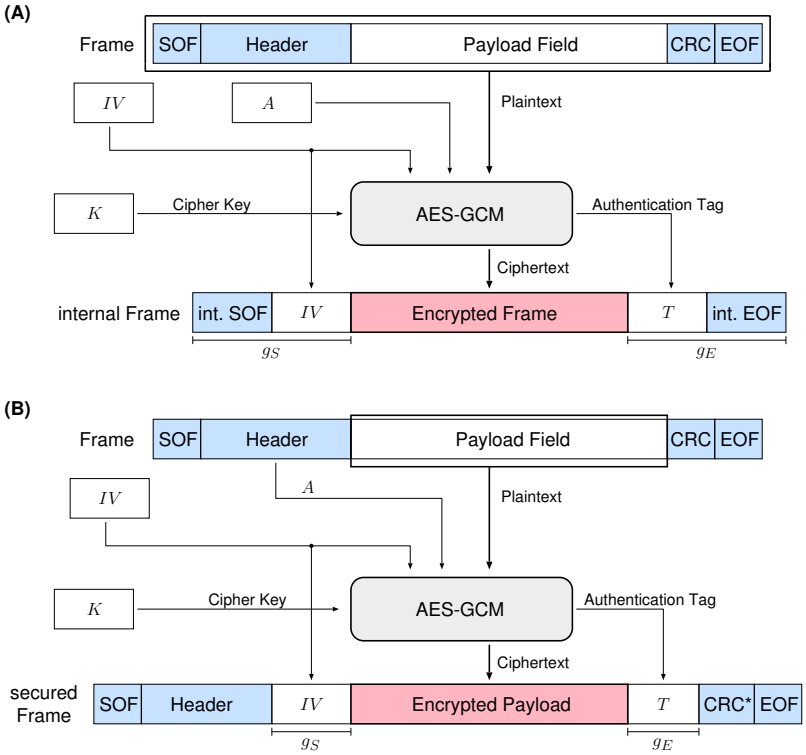

Figure 4.13: Frame encryption diagram for the encryption method: full encryption $\Pi[g_S + g_E]$ (A) and payload encryption $\Psi[g_S + g_E]$ (B). g_S and g_E are the total transmission overhead caused by the per-packet authentication.

other link encryptor. This full encryption mode Π, which is similar to the *tunnel mode* in IPsec, can be applied only in point-to-point network configurations, as the encryption of the header hides the addresses needed for a correct frame routing. Anyway, it provides a higher level of confidentiality for frames, since no information about the content and the source/destination of the frame is visible.

Table 4.3: Resource utilization of the implemented link encryptor. The target application is a Xilinx Virtex-4 FX100 FPGA.

Module	**Slice**	**# BRAM**
AES counter mode	3.1k	58
BPWS multiplier	3.8k	0
GCM core	7.1k	58
Encryption path	0.9k	9
Decryption path	1.1k	14
Link encryptor	19.3k	139

A second encryption method has been therefore implemented to exploit the link encryptor in multi-point networks. The need to preserve the header field of frames unencrypted involves a different approach to secure the incoming frames. In the payload encryption mode Ψ (see Figure 4.13B), only the payload field of the frame is encrypted. As the new payload of the secured frame hosts the IV, the encrypted payload, and the resulting authentication tag T, the cyclic redundancy checksum (CRC) must be re-computed over this updated field. To even improve the efficiency of the link encryptor, the GCM core uses the header field of the incoming frame as additional authenticated data A. The Ψ encryption corresponds to the IPsec *transport mode*.

4.5.4 Results and Discussion

Based on functional VHDL code, the architecture proposed in the previous section has been synthesized using Synplify Pro, placed and routed using the Xilinx ISE Design Suite, and tested in a Xilinx Virtex-4 FX100 FPGA. Table 4.3 resumes the hardware requirements of the principal modules inside the link encryptor.

In the performed tests, a FC traffic generator was connected in a loopback configuration with two link encryptors. Data were transmitted through the devices and then retrieved by the traffic generator. This enabled to transmit frames with variable sizes at different rates.

The results depicted in Figure 4.14 emphasize the effects of the per-packet frame authentication. Using larger frames, the speed of the communication can be kept at the 2G FC limit L, whereas with shorter frames, the overall throughput decreases down to 70 % of L for 128-bit frames. This effect is caused by the extension of the frames due to the appended GCM overhead (g_A and g_E, see Figure 4.13). In the Π configuration, the internal frame exceeds the dimension of the original frame by the internal SOF and EOF, IV, and T. For the Ψ mode the secured frame includes only the necessary IV and T of variable length t, cf. (4.1), therefore 84 % of L for 128-bit frames can be recovered. Nevertheless, the GCM overhead required for the authentication process in both encryption methods has always the same dimension for every frame size. Smaller frames proportionally carry more additional data with respect to larger frames. Thus, the throughput behavior can be predicted by the proportion between incoming frames and internal secured frames, i.e.,

$$\left(\frac{\text{User frame}}{\text{Secured frame}}\right) \text{FC limit} = \left(\frac{S}{S + (g_S + g_E)}\right) L, \tag{4.9}$$

where S is the frame size (g_S and g_E compose the authentication overhead). Figure 4.14 demonstrates the throughput decrease of the FC link for small packets, inherent (4.9). A reduction of the GCM overhead leads to an increase of the performance of the link encryptor, decreasing at the same time the resulting confidentiality level. The difference between measurements and simulations for the point-to-point encryption Π are due to the padding process of frames in 128-bit blocks.

The latency of a complete encryption and decryption cycle is depicted in Figure 4.15. The authentication process of each frame is executed at the end of the decryption when the received tag T and the generated tag T' are compared. For this reason the propagation delay becomes directly proportional to the frame size.

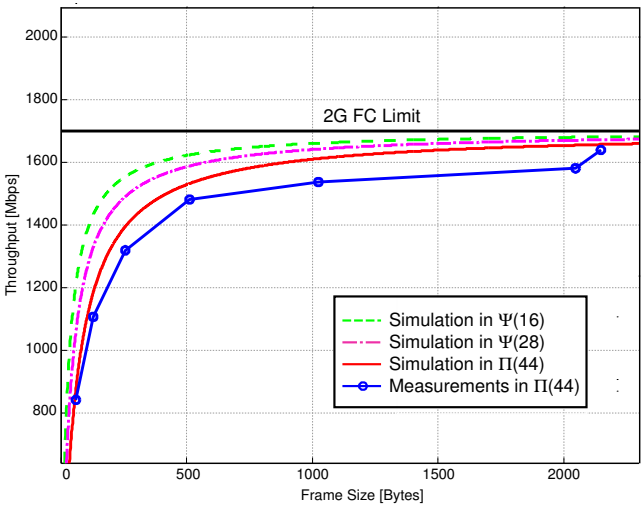

Figure 4.14: Throughput/frame-length trade-off for the link encryptor. Measurements in $\Pi(g_S + g_E = 44\,\text{bytes})$ and simulations in $\Pi(44)$, $\Psi(28)$, and $\Psi(16)$ demonstrate how the throughput for small frames can be increased, reducing the GCM overhead (g_S and g_E).

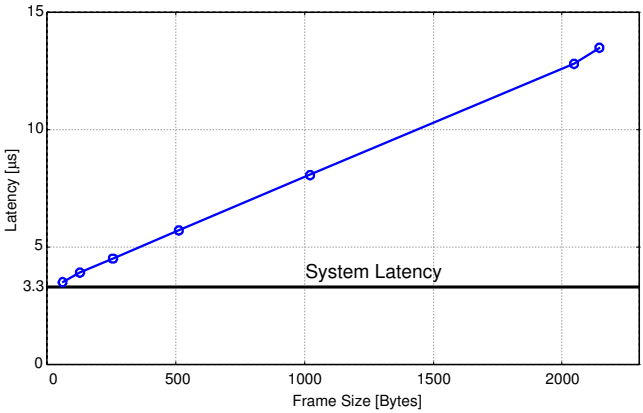

Figure 4.15: Measured latency behavior in $\Pi(44)$ mode. The system latency for internal data propagation is 3.3 µs.

5

Lightweight Hashing

Tiny RFID tags, long used for tracking supplies and inventory, are now appearing in a growing number of consumer items. Their application in automatic payment systems or in official identity documents has created a new set of privacy and security problems [112, 113, 3]. In particular, this embedded circuits impose sever physical limitations for security mechanisms, where standard cryptographic solutions becomes often obsolete and unfeasible. New algorithms that can cope with this limitations must thus be developed.

In this chapter, we give an overview of the most appealing low-cost designs of former cryptographic standards (block ciphers, stream ciphers, and hash functions) and recently designed dedicated algorithms.

Lightweight cryptography must deal with the security/complexity trade-off. To this end, we report of an FPGA implementation of cube testers on the lightweight stream cipher Grain. We argue the possibility of a distinguishing attack on the full algorithm in complexity time below the exhaustive search.

Finally, we describe the design and implementation process of the new hash function family QUARK, particularly advised for constrained environments.

5.1 Lightweight Cryptography Overview

Ubiquitous computing defines new standards for information security. The high-portability of smart devices results in fierce implementation constraints for embedded systems, which are faced with limited resources in terms of memory capacity, computing power, and design costs. In this scenario, cryptography needs to be tailored to completely new design paradigms. We use therefore the generic term *lightweight cryptography* to define the development of new primitives that puts on the same level of importance the security robustness and the implementation costs of the underlying algorithm.

Taking the case of the electronic product code (EPC) class of RFID chips for supply-chain management and product tracking, Juels and Weis report in [114] that current 10'000-GE devices have at most 2'000 gates available for cryptographic components. More drastic is the vision of Sarma et al. in [115], where they foresee an emerging generation of EPC tags that will likely have only between 250 and 1'000 gates available for security features.

In general, asymmetric cryptosystems are computationally far more expensive than symmetric algorithms. Only stream/block ciphers and hash functions can therefore overcome the stringent limitations of ultra-lightweight RFID applications.

Block Ciphers

Being the prominent element in many cryptosystems, block ciphers are the first choice to provide confidentiality in low-cost environments [75]. Starting with the former standard DES, in [116] a compact implementation of 2'310 GE is reported (56-bit key, 0.18 µm CMOS). However, this core and in particular DES is only usable for short-term security (see Section 4.2). Much more interesting is the fabricated 3'400-GE AES chip presented in [70] (128-bit, 0.35 µm). This is so far the smallest reported design of the worldwide approved block cipher.

In the last years, cryptographic researchers developed new block ciphers for low-end applications. Two algorithms stand out for their performance, the PRESENT cipher [117] and the KATAN and KTANTAN family of ciphers [118]. A serial implementation of PRESENT (80-bit, 0.35 µm) is indeed able to fit in 1'000 GE [119], while KATAN (80-bit, 0.13 µm) requires only 802 GE [118].

Stream Ciphers

Stream cipher are by nature more suited for low-size implementations due to the sequential processing of incoming data. With the development effort performed within the eSTREAM Project, two algorithms prevail for their outstanding versatility. Grain [120, 121, 122] and Trivium [123,124] are two hardware-oriented finalists based on FSR. A 1'294-GE architecture of Grain-v1 has been reported in [125] (80-bit, 0.13 µm), while Trivium has been implemented in 2'017 GE in [126] (80-bit, 0.35 µm).

Hash Functions

Recently, several protocols for RFID tag security have been proposed. They often relies on efficient one-way functions[1] to provide authentication. Hash functions becomes therefore the ideal target for such applications. However, as pointed out in [127], most currently used hash functions are optimized for software efficiency rather than for generic hardware performance. Among the widely used hash schemes, no one is indeed able to accomplish the constraints imposed by the RFID technology.

The hash function MD5 (128-bit, 0.35 µm) has been implemented in 8'001 GE in [69], while in [68] a SHA-1 (160-bit, 0.18 µm) core requires 6'122 GE. The smallest design of the current SHA-2 family is reported in [128] and fits in 8'588 GE (256-bit, 0.25 µm). Not even the ongoing NIST hash competition will fill the absence of lightweight dedicated hash functions, since all accepted candidates target general performances as advised by NIST [40]. Most of the first round algorithms can not fit in less than 10'000 GE, with the exception

[1] One-way since collisions resistance is not strictly required in the RFID challenge-response protocol.

of CubeHash that is so far the smallest implemented candidates, i.e., 7'630 GE [56] (512-bit, 0.13 µm). These designs era still too demanding for many low-end environments.

Although the situation has not improved [129], we should consider that most low-cost technologies as RFID or smart cards do not require the same level of security as generic applications. Indeed, a 256-bit security against preimage attacks is somehow disproportionate in devices that require a relaxed security of only 64 or 80 bits[2]; *a fortiori* if the extra security comes at the cost of additional silicon area.

In 2008, two design constructions for compact hashing especially suited for RFID security were proposed. The first approach is the SQUASH MAC designed by Shamir [130]. SQUASH provides 64-bit preimage resistance only (collision is not supported) and is expected to need fewer than 1'000 GE[3]. The second proposal is the set of hash constructions [131] based on the block cipher PRESENT. Interesting is the solution applying PRESENT-80 in the Davies-Meyer model (see Section 3.1.5) for 64-bit hash-values that fits in 1'600 GE in 0.18 µm CMOS; this is so far the smallest implemented hash function available in literature.

5.2 Cube Testers

Cube testers are a particular class of attacks that targets cryptographic algorithms with low algebraic degree equations over GF(2). They become thus a tool of choice to exploit weaknesses in lightweight designs where security (in form of algebraic complexity) is often sacrificed for efficiency.

Cube testers have been developed over previous techniques that can be resumed as "monomial tests" [132, 133, 134] and are strongly related to the key-recovery attacks presented in 2008 by Dinur and Shamir under the name of cube attacks [135, 63].

The peculiarity of cube tester and cube attacks is the black-box interaction with the cryptographic algorithm. They can thus be applied on a completely secret and unknown function. More specifically, cube

[2]For example, the target key size of the stream ciphers in the eSTREAM competition has been set to 80 bits.

[3]Rough estimate based on the results of the stream cipher Grain-128 in [125].

testers aim to detected nonrandomness in cryptographic primitives via multiple queries with chosen values of the *public variables*[4]. Cube testers are then classified as distinguishing attacks rather than key-recovery attacks, as they identify non-optimal functions from PRFs.

Cube testers were introduced in [8], where they were applied against reduced versions of the compression function of MD6 (a first round candidate at the SHA-3 competition designed by Rivest, the author of MD5) and the stream cipher Trivium. In [136], we implemented an FPGA design that was able to improve the efficacy of cube tester on the stream cipher Grain-128.

5.2.1 Theoretical Background

In this section, we briefly explain the principles behind cube testers, and we describe the type of cube testers used for attacking Grain-128. More details can be found in [8], and in the article introducing (key-recovery) cube attacks [63].

An important observation regarding cube testers is that for any function $f : \{0,1\}^n \mapsto \{0,1\}$, the sum (XOR) of all entries in the truth table

$$\sum_{x \in \{0,1\}^n} f(x) \tag{5.1}$$

equals the coefficient of the highest degree monomial $x_1 \cdots x_n$ in the algebraic normal form (ANF) of f.

For a stream cipher, one may consider as f the function mapping the key and the IV bits to the first bit of keystream. Obviously, evaluating f for each possible key/IV and XORing the values obtained yields the coefficient of the highest degree monomial in the implicit algebraic description of the cipher.

Instead, cube attacks work by summing $f(x)$ over a *subset* of its inputs. For example, if $n = 4$ and

$$f(x) = f(x_1, x_2, x_3, x_4) = x_1 + x_1x_2x_3 + x_1x_2x_4 + x_3, \tag{5.2}$$

[4]In stream ciphers, public variables are the IV bits, while in block ciphers the plaintext bits.

then summing over the four possible values of (x_1, x_2) yields

$$\sum_{(x_1,x_2)\in\{0,1\}^2} f(x_1, x_2, x_3, x_4) = 4x_1 + 4x_3 + (x_3 + x_4) \equiv x_3 + x_4, \tag{5.3}$$

where $(x_3 + x_4)$ is the factor of x_1x_2 in f:

$$f(x_1, x_2, x_3, x_4) = x_1 + x_1x_2(x_3 + x_4) + x_3. \tag{5.4}$$

Indeed, when x_3 and x_4 are fixed, then the maximum degree monomial becomes x_1x_2 and its coefficient equals the value $(x_3 + x_4)$. In the terminology of cube attacks, the polynomial $(x_3 + x_4)$ is called the *superpoly* of the *cube* x_1x_2. Cube attacks work by detecting linear superpolys, and then explicitly reconstructing them via probabilistic linearity tests [137].

Assume that we have a function $f(k_0, \ldots, k_{127}, v_0, \ldots, v_{95})$ that, given a key k and an IV v, returns the first keystream bit produced by Grain-128. For a fixed key $k_0, \ldots, k_{127}$, the sum

$$\sum_{(v_0,\ldots,v_{95})\in\{0,1\}^{96}} f(k_0, \ldots, k_{127}, v_0, v_{95})$$

gives the evaluation of the superpoly of the cube $v_0v_1 \cdots v_{95}$. More generally, one can fix some IV bits, and evaluate the superpoly of the cube formed by the other IV bits (then called the *cube variables*, or CV). Ideally, for a random key, this superpoly should be a uniformly distributed random polynomial. However, when the cipher is constructed with components of low degree, and sparse algebraically, this polynomial is likely to have some property which is efficiently detectable. More details about cube attacks and cube testers can be found in [8, 63].

In our tests below, we measure the *balance* of the superpoly over 64 instances with distinct random keys.

5.2.2 Description of Grain-128

The stream cipher Grain-128 was proposed by Hell et al. [138] as a variant of Grain-v1, to accept keys of up to 128 bits, instead of up

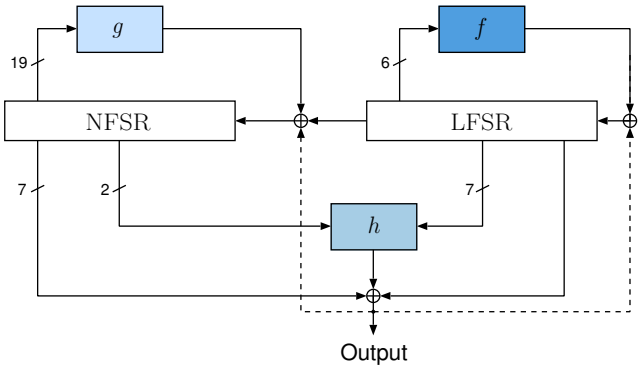

Figure 5.1: Schematic view of Grain-128's keystream generation mechanism (if not written the width of connections is one bit). During initialization, the output bit is fed back into both feedback registers (dotted lines), i.e., added to the output of f and g.

to 80 bits. Grain-v1 has been selected in the eSTREAM portfolio of promising stream ciphers for hardware and Grain-128 was expected to retain the merits of Grain-v1.

Grain-128 takes as input a 128-bit key and a 96-bit IV, and it produces a keystream after 256 rounds of initialization. Each round corresponds to clocking two feedback registers, i.e., a 128-bit LFSR and a 128-bit non-linerar feedback shift register (NFSR) (both over GF(2)). The feedback polynomial g of the NFSR has algebraic degree two, while the additional Boolean function h has degree three. An overview of the Grain-128 structure is depicted in Figure 5.1.

Given the key and the IV, one initializes Grain-128 by filling the NFSR with the key, and the LFSR with the IV padded with 1 bits. The mechanism is then clocked 256 times without producing output, and feeding the output of h back into both registers. Details can be found in [138].

Several attacks on Grain-128 were reported: [132] claims to detect nonrandomness on up to 313 rounds, but these results were not documented, and not confirmed by [133], which used similar methods to find a distinguisher on 192 rounds. Shortcut key-recovery attacks

on 180 rounds were presented in [139], while [140] exploited a sliding property to speed-up exhaustive search by a factor two. In conclusion, no attack significantly faster than brute-force is known for the complete Grain-128 in the standard attack model.

5.2.3 Software Implementation

To test small cubes we used a bitsliced implementation [141] of Grain-128 that runs 64 instances of Grain-128 in parallel, each with (potentially) different keys and different IVs. We stored the internal states of the 64 instances in two arrays of 128 words of 64 bits, where each bit slice corresponds to an instance of Grain-128, and the i-th word of each array contains the i-th bit in the LFSR (or NFSR) of each instance.

Our bitsliced implementation provides a considerable speedup compared to the reference implementation of Grain-128. For example, on a PC with an Intel Core 2 Duo processor, evaluating the superpoly of a cube of dimension 30 for 64 distinct instances of Grain-128 with a bitsliced implementation takes approximately 45 minutes, against more than a day with the designers' C implementation.

5.2.4 Hardware Implementation

FPGAs are reconfigurable hardware devices widely used in the implementation of cryptographic systems for high-speed or area-constrained applications. The possibility to reprogram the designed core makes FPGAs an attractive evaluation platform to test the hardware performances of selected algorithms. During the eSTREAM competition, many of the candidate stream ciphers were implemented and evaluated on various FPGAs [142,143]. Especially for Profile 1 (HW), the FPGA performance in terms of speed, area, and flexibility was a crucial criterion to identify the most efficient candidates.

To attack Grain-128, we used a Xilinx Virtex-5 LX330 FPGA to run the first reported implementation of cube testers in hardware. This FPGA offers a large number of embedded programmable logic blocks, memories and clock managers, and is an excellent platform for large scale parallel computations.

Note that FPGAs have already been used for cryptanalytic purposes, most remarkably with COPACOBANA [144, 145], a machine with 120 FPGAs that can be programmed for exhaustive search of small keys, or for parallel computation of discrete logarithms.

Implementation of Grain-128

The Grain ciphers (Grain-128 and Grain-v1) are particularly suitable for resource-limited hardware environments. Low-area implementations of Grain-v1 are indeed able to fill just a few slices in various types of FPGAs [146]. Using only shift registers combined with XOR and AND gates, the simplicity of the Grain's construction can also be easily translated into high-speed architectures. Throughput and circuit efficiency are indeed the two main characteristics that have been used as guidelines to design our Grain-128 module for the Virtex-5 chip. The relatively small degree of optimization for Grain allows the choice of different datapath widths, resulting in the possibility of a speedup by a factor 32 (see [138]).

We selected a 32-bit datapath to get the fastest and most efficient design in terms of area and speed. Figure 5.2 depicts our module, where both sides of the diagram contain four 32-bit register blocks. During the setup cycle, the key and the IV are stored inside these memory blocks. In normal functioning, they behave like shift register units, i.e., at each clock cycle the 32-bit vectors stored in the lower blocks are sent to the upper blocks. For the two lowest register blocks (indices between 96 and 127), the input vectors are generated by specific functions, according to the algorithm definition. The g' module executes the same computations of the function g plus the addition of the smallest index coming from the LFSR, while the output bits are entirely computed inside the h' module. Table 5.1 summarizes the overall structure of our 32×Grain-128 architecture.

As described in [138], Grain-128 schedules a key and an IV initialization phase, before starting to generate the keystream. During keystream generation, the output bits are fed back and XORed with the input of the bigger register block inside NFSR and LFSR. In our implementation, where only the first bit of the output should be collected, the feedback procedure is not required. This allows us to

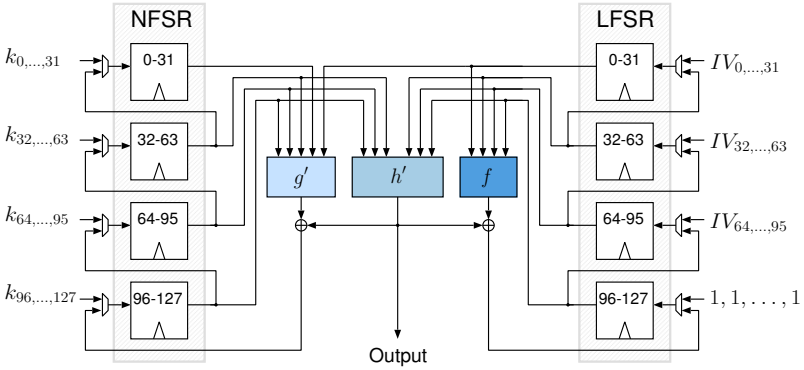

Figure 5.2: Overview of our Grain-128 architecture. At the beginning of the simulation, the key and the IV are directly stored in the NFSR and LFSR register blocks. All connections are 32-bit wide.

Table 5.1: Performance results of our Grain-128 implementation.

	Frequency [MHz]	**Throughput** [Mbps]	**Size** [Slices]	**Available area** [Slices]
Grain-128	200	6'400	180	51'840

constantly compute the first output bit for an increasing number of initialization rounds.

Implementation of Cube Testers

Besides the intrinsic speed improvement from software to hardware implementations of Grain-128, the main benefit resides in the possibility to parallelize the computations of the IV queries necessary for the cube tester. With 2^m instances of Grain-128 in parallel, running a cube tester with a $(n+m)$-dimensional cube will be as fast as with an n-dimensional cube on a single instance.

In addition to the array of Grain-128 modules, we designed three other components: the first provides the pseudorandom key and the

2^n IVs for each instance, the second collects and sums the outputs, and the last component is a controller unit. Figure 5.3 illustrates the architecture of our cube tester implementation fitted in a Virtex-5 chip. No special macro blocks has been used, we just tried to exploit all the available space to fit the largest Grain-128 array. Below we describe the mode of operation of each component:

- **Simulation controller**: This unit manages the I/O interface to control the cube tester core. Through the signal `s_inst`, a new instance is started. After the complete evaluation of the cube over the Grain-128 array, the `u_inst` signal is asserted and later a new instantiation with a different key is started. This operation mode works differently from the software implementation, where the 256 instances run in parallel.

- **Input generator**: After each run of the cipher array, the (n-m)-bit partial IV is incremented by one. This vector is then combined with different m-bit offset vectors to generate the 2^m IVs. The key distribution is also managed here. A single key is given to the parallel Grain-128 modules and is updated only when the partial IV is equal to zero.

- **Output collector**: The outcoming 32-bit vectors from the parallel Grain-128 modules are XORed, and the result is XORed again with the intermediate results of the previous runs. The updated intermediate results are then stored until the `u_inst` signal is asserted. This causes a reset of the 32-bit intermediate vector and an update of an internal counter.

The m-bit binary representations of the numbers in $0, \ldots, 2^m - 1$ are stored in offset vectors. These vectors are given to the Grain-128 modules as the last cube bits inside the IV. The correct allocation of the CV bits inside the IV is performed by the CV routers. These blocks take the partial IV and the offset vectors to form a 96-bit IV, where the remaining bits are set to zero. When the cube is updated, the offset bits are reallocated, varying the composition of the IVs.

In the input generator, the key is also provided by a LFSR with (primitive) feedback polynomial $x^{128} + x^{29} + x^{27} + x^2 + 1$. This guarantees a period of $2^{128} - 1$, thus ensuring that no key is repeated.

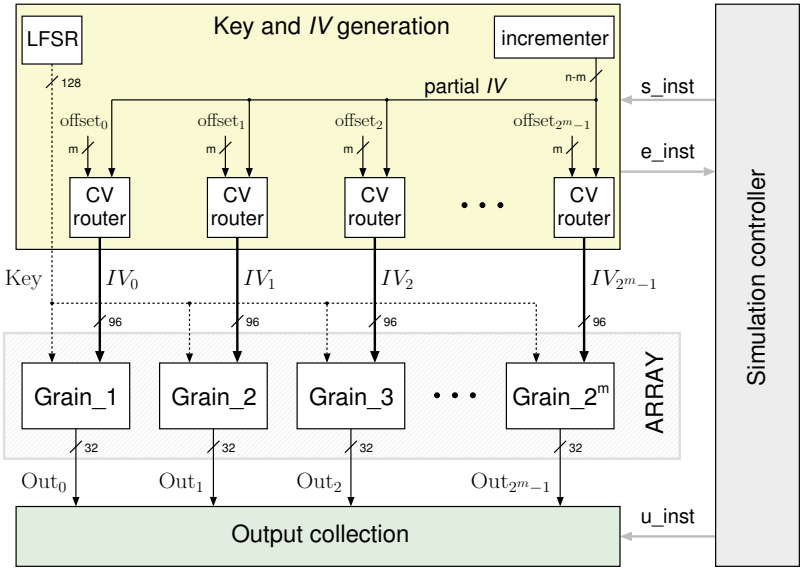

Figure 5.3: Architecture of the FPGA cube module. The width of all signals is written out, except for the control signals in grey.

The evaluation of the superpoly for all 256 instances with different pseudorandom keys is performed inside the output collection module. After the 2^{n-m} queries, the intermediate vector contains the final evaluation of the superpoly for a single instance. The implementation of a modified Grain-128 architecture with $\times 32$ speedup allows us to evaluate the same cube for 32 subsequent rounds. That is, after the exhaustive simulation of all possible values of the superpoly, we get the results for the same simulation done with an increasing number of initialization rounds r, $32i \leq r < 32(i+1)$ and $i \in [1, 7]$. This is particularly useful to test the maximal number of rounds attackable with a specific cube (we do not have to run the same initialization rounds 32 times to test 32 distinct round numbers).

Finally, 32 dedicated counters are incremented if the values of the according bit inside the intermediate result vector is zero or one, respectively. At the end of the repetitions, the counters indicate the

Table 5.2: FPGA evaluation time for cubes of different dimension with $2^m = 2^8$ parallel Grain-128 modules. Note that detecting nonrandomness requires the calculation of statistics on several trials, e.g., our experiments involved 64 trials with a 40-bit cube.

Cube dimension	30	35	37	40	44	46	50
Nb. of queries	2^{22}	2^{27}	2^{29}	2^{32}	2^{36}	2^{38}	2^{42}
Time	0.17 sec	5.4 sec	21 sec	3 min	45 min	3 h	2 days

proportion between zeros and ones for 32 different values of increasing rounds. This proportion vector can be constantly monitored using an I/O logic analyzer.

Since the required size of a single Grain-128 core is 180 slices, up to 256 parallel ciphers can be implemented inside a Virtex-5 LX330 chip (cf. Table 5.1). This gives $m = 8$, hence decreasing the number of queries to 2^{n-8}. Table 5.2 presents the evaluation time for cubes up to dimension 50. The critical path has been kept inside the Grain-128 modules, so the working frequency of the cube machine is 200 MHz.

Estimate for an ASIC Implementation

The utilization of ASIC is a further solution to enhance the performances of cube testers on Grain-128. Like in the FPGA, several parallel cipher modules should run at the same time, decreasing the evaluation period of a cube. Using the ASIC results presented in [146, 143], we can estimate a speed increase up to 400 MHz for a 90 nm CMOS technology. Evaluating a related area cost of about 10 kGE for a single Grain-128 module (broad estimate), we can take into account a single chip design of 4 mm×4 mm size, hosting the same number of Grain-128 elements of 256. This leads to a similar ASIC cube tester implementation, which is able to compute a cube in half the time of the FPGA. However, in this rough estimate we omitted several problematics related to ASIC design, like the expensive fabrication costs or the development of an interface to communicate the cube indices inside the chip. In conclusion, we still believe that FPGAs

Table 5.3: Best results for various cube dimensions on Grain-128.

Cube dimension	6	10	14	18	22	26	30	37	40
Rounds	180	195	203	208	215	222	227	233	237

are the most suitable tool to enhance cryptanalytic attacks exploiting hardware implementations.

5.2.5 Experimental Results

Table 5.3 summarizes the maximum number of initialization rounds after which we can detect imbalance in the superpoly corresponding to the first keystream bit. It follows that one can mount a distinguisher for 195-round Grain-128 in time 2^{10}, and for 237-round Grain-128 in time 2^{40}. The cubes used are given in Table 5.4.

Extrapolation

We used standard methods to extrapolate our results, using the generalized linear model fitting of the Matlab tool. We selected the Poisson regression in the "log" value, i.e., logarithm as canonical function and the Poisson distribution, since the achieved results suggested a logarithmic behavior between cube size and number of round. The obtained extrapolation, depicted on Figure 5.4, suggests that cubes of dimension 77 may be sufficient to construct successful cube testers on the full Grain-128, i.e., with 256 initialization rounds.

If this extrapolation is correct, then a cube tester with $64 \times 2^{77} = 2^{83}$ chosen-IV queries can distinguish the full Grain-128 from an ideal stream cipher, against 2^{128} ideally. We add the factor 64 because our extrapolation is done with respect to results obtained with statistic over 64 random keys. That complexity excludes the precomputation required for finding a good cube; based on our experiments with 40-dimensional cubes, less than 2^5 trials would be sufficient to find a good cube (based on the finding of good small cubes, e.g., using the evolutionary algorithm described in [136]). That is, precomputation would be less than 2^{88} initializations of Grain-128.

Table 5.4: Cubes used for Grain-128.

Cube dimension	Indices
6	33, 36, 61, 64, 67, 69
10	5, 28, 34, 36, 37, 66, 68, 71, 74, 79
14	5, 28, 34, 36, 37, 51, 53, 54, 56, 63, 66, 68, 71, 74
18	5, 28, 30, 32, 34, 36, 37, 62, 63, 64, 65, 66, 67, 68, 69, 71, 73, 74
22	4, 5, 28, 30, 32, 34, 36, 37, 51, 62, 63, 64, 65, 66, 67, 68, 69, 71, 73, 74, 79, 89
26	4, 7, 20, 22, 25, 28, 30, 31, 33, 36, 39, 40, 41, 51, 53, 54, 56, 57, 61, 62, 63, 64, 65, 66, 67, 68
30	4, 7, 20, 22, 25, 28, 30, 31, 33, 36, 39, 40, 41, 51, 53, 54, 56, 57, 59, 62, 65, 66, 69, 72, 75, 78, 79, 80, 83, 86
37	4, 7, 12, 14, 20, 22, 25, 28, 30, 31, 33, 36, 39, 40, 41, 51, 53, 54, 56, 57, 61, 62, 63, 64, 65, 66, 67, 68, 74, 75, 76, 77, 78, 79, 89, 90, 91
40	4, 7, 12, 14, 20, 22, 25, 28, 30, 31, 33, 36, 39, 40, 41, 51, 53, 54, 56, 57, 61, 62, 63, 64, 65, 66, 67, 68, 74, 75, 76, 77, 78, 79, 86, 87, 88, 89, 90, 91

5.2.6 Conclusion

We developed and implemented a hardware cryptanalytical device for attacking the stream cipher Grain-128 with cube testers (which give distinguishers rather than key recovery). We were able to run our tests on 256 instances of Grain-128 in parallel, each instance being itself parallelized by a factor 32. The heaviest experiment run involved about 2^{54} clockings of the Grain-128 mechanism.

To find good parameters for our experiments in hardware, we first ran light experiments in software with a dedicated bitsliced implementation of Grain-128, using a simple evolutionary algorithm (see full version of [136]). We were then able to attack reduced versions of Grain-128 with up to 237 rounds. An extrapolation of our results

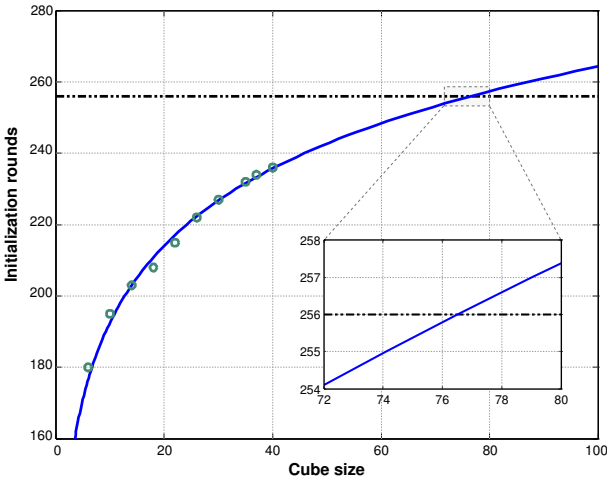

Figure 5.4: Extrapolation of our cube testers on Grain-128, obtained by general linear regression using the Matlab software, in the "poisson-log" model. The required dimension for the full Grain-128 version is 77 (see zoom).

suggests that the full Grain-128 can be attacked in time 2^{83} instead of 2^{128} ideally. Therefore, Grain-128 may not provide full protection when 128-bit security is required.

5.3 The Quark Hash Function

Facing the lack of a reliable hash function with limited silicon requirements, we started the development of a new hash family specifically designed for lightweight applications (see [147,148]). As noted in [131, §2], designers of lightweight algorithms or protocols have to trade off between two opposite design philosophies: the first consists in creating new schemes from scratch, and the second consists in reusing available schemes and adapting them to system constraints. While in [131], the authors are more in line with the latter approach (as illustrated by their DM-PRESENT proposal) we tend more towards the former, for we

believe that lightweight hash functions require dedicated components in every part of the design, as discussed below.

The introduction of the sponge construction [33] was a first step towards the distinction between the hash-value length and the security level of hash functions. This led to more original designs as showed by the hash family RadioGatún [149]. Designers may thus relax the security requirements against (second) preimages, as recently suggested by several researchers in the context of the SHA-3 competition, so as to propose more efficient algorithms. For this, we used a sponge construction and target a single security level against preimage attacks, collision attacks, and any differentiating attack.

Second, we opted for an algorithm based on shift registers combined with Boolean functions, rather than a S-box-based approach; the former indeed tends to perform significantly better, and to lead to simpler designs. Since good shift register-based algorithms are known, we preferred not to reinvent the wheel and propose a core algorithm inspired from the stream cipher family Grain [120, 122] and from the block cipher family KATAN [118], which are arguably the lightest known secure stream cipher and block cipher. Although both these designs are inappropriate for direct reuse in a hash function, they contain excellent design ideas, which we integrate to our lightweight hash Quark. A goal of this best-of-both approach is to build on solid bases, while adapting the algorithm to the attack model of a hash function.

To summarize, our approach is not to instantiate classical general-purpose constructions with lightweight components, but rather to make the whole design lightweight by optimizing all its parts: security level, construction, and core algorithm. An outcome of this design philosophy, the hash family Quark, is described in the next section.

5.3.1 Description of the Quark hash family

Sponge construction

Quark uses the sponge construction, described in Section 3.1.4. Following the notations introduced in [33], it is parametrized by a *rate* (or block length) r, a *capacity* c, and the *output length* n. The *width* is the size of its internal state $b = r + c \geq n$.

Given an initial state, the QUARK iteration mode processes a message m as follows:

1. **Initialization**: the message is padded by appending a '1' bit and sufficiently many zeroes to reach length a multiple of r.
2. **Absorbing phase**: the r-bit message blocks are XORed into the last r bits of the state (i.e., $Y_{b/2-r}, \ldots, Y_{b/2-1}$), interleaved with applications of the permutation P.
3. **Squeezing phase**: the last r bits of the state are returned as output, interleaved with applications of the permutation P, until n bits are returned.

Permutation

QUARK uses a permutation P inspired by the stream cipher Grain and by the block cipher KATAN, as depicted in Figure 5.5.

The permutation P relies on three non-linear Boolean functions f, g, and h, on a linear Boolean function p, and on an internal state composed, at epoch $t \geq 0$, of

- an NFSR X of $b/2$ bits set to $X^t = (X_0^t, \ldots, X_{b/2-1}^t)$;
- an NFSR Y of $b/2$ bits set to $Y^t = (Y_0^t, \ldots, Y_{b/2-1}^t)$;
- an LFSR L of $\lceil \log 4b \rceil$ bits set to $L^t = (L_0^t, \ldots, L_{\lceil \log 4b \rceil - 1}^t)$.

P processes a b-bit input in three stages, as described below:

Initialization Upon input $s = (s_0, \ldots, s_{b-1})$, P initializes its internal state as follows:

- X is initialized with the first $b/2$ input bits:
$$(X_0^0, \ldots, X_{b/2-1}^0) := (s_0, \ldots, s_{b/2-1});$$
- Y is initialized with the last $b/2$ input bits:
$$(Y_0^0, \ldots, Y_{b/2-1}^0) := (s_{b/2}, \ldots, s_{b-1});$$
- L is initialized to the all-one string:
$$(L_0^0, \ldots, L_{\lceil \log 4b \rceil - 1}^0) := (1, \ldots, 1).$$

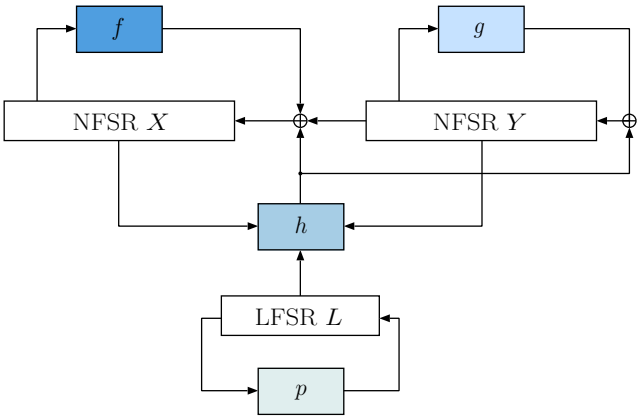

Figure 5.5: Diagram of the permutation of QUARK (for clarity, the feedback of the LFSR with the function p is omitted).

State update From an internal state (X^t, Y^t, L^t), the next state $(X^{t+1}, Y^{t+1}, L^{t+1})$ is determined by *clocking* the internal mechanism as follows:

1. The function h is evaluated upon input bits from X^t, Y^t, and L^t, and the result is written h^t: $h^t := h(X^t, Y^t, L^t)$;

2. X is clocked using Y_0^t, the function f, and h^t:

$$(X_0^{t+1}, \ldots, X_{b/2-1}^{t+1}) := (X_1^t, \ldots, X_{b/2-1}^t, Y_0^t + f(X^t) + h^t);$$

3. Y is clocked using the function g and h^t:

$$(Y_0^{t+1}, \ldots, Y_{b/2-1}^{t+1}) := (Y_1^t, \ldots, Y_{b/2-1}^t, g(Y^t) + h^t);$$

4. L is clocked using the function p:

$$(L_0^{t+1}, \ldots, L_{\lceil \log 4b \rceil}^{t+1}) := (L_1^t, \ldots, L_{\lceil \log 4b \rceil - 1}^t, p(L^t)).$$

Table 5.5: QUARK instances.

	Security	Parallelization	Sponge numbers				Rounds
			r	c	b	n	
u-QUARK	64	8	8	128	2×68	128	512
d-QUARK	80	8	16	160	2×88	160	704
t-QUARK	112	16	32	224	2×128	224	1024

Computation of the output Once initialized, the state of QUARK is updated $4b$ times, and the output is the final value of the NFSRs X and Y, using the same bit ordering as for the initialization.

Proposed instances

There are three different flavors[5] of QUARK: u-QUARK, d-QUARK, and t-QUARK. Table 5.5 gives an overview of the three QUARK instances and summarizes the sponge parameters (rate r, capacity c, width b, digest length n). For a complete specification of the functions f, g, and h consult [147].

5.3.2 Hardware implementation

This section reports our hardware implementation of the QUARK instances. Note that QUARK is not optimized for software (be it 64- or 8-bit processors), and other types of designs are preferable for such platforms. We thus focus on hardware efficiency.

Our results arise from pure simulations, and they are not supported by real measurements on a fabricated chip. However, we believe that this evaluation gives a fair and reliable overview of the overall VLSI performance of QUARK.

[5]In particle physics, the u-quark is lighter than the d-quark, which itself is lighter than the t-quark; our eponym hash functions compare similarly.

Architectures

Three characteristics make QUARK particularly attractive for lightweight hashing: first, the absence in its sponge construction of "feedforward" values, which normally would require additional dedicated memory components; second, its use of shift registers, which are extremely easy to implement in hardware; and third, the possibility of several space/time implementation trade-offs. Based on the two extremal trade-off choice, we designed two architecture variants of u-QUARK, d-QUARK, and t-QUARK:

- **Serial**: Only one permutation module, hosting the circuit for the functions f, g, and h, is implemented. Each clock cycle, the bits of the registers X, Y, and L are shifted by one. These architectures corresponds to the most compact designs. They contain the minimal circuitry needed to handle incoming messages and to generate the correct output digests.
- **Parallel**: The number of the implemented permutation modules corresponds to the parallelization degree given in Table 5.5. The bits in the registers are accordingly shifted. These architectures increase the number of rounds computed per cycle, and therefore the throughput, at extra area costs.

In addition to the three FSRs, each design has a dedicated controller module that handles the sponge process. This module is made up of a finite-state machine and of two counters for the round and the output digest computation. After processing all message blocks during the absorbing phase, the controller switches automatically to the squeezing phase (computation of the digest), if no further r-bit message blocks are given. This implies that the message has been externally padded.

Methodology

We described the serial and parallel architectures of each QUARK instance in functional VHDL, and synthesized the code with Synopsys Design Vision-2009.06 targeting the UMC 0.18 µm 1P6M CMOS technology with the FSA0A_C cell library from Faraday Technology Corporation. We used the generic process (at typical conditions),

instead of the low-leakage for two reasons: first the leakage dissipation is not a big issue in 0.18 µm CMOS, and second, for such small circuits the leakage power is about two orders of magnitude smaller than the total power. To provide a thorough and more reliable analysis, we extended the implementation up to the back-end design. Place and route have been carried out with the help of Cadance Design Systems Velocity-9.1. In a square floorplan, we set a 98 % row density, i.e., the utilization of the core area. Two external power rings of 1.2 µm were sufficient for power and ground distribution. In this technology six metal layers are available for routing. However, during the routing phase, the fifth and the sixth layers were barely used. The design flow has been placement, clock tree synthesis, and routing with intermediate timing optimizations.

Each architecture was implemented at the target frequency of 100 kHz. As noted in [131, 118], this is a typical operating frequency of cryptographic modules in RFID systems. Power simulation was measured for the complete design under real stimuli simulations (two consecutive 512-bit messages) at 100 kHz. The switching activity of the circuit's internal nodes was computed generating VCD files. These were then used to perform statistical power analysis in the velocity tool. Besides the mean value, we also report the peak power consumption, which is a limiting parameter in RFID systems ([69] suggests a maximum of 27 µW). Table 5.6 reports the performance metrics obtained from our simulations at 100 kHz.

To give an overview of the best speed achievable, we also implemented the parallel architectures increasing the timing constraints (see Table 5.7).

5.3.3 Discussion

As reported in Table 5.6, the three serial designs need fewer than 2'300 GE, thus making 112-bit security affordable for restricted-area environments. Particularly appealing for ultra-compact applications is the u-Quark function, which offers 64-bit security but requires only 1'379 GE and dissipates less than 2.5 µW. To the best of our knowledge, u-Quark is lighter than all previous designs with comparable security claims. We expect an instance of Quark with 256-bit security (e.g., with $r = 64$, $c = 512$) to fit in 4'500 GE.

Table 5.6: Compared hardware performance of PRESENT-based and QUARK lightweight hash functions. Security is expressed in bits (e.g., "64" in the "2pre." column means that second preimages can be found within approximately 2^{64} calls to the function). Throughput and power consumption are given for a frequency of 100 kHz.

Hash function	**Security**		**Block**	**Area**	**Lat.**	**Thr.**	**Power** [µW]	
	2pre.	Coll.	[bits]	[GE]	[cycles]	[kbps]	Mean	Peak
DM-PRESENT-80	64	32	80	1'600	547	14.63	1.83	-
DM-PRESENT-80	64	32	80	2'213	33	242.42	6.28	-
DM-PRESENT-128	64	32	128	1'886	559	22.90	2.94	-
DM-PRESENT-128	64	32	128	2'530	33	387.88	7.49	-
H-PRESENT-128	128	64	64	2'330	559	11.45	6.44	-
H-PRESENT-128	128	64	64	4'256	32	200.00	8.09	-
U-QUARK	64	64	8	1'379	544	1.47	2.44	2.96
U-QUARK×8	64	64	8	2'392	68	11.76	4.07	4.84
D-QUARK	80	80	16	1'702	704	2.27	3.10	3.95
D-QUARK×8	80	80	16	2'819	88	18.18	4.76	5.80
T-QUARK	112	112	32	2'296	1024	3.13	4.35	5.53
T-QUARK×16	112	112	32	4'640	64	50.00	8.39	9.79

Table 5.7: Maximum-speed performances of the parallel QUARK cores, when run at a frequency of 714.29 MHz, i.e., 1.4 ns of period.

Hash function	**Area** [GE]	**Lat.** [cycles]	**Freq.** [MHz]	**Thr.** [Mbps]	**Power** [µW] Mean	Peak
u-QUARK×8	3'032	68	714	84	30.46	37.01
d-QUARK×8	3'561	88	714	130	37.14	43.35
t-QUARK×16	6'220	64	714	357	65.34	75.27

Note that in the power results of the QUARK circuits, the single contributions of the mean power consumption are 68 % of internal, 30 % of switching, and 2 % of leakage power. Also important is that the peak value exceeds maximally 27 % of the mean value.

From Table 5.6, DM-PRESENT-80/128 and H-PRESENT-128 also offer implementation trade-offs. For a same (second) preimage resistance of at least 64 bits, u-QUARK fits in a smaller area, and even the 80-bit-secure d-QUARK does not need more gates than DM-PRESENT. In terms of throughput, however, QUARK underperforms PRESENT-based designs. This is clearly visible in the bottom graph of Figure 5.6, where the energy per bit consumption of the QUARK cores appears significantly higher with respect to that of the PRESENT-based cores. This may be due to its high security margin (note that 26 of the 31 rounds of PRESENT, as a block cipher, were attacked [150], suggesting a thin security margin against distinguishers in the "open key" model of hash functions).

Interestingly, for a budget of approximately 2'300 GE, one can choose between H-PRESENT and t-QUARK, which respectively fit in 2'330 and 2'296 GE. The former offers a higher second preimage resistance (128 vs. 112), while the latter offers a higher collision resistance (64 vs. 112). One may thus opt for H-PRESENT for applications requiring only preimage resistance (e.g., challenge-response protocols), and choose t-QUARK when collision resistance is needed (e.g., digital signatures). The two designs can thus be seen as complementary.

The maximum speed achieved by the parallel cores is 357 Mbps with t-QUARK×16 (see Table 5.7). At this frequency, the values of

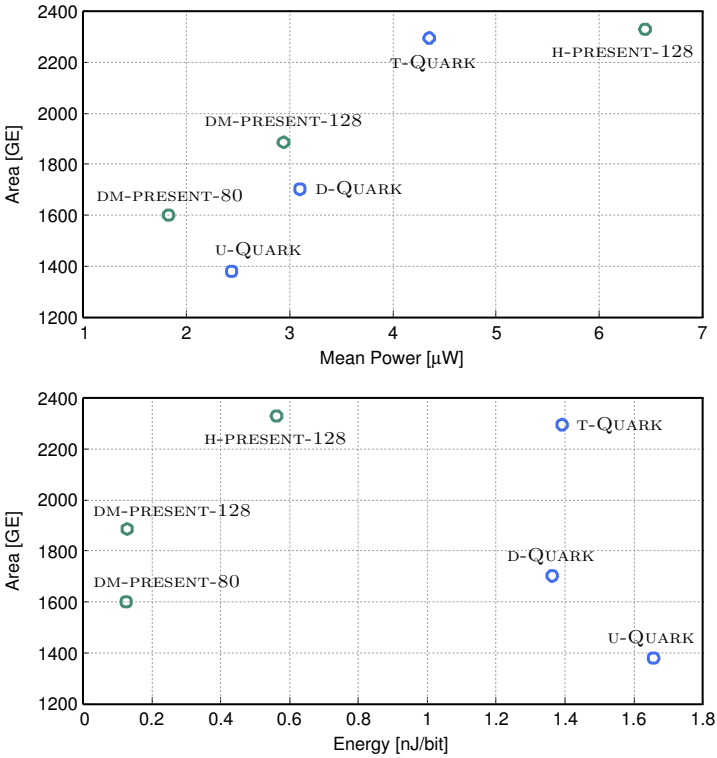

Figure 5.6: Area vs. power (top) and area vs. energy (bottom) comparison of the PRESENT-based and the QUARK hash functions.

the power dissipation increase up to 30-60 mW . The leakage component does not contribute significantly. Indeed, 38 % of the total power is devoted to switching, with the rest for internal power. We do not exclude the possibility to reach higher speed ratios with different architectures. In practice, the latency could be further reduced by implementing more permutation modules. Due to the tap configuration in the function f, g, and h, this would also increase the circuit complexity (i.e., more area and lower frequency), which was outside the scope of this analysis.

6

Summary and Conclusion

6.1 Summary

Keyed hash functions and authenticated encryption algorithms were demonstrated to be fundamental cryptographic schemes to provide data integrity and origin authentication. However, state-of-the-art security protocols for generic authenticity rely on well-established one-way functions, which can not be considered optimal (lack of security, limited versatility, insufficient performance, etc.). Moreover, with the radical diversification of multimedia communication infrastructures (i.e., from tiny portable RFID devices of pervasive computing to massive network backbones and storage servers), the hardware implementation of this security mechanisms becomes the most suitable solution to achieve high-performance results at narrow fabrication costs. To this end, new cryptographic algorithms tailored to specific application constrains must be designed, focusing on the joint optimization of security and architectural aspects.

This thesis is mainly dedicated to the investigation of efficient VLSI architectures for cryptographic primitives that are widely applied in modern communication standards. A large part is devoted to the description of the development process of three new hash functions that have been meticulously evaluated and consequently optimized to achieve maximum hardware performance. With the acquired experience, we felt therefore able to expose a uniform methodology to compare cryptographic algorithms in hardware.

SHA-3 candidate algorithm design and development of a hardware comparison methodology

A large section of this thesis is dedicated to the specification of two new hash functions, which we submitted to the NIST hash competition. Between the two, the BLAKE algorithm is still a contender to become the new hash standard SHA-3. In particular, we presented a complete hardware characterization of the candidates, using different design approaches to generate fully-autonomous high-speed and compact implementations. To this end, a 256-bit BLAKE architecture has been optimized for lightweight applications and fabricated in a 0.18 µm CMOS technology. Measurements on the manufactured ASIC revealed that the compact BLAKE-32 core with circuit area of 13.6 kGE (8.6 kGE without block memory and salt support) has a minimal power dissipation of 130 µW at the RFID nominal frequency of 13.56 MHz, by scaling the supply voltage at 0.65 V.

The organizing committee NIST encouraged the publication of results that investigate and evaluate the performances of the second round SHA-3 candidates. After the fabrication in 90 nm and 0.18 µm CMOS of three ASICs, hosting several circuits of 12 second round algorithms, we decided to extend the analysis including all the candidates. Besides the hardware characterization of the algorithms, the main goal was the description of a reliable methodology that efficiently characterizes and compares VLSI circuits of cryptographic primitives. We took the opportunity to apply it on the ongoing SHA-3 competition. To this end, we implemented several architectures in the 90 nm CMOS technology, targeting high- and moderate-speed constraints, separately. Thanks to this analysis, we were able to

present a complete benchmark of the achieved post-layout results of the circuits.

High-speed authenticated encryption for multi-gigabit link encryptors

A design methodology to implement in reconfigurable hardware devices the GCM combined with the AES for authenticated encryption hash been proposed. Thanks to the replication of four AES cores and four binary-field multipliers we were able to demonstrate how to break the 100 Gbps speed bound in FPGA. In order to reduce the critical path of the GHASH operation, four pipeline stages have been inserted within the GF(2^{128}) multipliers. The final GCM architecture relies on a 4×4 construction and achieves 119 Gbps in Xilinx Virtex-5 devices. The presented FPGA AES-GCM core outperforms state-of-the-art designs for authenticated encryption. It is also the only implementation that matches the bandwidth requirements imposed by the recently-approved IEEE 802.3ba Ethernet standard.

A compact FPGA implementation of a 2G FC link encryptor has further been presented. Authentication and en/decryption processes inside the GCM core were optimized to reach the full throughput of the 2G FC standard with the smallest possible resource utilization. Besides, an evaluation of per-packet authentication for multi-gigabit communication protocols has been performed. The trade-off between security level and maximum throughput was investigated by proposing two different methods of frame encryption. The complete point-to-point secure data communication system has been applied within the SwissQuantum project in the second half of 2009 to encrypt and authenticate a 3 km network node.

FPGA implementation of high-dimensional cube testers on the stream cipher Grain-128

An efficient FPGA implementation of cube testers on the stream cipher Grain-128 has been presented. Our best result (a distinguisher on Grain-128 reduced to 237 rounds, out of 256) was achieved after a computation involving 2^{54} clockings of Grain-128, with a 256×32 parallelization. An extrapolation of our results with standard methods

suggests the possibility of a distinguishing attack on the full Grain-128 in time 2^{83}, which is well below the 2^{128} complexity of exhaustive search. For instance, running a 30-dimensional cube tester on Grain-128 takes 10 seconds with our FPGA machine, against about 45 minutes with our bitsliced C implementation, and more than a day with a straightforward C implementation. Our results suggest that the Grain-128 cipher may not provide 128-bit security.

Development of the lightweight hash function family Quark

The need for lightweight cryptographic hash functions has been repeatedly expressed by application designers, notably for implementing RFID protocols. However not many designs are available, and the ongoing SHA-3 competition probably won't help, as it concerns general-purpose designs and focuses on software performance. We thus proposed a novel design philosophy for lightweight hash functions, based on a single security level and on the sponge construction, to minimize memory requirements. Inspired by the lightweight ciphers Grain and KATAN, we presented the hash function family Quark, composed of the three instances u-Quark, d-Quark, and t-Quark. Hardware benchmarks showed that Quark compares well to previous lightweight hashes. For example, our lightest instance u-Quark conjecturally provides at least 64-bit security against all attacks (preimages, collisions, distinguishers, etc.), fits in 1'379 GE, and consumes in average 2.44 µW at 100 kHz in 0.18 µm ASIC. For 112-bit security, we propose t-Quark, which we implemented with 2'296 GE.

6.2 Conclusion

Following the development of the Intel AES instructions set [151], the next cryptographic standard for message hashing could be integrated in modern central processing units (CPUs) as dedicated hardware module. The next class of microprocessors would then reserve a small portion of the silicon area for embedded security components. The design of new cryptographic hash functions becomes an interdisciplinary task where security and (hardware) efficiency must be jointly

maximized. This thesis covers the design of three novel hash functions that have been optimized to reach the best possible performance in hardware. This is of paramount importance as, e.g., in the AES competition, candidate algorithms have also been judged for their overall implementation performance.

We reported on compact architectures of hash functions that minimize the circuit area. We demonstrated that the optimal solution is the application of low-cost schemes that are able to balance security and implementability. To this end, we proposed the lightweight has family QUARK. At the same time, we developed a semi-custom design approach for memory units, which we believe is a valuable choice to reduce the area and power consumption of integrated circuit in VLSI implementations of cryptographic protocols.

Eventually, the design of efficient authenticated encryption cores, combining data encryption and MACs, has been investigated. The achievement of throughput rates, compliant to the most advanced communication standards, exploits the limits of modern programmable semiconductor devices. In order to reduce the implementation costs, new authenticated encryption modes must be designed. Based on our experience, we strongly advise the exploration of stream ciphers-based authenticated encryption, which could take advantage from the remarkable versatility of stream ciphers in hardware.

A

Hardware Architectures

Tables A.1-A.2 give an overview of the architectures used in the comparison framework of Section 3.5. For some candidates we used the same design for the 20 Gbps (HS) and 0.2 Gbps (MS) analysis. In such cases, different optimization parameters were used. The detailed description of the architectures has been omitted due to space problems. Anyway, the complete implementation process up to the back-end design of the three ASICs could be found in [83, 29]. Furthermore, the source code of all the architectures used in the evaluation of the 14 SHA-3 candidates and the EDA scripts used to obtain the performance results are available online in [78].

Table A.1: Design specification of the HS and MS-target architectures. For the latency, the enclosed value refers to the finalization cycles.

Algorithm	**Message Block Size** [bits]	**Arch.**	**Latency** [cycles]	**Implementation details**
BLAKE	512	HS	21	Four parallel G function modules, anticipation of the first message-constant addition.
		MS	81	One G function module.
BMW	512	HS-MS	18 (+18)	f_0 and f_2 computed in one cycle, while f_1 iteratively decomposed in a single *expand* block.
CubeHash	256	HS	16 (+160)	Single round per cycle, initial state stored.
		MS	32 (+320)	Half round, initial state stored.
ECHO	1536	HS	32	8 AES rounds per clock cycle.
		MS	1034	Single 32-bit AES core, one parallel BigMixColumn unit.
Fugue	32	HS	2 (+37)	S-box as LUT.
		MS	2 (+37)	S-box as composite field logic.
Grøstl	512	HS	21 (+21)	Interleaved P and Q permutation with one pipeline stage, *SubBytes* as LUT.
		MS	160 (+160)	Single-column round (64-bit datapath), *SubBytes* as composite field.
Hamsi	32	HS	3 (+6)	Message expansion in three 256×256 LUTs, single round per cycle, substitution layer as logic.
		MS	24 (+48)	Same as HS, datapath reduced to 128 bits.

Table A.2: Design specification of the HS and MS-target architectures. For the latency, the enclosed value refers to the finalization cycles.

Algorithm	**Message Block Size** [bits]	**Arch.**	**Latency** [cycles]	**Implementation details**
JH	512	HS-MS	36	S-boxes S_0 and S_1 stored in LUTs, constants stored.
Keccak	1088	HS-MS	24	Single round per cycle.
Luffa	256	HS	8	Three parallel *Step* function modules, *SubCrumb* function as logic.
		MS	24	One *Step* function modules, *SubCrumb* function as logic.
Shabal	512	HS	52 (+156)	One keyed permutation round per cycle. In total, 30 adders and 16 subtractors.
		MS	165	One adder and one subtractor only.
SHAvite-3	512	HS	36	One AES round for message expansion and one AES round for the F^3 round, *SubBytes* as LUT.
		MS	36	Same as HS, *SubBytes* in composite field.
SIMD	512	HS-MS	36 (+36)[a]	Four parallel Feistel modules, message expansion based on NNT_8 and eight multipliers for tweadle mult.
Skein	256	HS	19 (+19)	Four unrolled Threefish rounds.
		MS	152 (+152)	Half Threefish round.

[a]Further 36 cycles of initialization required for message expansion.

List of Acronyms

AE	authenticated-encryption
AEAD	authenticated-encryption with associated-data
AES	Advanced Encryption Standard
ANF	algebraic normal form
ASIC	application-specific integrated circuit
BIST	built-in self-test
BPWS	bit-parallel word-serial
BRAM	bloc RAM
CA	certificate authority
CBC	cipher-block chaining
CCM	Counter Mode with CBC-MAC
CFB	cipher feedback
CMOS	complementary metal-oxide semiconductor
CPU	central processing unit
CRC	cyclic redundancy checksum
CTR	counter
DES	Data Encryption Standard
DRC	design rule check
DSP	digital signal processor
ECB	electronic codebook

ECRYPT	European Network of Excellence for Cryptology
EDA	electronic design automation
EOF	end of frame
EPC	electronic product code
ESP	encapsulating security payload
FC	fibre channel
FPGA	field-programmable gate array
FSR	feedback shift register
GCM	Galois/Counter Mode
GE	gate equivalent
GF	Galois field
GMAC	Galois message authentication code
HDTV	high-definition television
HF	high-frequency
HMAC	hash-based message authentication code
ICAO	International Civil Aviation Organization
I/O	input/output
IP	internet protocol
IPsec	internet protocol security
KO	Karatsuba-Ofman
LF	low-frequency
LFSR	linear feedback shift register
LUT	look-up table
MAC	message authentication code
MD	Merkle-Damgård
MGT	multi-gigabit transceiver
MPW	multi project wafer

NFSR	non-linerar feedback shift register
NIST	National Institute of Standards and Technology
OFB	output feedback
OSI	open systems interconnection
PCS	physical conding sublayer
PHY	physical layer
PKI	public key infrastructure
PMA	physical media attachment
PRF	pseudorandom function
PRG	pseudorandom generator
PRP	pseudorandom permutation
QKD	quantum key distribution
RAM	random-access memory
RF	radio-frequency
RFID	radio-frequency identification
RTL	register transfer level
S-box	substitution table
SAN	storage area network
SDH	synchronous digital hierarchy
SERDES	serializer/deserializer
SHA	secure hash standard
SOF	start of frame
SONET	synchronous optical networking
SSH	secure shell
SSL	secure socket layer
TLS	transport layer security
TTM	time to market

UMC	United Microelectronics Corporation
UMTS	universal mobile telecommunications system
USIM	universal subscriber identity module
VCD	value change dump
VLSI	very large scale integration
WAN	wide area network
WEP	wired equivalent privacy
WLAN	wireless local area network
WPA	wi-fi protected access
XOR	exclusive-OR

List of Figures

List of Tables

Bibliography

[1] D. Kahn, *The Codebreakers: The Comprehensive History of Secret Communication from Ancient Times to the Internet.* Scribner, 1967.

[2] S. Vaudenay, *A Classical Introduction to Cryptography.* Springer, 2006.

[3] M. Rieback, B. Crispo, and A. Tanenbaum, "The evolution of RFID security," *IEEE Pervasive Computing*, vol. 5, no. 1, pp. 62–69, 2006.

[4] A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone, *Handbook of Applied Cryptography.* CRC Press, 2001. [Online]. Available: http://www.cacr.math.uwaterloo.ca/hac/

[5] T. Eisenbarth, S. Kumar, C. Paar, A. Poschmann, and L. Uhsadel, "A survey of lightweight-cryptography implementations," *IEEE Design & Test of Computers*, vol. 24, no. 6, pp. 522–533, 2007.

[6] "IEEE Standard for Information technology; Part 3: Carrier Sense Multiple Access with Collision Detection (CSMA/CD) Access Method and Physical Layer Specifications," P802.3ba/D2.0, June 2010.

[7] "The SwissQuantum testbed network," 2009, http://www.swissquantum.com/.

[8] J.-P. Aumasson, I. Dinur, W. Meier, and A. Shamir, "Cube testers and key recovery attacks on reduced-round MD6 and trivium," in *Fast Software Encryption*, ser. LNCS, vol. 5665. Springer Verlag, 2009, pp. 1–22.

[9] M. A. Nielsen and I. L. Chuang, *Quantum Computation and Quantum Information*. Cambridge University Press, 2000.

[10] U. Maurer, "Cryptography," Lecture Notes, ETH Zurich, Autumn Term, 2009.

[11] M. Bellare, R. Canetti, and H. Krawczyk, "Keying hash functions for message authentication," in *Advances in Cryptology – CRYPTO*, ser. LNCS, vol. 1109. Springer Verlag, 1996, pp. 1–15.

[12] R. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," in *Communications of the ACM 21,2*, 1978, pp. 120–126.

[13] D. Hankerson, A. J. Menezes, and S. A. Vanstone, *Guide to elliptic Curve Cryptography*. Springer, USA: Springer Professional Computing, 2003.

[14] D. J. Bernstein, "Understanding brute force," Draft paper, 2005.

[15] C. H. Bennett, G. Brassard, and J.-M. Robert, "Privacy amplification by public discussion," *SIAM J. Comput.*, vol. 17, no. 2, pp. 210–229, 1988.

[16] G. Brassard and L. Salvail, "Secret-key reconciliation by public discussion," in *Advances in Cryptology – EUROCRYPT*, ser. LNCS, vol. 765. Springer Verlag, 1994, pp. 410–423.

[17] C. H. Bennett, G. Brassard, C. Crepeau, and U. Maurer, "Generalized privacy amplification," *IEEE Trans. on Information Theory*, vol. 41, pp. 1915–1923, 1995.

[18] J. Cederlöf and J.-A. Larsson, "Security aspects of the authentication used in quantum cryptography," *IEEE Trans. on Information Theory*, vol. 54, no. 4, pp. 1735–1741, 2008.

[19] M. Fischlin, "Perfectly-crafted Swiss Army knives - in theory," Workshop on Hash Functions in Cryptography: Theory and Practice, 2008.

[20] R. C. Merkle, "A certified digital signature," in *Advances in Cryptology – CRYPTO*, ser. LNCS, vol. 435. Springer Verlag, 1989, pp. 218–238.

[21] I. Damgård, "A design principle for hash functions," in *Advances in Cryptology – CRYPTO*, ser. LNCS, vol. 435. Springer Verlag, 1989, pp. 416–427.

[22] B. Preneel, "The state of hash functions and the NIST SHA-3 competition," in *Information Security and Cryptology*, ser. LNCS. Springer Verlag, 2009, vol. 5487, pp. 1–11.

[23] L. Henzen, F. Carbognani, J. P. Aumasson, S. O'Neil, and W. Fichtner, "VLSI implementations of the cryptographic hash functions MD6 and ïrRUPT," in *Proc. of IEEE ISCAS*, 2009, pp. 2914–2917.

[24] H. Kaeslin, *Digital Integrated Circuit Design, from VLSI Architectures to CMOS Fabrication.* Cambridge University Press, 2008.

[25] L. Henzen, F. Carbognani, N. Felber, and W. Fichtner, "Hardware comparison of the hash function candidates RadioGatun, MAME, and LAKE," in *Proc. of IEEE NORCHIP*, 2008, pp. 65–68.

[26] J. Deepakumara, H. Heys, and R. Venkatesan, "FPGA implementation of MD5 hash algorithm," in *Proc. of IEEE CCECE*, vol. 2, 2001, pp. 919–924.

[27] R. Lien, T. Grembowski, and K. Gaj, "A 1 gbit/s partially unrolled architecture of hash functions SHA-1 and SHA-512," in *Topics in Cryptology – CT-RSA*, ser. LNCS, vol. 2964. Springer Verlag, 2004.

[28] S. Lucks, "A failure-friendly design principle for hash functions," in *Advances in Cryptology – ASIACRYPT*, ser. LNCS, vol. 3788. Springer Verlag, 2005, pp. 474–494.

[29] P. Guillet, E. Pargaetzi, and M. Zoller, "Silicon implementation of second-round SHA-3 candidates," Semester's thesis, ETH Zurich, Autumn Term, 2009.

[30] E. Biham and O. Dunkelman, "A framework for iterative hash functions - HAIFA," Cryptology ePrint Archive, Report 2007/278, 2007.

[31] NIST, "Randomized hashing digital signatures," 2007, special Publication 800-106.

[32] S. Halevi and H. Krawczyk, "Strengthening digital signatures via randomized hashing," in *Advances in Cryptology – CRYPTO*, ser. LNCS, vol. 4117. Springer Verlag, 2006, pp. 41–59.

[33] G. Bertoni, J. Daemen, M. Peeters, and G. V. Assche, "Sponge functions," Ecrypt Hash Workshop, 2007.

[34] G. Bertoni, J. Daemen, M. Peeters, and G. V. Assche, "On the indifferentiability of the sponge construction," in *Advances in Cryptology – EUROCRYPT*, ser. LNCS, vol. 4965. Springer Verlag, 2008, pp. 181–197.

[35] NIST, "Secure hash standard," FIPS Publication 180-2, 2002.

[36] X. Wang and H. Yu, "How to break MD5 and other hash functions," in *Advances in Cryptology – EUROCRYPT*, ser. LNCS, vol. 3494. Springer Verlag, 2005, pp. 19–35.

[37] C. D. Cannière and C. Rechberger, "Finding SHA-1 characteristics: General results and applications," in *Advances in Cryptology – ASIACRYPT*, ser. LNCS, vol. 4284. Springer Verlag, 2006.

[38] V. Klima, "Tunnels in hash functions: MD5 collisions within a minute," Cryptology ePrint Archive, Report 2006/105, 2006.

[39] A. Sotirov, M. Stevens, J. Appelbaum, A. Lenstra, D. Molnar, D. A. Osvik, and B. de Weger, "MD5 considered harmful today. Creating a rogue CA certificate," in *Proc. of the 25th Chaos Communication Congress*, 2008.

[40] NIST, "Announcing request for candidate algorithm nominations for a new cryptographic hash algorithm (SHA-3) family,"

Federal Register, Vol.72, No.212, pp. 62 212–62 220, 2007, http://www.nist.gov/hash-competition.

[41] S. O'Neil, "EnRUPT: First all-in-one symmetric cryptographic primitive," SASC, 2008.

[42] S. O'Neil, K. Nohl, and L. Henzen, "EnRUPT hash function specification," Submission to the NIST Hash Competition, 2008.

[43] S. Indesteege and B. Preneel, "Practical collisions for EnRUPT," in *Fast Software Encryption*, ser. LNCS, vol. 5665. Springer Verlag, 2009, pp. 246–259.

[44] S. Indesteege and B. Preneel, "Practical collisions for EnRUPT," *Journal of Cryptology*.

[45] L. Henzen, F. Carbognani, N. Felber, and W. Fichtner, "Hardware evaluation of the stream cipher-based hash functions RadioGatun and irRUPT," in *Proc. of IEEE DATE*, 2009, pp. 646–651.

[46] J.-P. Aumasson, W. Meier, and R. C. W. Phan, "The hash function family LAKE," in *Fast Software Encryption*, ser. LNCS, vol. 5086. Springer Verlag, 2008, pp. 36–53.

[47] D. J. Bernstein, "ChaCha, a variant of Salsa20," SASC, 2008.

[48] D. J. Bernstein, "Salsa20," eSTREAM, ECRYPT Stream Cipher Project, Report 2005/025, 2005.

[49] D. J. Bernstein, "The salsa20 family of stream ciphers," in *New Stream Cipher Designs*, ser. LNCS, vol. 4986. Springer Verlag, 2008, pp. 84–97.

[50] L. Henzen, J.-P. Aumasson, W. Meier, and R. C.-W. Phan, "VLSI characterization of the cryptographic hash function BLAKE," *IEEE Trans. on VLSI Systems*, vol. PP, no. 99, pp. 1–9, 2010.

[51] J.-P. Aumasson, L. Henzen, W. Meier, and R. C.-W. Phan, "SHA-3 proposal BLAKE," Submission to the NIST Hash Competition, 2008.

[52] L. Henzen, F. Carbognani, N. Felber, and W. Fichtner, "VLSI hardware evaluation of the stream ciphers Salsa20 and ChaCha, and the compression function Rumba," in *Proc. of IEEE SCS*, 2008, pp. 1–5.

[53] S. Tillich, M. Feldhofer, M. Kirschbaum, T. Plos, J.-M. Schmidt, and A. Szekely, "High-speed hardware implementations of BLAKE, Blue Midnight Wish, CubeHash, ECHO, Fugue, Grøstl, Hamsi, JH, Keccak, Luffa, Shabal, SHAvite-3, SIMD, and Skein," Cryptology ePrint Archive, Report 2009/510, 2009.

[54] P. Gauravaram, L. R. Knudsen, K. Matusiewicz, F. Mendel, C. Rechberger, M. Schläffer, and S. S. Thomsen, "Grøstl - a SHA-3 candidate," Submission to NIST, 2008.

[55] A. Satoh, "ASIC hardware implementations for 512-bit hash function Whirlpool," in *Proc. of IEEE ISCAS*, 2008, pp. 2917–2920.

[56] M. Bernet, L. Henzen, H. Kaeslin, N. Felber, and W. Fichtner, "Hardware implementations of the SHA-3 candidates Shabal and CubeHash," in *Proc. of IEEE MWSCAS*, 2009, pp. 515–518.

[57] L. Lu, M. O'Neill, and E. Swartzlander, "Hardware evaluation of SHA-3 hash function candidate ECHO," in *Proc. of the Claude Shannon Workshop on Coding and Cryptography*, 2009.

[58] O. Küçük, "The hash function Hamsi," Submission to NIST, 2008.

[59] G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche, "Keccak sponge function family," Submission to NIST, 2008.

[60] C. D. Cannière, H. Sato, and D. Watanabe, "Hash function Luffa," Submission to NIST, 2008.

[61] Y. K. Lee, H. Chan, and I. Verbauwhede, "Iteration bound analysis and throughput optimum architecture of SHA-256 (384,512) for hardware implementations," in *Information Security Applications*, ser. LNCS, vol. 4867. Springer Verlag, 2008, pp. 102–114.

[62] S. Halevi, W. E. Hall, and C. S. Jutla, "The hash function Fugue," Submission to NIST, 2008.

[63] I. Dinur and A. Shamir, "Cube attacks on tweakable black box polynomials," in *Advances in Cryptology – EUROCRYPT*, ser. LNCS, vol. 5479. Springer Verlag, 2009, pp. 278–299.

[64] C. Boura and A. Canteaut, "A zero-sum property for the Keccak-f permutation with 18 rounds," NIST mailing list, 2010. [Online]. Available: http://www-roc.inria.fr/secret/Anne.Canteaut/Publications/zero_sum.pdf

[65] F. Gut and M. Hotz, "VLSI implementation of the Whirlpool hash function," Semester's thesis, ETH Zurich, Autumn Term, 2008.

[66] A. Juels, "RFID security and privacy: a research survey," *IEEE Trans. on Selected Areas in Communications*, vol. 24, no. 2, pp. 381–394, 2006.

[67] Y. Eslami, A. Sheikholeslami, P. G. Gulak, S. Masui, and K. Mukaida, "An area-efficient universal cryptography processor for smart cards," *IEEE Trans. on VLSI Systems*, vol. 14, no. 1, pp. 43–56, 2006.

[68] M. O'Neill, "Low-cost SHA-1 hash function architecture for RFID tags," in *Proc. of RFIDsec*, 2008.

[69] M. Feldhofer and J. Wolkerstorfer, "Strong crypto for RFID tags - a comparison of low-power hardware implementations," in *Proc. of IEEE ISCAS*, 2007, pp. 1839–1842.

[70] M. Feldhofer, J. Wolkerstorfer, and V. Rijmen, "AES implementation on a grain of sand," in *Proc. of IEE Information Security*, vol. 152, 2005, pp. 13–20.

[71] D. Hein, J. Wolkerstorfer, and N. Felber, "ECC is ready for RFID - a proof in silicon," in *Selected Areas in Cryptography*, ser. LNCS, vol. 5381. Springer Verlag, 2009, pp. 401–413.

[72] L. Batina, J. Guajardo, B. Preneel, P. Tuyls, and I. Verbauwhede, "Public-key cryptography for RFID tags and applications," in *RFID Security*. Springer US, 2009, pp. 317–348.

[73] S. E. Sarma, S. A. Weis, and D. W. Engels, "RFID systems and security and privacy implications," in *Cryptographic Hardware and Embedded Systems - CHES*, ser. LNCS, vol. 2523. Springer Verlag, 2002, pp. 454–469.

[74] J. Wolkerstorfer, "Is elliptic-curve cryptography suitable to secure RFID tags?" in *Proc. of the Workshop on RFID and Lightweight Crypto*, 2005.

[75] M. Feldhofer, S. Dominikus, and J. Wolkerstorfer, "Strong authentication for RFID systems using the AES algorithm," in *Cryptographic Hardware and Embedded Systems – CHES*, ser. LNCS, vol. 3156. Springer Verlag, 2004, pp. 85–140.

[76] S. Tillich, M. Feldhofer, W. Issovits, T. Kern, H. Kureck, M. Mühlberghuber, G. Neubauer, A. Reiter, A. Köfler, and M. Mayrhofer, "Compact hardware implementations of the SHA-3 candidates ARIRANG, BLAKE, Grøstl, and Skein," Cryptology ePrint Archive: Report 2009/349, 2009.

[77] L. Henzen, P. Gendotti, P. Guillet, E. Pargaetzi, M. Zoller, and F. K. Gürkaynak, "Developing a hardware evaluation method for SHA-3 candidates," in *Cryptographic Hardware and Embedded Systems – CHES*, ser. LNCS, vol. 6225. Springer Verlag, 2010, pp. 248–263.

[78] F. K. Gürkaynak, L. Henzen, P. Gendotti, P. Guillet, E. Pargaetzi, and M. Zoller, "Hardware evaluation of the second-round SHA-3 candidate algorithms," 2010, http://www.iis.ee.ethz.ch/~sha3/.

[79] A. H. Namin and M. A. Hasan, "Hardware implementation of the compression function for selected SHA-3 candidates," CACR 2009-28, 2009.

[80] K. Kobayashi, J. Ikegami, S. Matsuo, K. Sakiyama, and K. Ohta, "Evaluation of hardware performance for the SHA-3

candidates using SASEBO-GII," Cryptology ePrint Archive, Report 2010/010, 2010.

[81] F. K. Gürkaynak, P. Luethi, N. Bernold, R. Blattmann, V. Goode, M. Marghitola, H. Kaeslin, N. Felber, and W. Fichtner, "Hardware evaluation of eSTREAM candidates: Achterbahn, grain, mickey, mosquito, sfinks, trivium, vest, zk-crypt," eSTREAM, ECRYPT Stream Cipher Project, Report 2006/015, 2006.

[82] M. Feldhofer, "Comparison of low-power implementations of Trivium and Grain," eSTREAM, ECRYPT Stream Cipher Project, Report 2007/027, 2007.

[83] P. Gendotti, "Silicon implementation of non-AES-based SHA-3 second round candidates," Master's thesis, ETH Zurich, Spring Term, 2009.

[84] M. Bellare and C. Namprempre, "Authenticated encryption: Relations among notions and analysis of the generic composition paradigm," in *Advances in Cryptology – ASIACRYPT*, ser. LNCS, vol. 1976. Springer Verlag, 2000, pp. 531–545.

[85] T. Kohno, J. Viega, and D. Whiting, "The CWC authenticated encryption (associated data) mode," 2003, Submission to NIST Modes of Operation.

[86] P. Rogaway, "Authenticated-encryption with associated-data," in *Proc. of ACM CCS-9*, 2002, pp. 98–107.

[87] F. Xu, B. Qi, and H.-K. Lo, "Experimental demonstration of phase-remapping attack in a practical quantum key distribution system," eprint arXiv:1005.2376, 2010.

[88] S. Sauge, V. Makarov, and A. Anisimov, "Quantum hacking: how Eve can exploit component imperfections to control yet another of Bob's single-photon qubit detectors," in *CLEO/Europe-EQEC*, 2009.

[89] A. Biryukov and D. Khovratovich, "Related-key cryptanalysis of the full AES-192 and AES-256," in *Advances in Cryptology*

– *ASIACRYPT*, ser. LNCS, vol. 5912. Springer Verlag, 2009, pp. 1–18.

[90] A. Biryukov, O. Dunkelman, N. Keller, D. Khovratovich, and A. Shamir, "Key recovery attacks of practical complexity on AES variants with up to 10 rounds," Cryptology ePrint Archive, Report 2009/374, 2009.

[91] H. Gilbert and T. Peyrin, "Super-sbox cryptanalysis: Improved attacks for AES-like permutations," in *Fast Software Encryption*, ser. LNCS, vol. 6147. Springer Verlag, 2010, pp. 365–383.

[92] A. Biryukov, D. Khovratovich, and I. Nikolic, "Distinguisher and related-key attack on the full AES-256," in *Advances in Cryptology – CRYPTO*, ser. LNCS, vol. 5677. Springer Verlag, 2009, pp. 231–249.

[93] NIST, "Advanced encryption standard (AES)," FIPS Publication 197, 2001.

[94] T. Good and M. Benaissa, "AES on FPGA from the fastest to the smallest," in *Cryptographic Hardware and Embedded Systems – CHES*, ser. LNCS, vol. 3659. Springer Verlag, 2005, pp. 427–440.

[95] K. Gürkaynak, "GALS system design: Side channel attack secure cryptographic accelerators," Ph.D. dissertation, ETH Zürich, Switzerland, 2006.

[96] V. Rijmen, "Efficient implementation of the Rijndael S-box," 2000.

[97] J. Wolkerstorfer, E. Oswald, and M. Lamberger, "An ASIC implementation of the AES SBoxes," in *Topics in Cryptology – CT-RSA*, ser. LNCS, vol. 2271. Springer Verlag, 2002, pp. 29–52.

[98] F.-X. Standaert, G. Rouvroy, J.-J. Quisquater, and J.-D. Legat, "Efficient implementation of Rijndael in reconfigurable hardware: Improvements and design tradeoffs," in *Cryptographic Hardware and Embedded Systems – CHES*, ser. LNCS, vol. 2779. Springer Verlag, 2003, pp. 334–350.

[99] F. Coduri, C. Pagnamenta, and R. Poretti, "Fast AES cipher core on ASIC with new cells based on CMOS and pass-transistor logic," Master's thesis, ETH Zurich, Winter Term, 2007.

[100] A. Hodjat and I. Verbauwhede, "Minimum area cost for a 30 to 70 Gbits/s AES processor," in *Proc. of IEEE ISVLSI*, 2004, pp. 83–88.

[101] D. A. McGrew and J. Viega, "The Galois/Counter Mode of operation (GCM)," 2005, Submission to NIST Modes of Operation.

[102] J. Carter and M. N. Wegman, "Universal classes of hash functions." *Journal of Computer and System Sciences*, vol. 18, pp. 143–154, 1979.

[103] M. N. Wegman and J. Carter, "New hash functions and their use in authentication and set equality." *Journal of Computer and System Sciences*, vol. 22, pp. 265–279, 1981.

[104] L. Henzen and W. Fichtner, "FPGA parallel-pipelined AES-GCM core for 100G Ethernet applications," in *Proc. of IEEE ESSCIRC*, 2010, to appear.

[105] A. Bouhraoua, "Design feasibility study for a 500 Gbits/s Advanced Encryption Standard cipher/decipher engine," *IET Computers & Digital Techniques*, vol. 4, no. 4, pp. 334–348, 2010.

[106] G. Zhou, H. Michalik, and L. Hinsenkamp, "Improving throughput of AES-GCM with pipelined karatsuba multipliers on FPGAs," in *Reconfigurable Computing: Architectures, Tools and Applications*, ser. LNCS, vol. 5453. Springer Verlag, 2009, pp. 193–203.

[107] M. Dworkin, "Recommendation for block cipher mode of operation: Galois/Counter Mode (GCM) and (GMAC)," 2007, NIST Special Publication 800-38D.

[108] G. Zhou, H. Michalik, and L. Hinsenkamp, "Complexity analysis and efficient implementations of bit parallel finite field multipliers based on Karatsuba-Ofman algorithm on FPGAs," *IEEE Trans. on VLSI Systems*, vol. 18, no. 7, pp. 1057–1066, 2010.

[109] L. Henzen, F. Carbognani, N. Felber, and W. Fichtner, "FPGA implementation of a 2G Fibre Channel link encryptor with authenticated encryption mode GCM," in *Proc. of IEEE ISSOC*, 2008, pp. 1–4.

[110] W. Tang, H. Wu, and M. Ahmadi, "VLSI implementation of bit-parallel word-serial multiplier in GF(2^{233})," in *Proc. of IEEE NEWCAS*, 2005, pp. 399–402.

[111] R. Snively, C. DeSanti, C. Carlson, W. Martin, and R. Nixon, "Fibre Channel, framing and signaling-2 (FC-FS-2)," 2006, INCITS woking draft proposed American National Standard for Information Technology.

[112] A. Juels, "The vision of secure RFID," *Proceedings of the IEEE*, vol. 95, no. 8, pp. 1507–1508, 2007.

[113] S. Garfinkel, A. Juels, and R. Pappu, "RFID privacy: an overview of problems and proposed solutions," *IEEE Security & Privacy*, vol. 3, no. 3, pp. 34–43, 2005.

[114] A. Juels and S. A. Weis, "Authenticating pervasive devices with human protocols," in *Advances in Cryptology – CRYPTO*, ser. LNCS, vol. 3621. Springer Verlag, 2005, pp. 293–308.

[115] S. E. Sarma, S. A. Weis, and D. W. Engels, "Radio-frequency identification: Security risks and challenges," in *RSA Laboratories CryptoBytes*, vol. 6, 2003.

[116] G. Leander, C. Paar, A. Poschmann, and K. Schramm, "New lightweight DES variants," in *Fast Software Encryption*, ser. LNCS, vol. 4593. Springer Verlag, 2007, pp. 196–210.

[117] A. Bogdanov, L. R. Knudsen, G. Leander, C. Paar, A. Poschmann, M. J. B. Robshaw, Y. Seurin, and C. Vikkelsoe,

"PRESENT: An ultra-lightweight block cipher," in *Cryptographic Hardware and Embedded Systems - CHES*, ser. LNCS, vol. 4727. Springer Verlag, 2007, pp. 450–466.

[118] C. D. Cannière, O. Dunkelman, and M. Knezevic, "KATAN and KTANTAN – a family of small and efficient hardware-oriented block ciphers," in *Cryptographic Hardware and Embedded Systems - CHES*, ser. LNCS, vol. 5747. Springer Verlag, 2009, pp. 272–288.

[119] C. Rolfes, A. Poschmann, G. Leander, and C. Paar, "Ultra-lightweight implementations for smart devices – security for 1000 gate equivalents," in *Smart Card Research and Advanced Applications*, ser. LNCS, vol. 5189. Springer Verlag, 2008, pp. 89–103.

[120] M. Hell, T. Johansson, and W. Meier, "Grain – a stream cipher for constrained environments," eSTREAM, ECRYPT Stream Cipher Project, Report 2005/010, 2005.

[121] M. Hell, T. Johansson, and W. Meier, "Grain: A stream cipher for constrained environments," *International Journal of Wireless and Mobile Computing*, vol. 2, no. 1, pp. 86–93, 2007.

[122] M. Hell, T. Johansson, A. Maximov, and W. Meier, "The grain family of stream ciphers," in *New Stream Cipher Designs*, ser. LNCS, vol. 4986. Springer Verlag, 2008, pp. 179–190.

[123] C. D. Cannière and B. Preneel, "Trivium – a stream cipher construction inspired by block cipher design principles," eSTREAM, ECRYPT Stream Cipher Project, Report 2005/030, 2005.

[124] C. D. Cannière and B. Preneel, "Trivium," in *New Stream Cipher Designs*, ser. LNCS, vol. 4986. Springer Verlag, 2008, pp. 244–266.

[125] T. Good and M. Benaissa, "Hardware results for selected stream cipher candidates," eSTREAM, ECRYPT Stream Cipher Project, Report 2007/023, 2007.

[126] N. Mentens, J. Genoe, B. Preneel, and I. Verbauwhede, "A low-cost implementation of Trivium," SASC, 2008.

[127] M. Feldhofer and C. Rechberger, "A case against currently used hash functions in RFID protocols," in *OTM Workshops*, ser. LNCS, vol. 4277. Springer Verlag, 2006, pp. 372–381.

[128] M. Kim, J. Ryou, and S. Jun, "Efficient hardware architecture of SHA-256 algorithm for trusted mobile computing," in *Information Security and Cryptology*, ser. LNCS, vol. 5487. Springer Verlag, 2009, pp. 240–252.

[129] B. Preneel, "Status and challenges of lightweight crypto," Talk at the Early Symmetric Crypto (ESC) seminar, January 2010.

[130] A. Shamir, "SQUASH – a new MAC with provable security properties for highly constrained devices such as RFID tags," in *Fast Software Encryption*, ser. LNCS, vol. 5086. Springer Verlag, 2008, pp. 144–157.

[131] A. Bogdanov, G. Leander, C. Paar, A. Poschmann, M. J. B. Robshaw, and Y. Seurin, "Hash functions and RFID tags: Mind the gap," in *Cryptographic Hardware and Embedded Systems – CHES*, ser. LNCS, vol. 5154. Springer Verlag, 2008, pp. 283–299.

[132] S. O'Neil, "Algebraic structure defectoscopy," Cryptology ePrint Archive, Report 2007/378, 2007.

[133] H. Englund, T. Johansson, and M. S. Turan, "A framework for chosen IV statistical analysis of stream ciphers," in *Progress in Cryptology – INDOCRYPT*, ser. LNCS, vol. 4859. Springer Verlag, 2007, pp. 268–281.

[134] M.-J. O. Saarinen, "Chosen-IV statistical attacks on eSTREAM stream ciphers," in *Proc. of SECRYPT*, 2006, pp. 260–266.

[135] A. Shamir, "How to solve it: New techniques in algebraic cryptanalysis," Invite Talk at CRYPTO, 2008.

[136] J.-P. Aumasson, I. Dinur, L. Henzen, W. Meier, and A. Shamir, "Efficient FPGA implementations of high-dimensional cube testers on the stream cipher Grain-128," Cryptology ePrint Archive, Report 2009/218, 2009, presented at SHARCS'09.

[137] M. Blum, M. Luby, and R. Rubinfeld, "Self-testing/correcting with applications to numerical problems," in *Proc. of the ACM STOC*, 1990, pp. 73–83.

[138] M. Hell, T. Johansson, A. Maximov, and W. Meier, "A stream cipher proposal: Grain-128," in *Proc. of IEEE ISIT*, 2006, pp. 1614–1618.

[139] S. Fischer, S. Khazaei, and W. Meier, "Chosen IV statistical analysis for key recovery attacks on stream ciphers," in *Progress in Cryptology – AFRICACRYPT*, ser. LNCS, vol. 5023. Springer Verlag, 2008, pp. 236–245.

[140] C. D. Cannière, Ö. Küçük, and B. Preneel, "Analysis of grain's initialization algorithm," in *Progress in Cryptology – AFRICACRYPT*, ser. LNCS, vol. 5023. Springer Verlag, 2008, pp. 276–289.

[141] E. Biham, "A fast new DES implementation in software," in *Fast Software Encryption*, ser. LNCS, vol. 1267. Springer Verlag, 1997, pp. 260–272.

[142] P. Bulens, K. Kalach, F.-X. Standaert, and J. J. Quisquater, "FPGA implementations of eSTREAM Phase-2 focus candidates with hardware profile," eSTREAM, ECRYPT Stream Cipher Project, Report 2007/024, 2007.

[143] K. Gaj, G. Southern, and R. Bachimanchi, "Comparison of hardware performance of selected Phase ii eSTREAM candidates," eSTREAM, ECRYPT Stream Cipher Project, Report 2007/026, 2007.

[144] S. Kumar, C. Paar, J. Pelzl, G. Pfeiffer, and M. Schimmler, "Breaking ciphers with COPACOBANA – a cost-optimized parallel code breaker," in *Cryptographic Hardware and Embedded*

Systems - CHES, ser. LNCS, vol. 4249. Springer Verlag, 2006, pp. 101–118.

[145] T. Guneysu, T. Kasper, M. Novotny, C. Paar, and A. Rupp, "Cryptanalysis with COPACOBANA," *IEEE Computer*, vol. 57, no. 11, pp. 1498–1513, 2008.

[146] T. Good, W. Chelton, and M. Benaissa, "Review of stream cipher candidates from a low resource hardware perspective," eSTREAM, ECRYPT Stream Cipher Project, Report 2006/016, 2007.

[147] J.-P. Aumasson, L. Henzen, W. Meier, and M. Naya-Plasencia, "QUARK a lightweight hash," in *Cryptographic Hardware and Embedded Systems – CHES*, ser. LNCS, vol. 6225. Springer Verlag, 2010, pp. 1–15.

[148] J.-P. Aumasson, L. Henzen, W. Meier, and M. Naya-Plasencia, "QUARK: a lightweight hash," Upon solicitation from the CHES 2010 committee, submitted to the *Journal of Cryptology*, 2010, extended version of [147].

[149] G. Bertoni, J. Daemen, M. Peeters, and G. V. Assche, "RadioGatún, a belt-and-mill hash function," Second NIST Cryptographic Hash Function Workshop, 2006.

[150] J. Y. Cho, "Linear cryptanalysis of reduced-round PRESENT," in *Topics in Cryptology – CT-RSA*, ser. LNCS, vol. 5985. Springer Verlag, 2010, pp. 302–317.

[151] "Intel Advanced Encryption Standard (AES) Instructions set," White paper of Intel Corporation, 2010.