KATHOLIEKE UNIVERSITEIT
LEUVEN

**Arenberg Doctoral School of Science, Engineering & Technology**
Faculty of Engineering
Department of Electrical Engineering (ESAT)

# Efficient Hardware Implementations of Cryptographic Primitives

**Miroslav KNEŽEVIĆ**

Dissertation presented in partial fulfillment of the requirements for the degree of Doctor in Engineering

March 2011

# Efficient Hardware Implementations of Cryptographic Primitives

**Miroslav KNEŽEVIĆ**

Jury:
Prof. dr. Hugo Hens, chairman
Prof. dr. Ingrid Verbauwhede, promotor
Prof. dr. Lejla Batina
(University of Nijmegen, The Netherlands)
Prof. dr. Georges Gielen
Prof. dr. Bart Preneel
Prof. dr. Kazuo Sakiyama
(The University of Electro-Communications, Japan)

Dissertation presented in partial fulfillment of the requirements for the degree of Doctor in Engineering

March 2011

Legal depot number D/2011/7515/32
ISBN number 978-94-6018-330-0

*To the memory of my mother*

*Silent gratitude isn't*
*much use to anyone.*

*G. B. Stern*

# Acknowledgments

Writing this personal letter of gratitude sets me up for travelling back in time, recalling all the valuable moments I had during the past several years. Simply too many people have been involved and generously contributed to the final shape of this dissertation. Several individuals deserve my greatest gratitude for their contributions and support.

First of all, I am very grateful to my PhD advisor, Prof. Ingrid Verbauwhede. Not only did she guide me through the jungle of research all along, she also gave me a unique opportunity to join COSIC at the very first place. I appreciate the research freedom she offered, always giving me a chance to choose.

I am honored to have Prof. Lejla Batina, Prof. Georges Gielen, Prof. Bart Preneel, and Prof. Kazuo Sakiyama as the members of my jury, and Prof. Hugo Hens for chairing the jury. Thank you for carefully evaluating my dissertation and providing the valuable feedback.

I owe my deepest gratitude to Kazuo Sakiyama and Frederik Vercauteren for spending their precious time with the rookie, who I undoubtedly was at the beginning of this journey. Please forgive my ignorance and silly questions I was able to ask.

These acknowledgments would certainly remain incomplete without mentioning the name of Vesselin Velichkov, my first office mate and comrade during the years of our doctorates. I learned a lot from Junfeng Fan and Yong Ki Lee, and had a unique chance to work together with dozens of eminent researchers. I would therefore like to acknowledge some of my co-authors and people who inspired my research in one way or another: Lejla Batina, Andrey Bogdanov, Christophe De Cannière, Elke De Mulder, Orr Dunkelman, Benedikt Gierlichs, Duško Karaklajić, Roel Maes, Christian Rechberger, Vladimir Rožić, and Dai Watanabe. I am also very grateful to the rest of the COSIC crew who made my working environment and life in Leuven very pleasant. A special thanks goes to Péla Noé and Elsy Vermoesen for unravelling the labyrinths of bureaucracy and always effectively solving a problem with a smile.

I am very grateful to all of my friends who always made sure to bring me back down to earth, never leaving me stranded. The guys from Belgrade, Leuven, Moscow, Lausanne, Brussels. Thanks for giving me a chance to win, to lose, to be happy, to be stubborn, to be selfish, to be generous. I would also like to thank Anya for being kind, so unselfish, and always there for me.

Finally, I would like to thank my parents, my sister, and my big family for their unconditional support and for taking care of me more than I sometimes deserved.

Miroslav Knežević
Leuven, March 2011

# Abstract

Society is undergoing a paradigm shift where the Information and Communication Technology (ICT) revolution goes along with the evolution of the humankind. The Internet is all around us and plays a crucial role in our ability to communicate. We often distribute our personal and other classified information using the benefits of the global network. Our demands to conceal confidential data are therefore being strongly manifested and become very important. By ensuring the objectives of information security, such as confidentiality, data integrity, entity authentication, non-repudiation, and many more, cryptography provides a natural solution to the issue of data protection.

The ICT revolution has driven cryptography from the art of secret writing into a multidisciplinary scientific study of techniques for securing digital information. While providing aspects of information security, cryptography uses complex mathematical objects and often represents a bottleneck in hardware and software implementations. The research presented in this thesis deals with efficient hardware implementations of cryptographic primitives.

The first part of the thesis is devoted to efficient implementations of finite field arithmetic, with the application in public-key cryptography. Our focus on state of the art algorithms for efficient modular multiplication eventually leads to the introduction of several sets of moduli for which the modular multiplication performs faster. Furthermore, by combining several existing algorithms, we propose the tripartite modular multiplication, a novel method that reduces the computational complexity of modular multiplication and increases the potential of parallel processing.

The second part of the thesis presents techniques for high-throughput hardware implementations of cryptographic hash functions. Our hardware implementation of the RIPEMD-160 hash algorithm represents the fastest implementation of this algorithm reported in the literature. As a contribution to the SHA-3 competition launched by the National Institute of Standards and Technology (NIST), we define a standard testing framework for a comprehensive hardware evaluation of fourteen second-round SHA-3 candidates.

Finally, we discuss recent advances in lightweight cryptography. Our contribution to this field is KATAN & KTANTAN – a family of small, very efficient, hardware-oriented block ciphers. The family comprises six designs, the smallest of which has size of only 462 NAND gate equivalences (GE). KATAN & KTANTAN is the smallest family of cryptographic primitives suitable for the current CMOS technology reported in the literature.

# Samenvatting

De samenleving ondergaat een paradigmaverschuiving waar de Informatie en Communicatie Technologie (ICT) revolutie hand in hand gaat met de menselijke evolutie. Het internet is alomtegenwoordig en speelt een cruciale rol in ons communicatievermogen. We delen onze persoonlijke en andere geheime informatie vaak over het internet. De nood om vertrouwelijke informatie te verbergen, manifesteert zich daardoor in grotere mate en is zeer belangrijk geworden. Door een aantal aspecten van informatiebeveiliging, zoals vertrouwelijkheid, integriteit van gegevens, authenticatie van entiteiten, onweerlegbaarheid en nog veel meer te garanderen, biedt cryptografie een natuurlijke oplossing aan voor het vraagstuk naar veiligheid.

De ICT revolutie heeft cryptografie omgevormd van een kunst in het geheimschrift naar een multidisciplinair wetenschappelijk onderzoeksgebied voor technieken om digitale informatie te beveiligen. Om bepaalde aspecten van informatiebeveiliging te verwezenlijken, maakt cryptografie gebruik van complexe wiskundige objecten en is daardoor vaak een knelpunt in hardware- en software-implementaties. Het onderzoek gepresenteerd in dit proefschrift behandelt efficiënte hardware-implementaties van cryptografische primitieven.

Het eerste deel van het proefschrift is gewijd aan efficiënte implementaties van eindige-veld berekeningen. Onze focus op state-of-the-art algoritmes voor efficiënte modulaire vermenigvuldigingen leidt uiteindelijk tot de invoering van verschillende sets van moduli waarvoor de modulaire vermenigvuldiging sneller presteert. Bovendien, door het combineren van verschillende bestaande algoritmes, stellen we de tripartiete modulaire vermenigvuldiging voor, een nieuwe methode die de computationele complexiteit van een modulaire vermenigvuldiging vermindert en de mogelijkheden voor parallelle verwerking verhoogt.

Het tweede deel van het proefschrift presenteert technieken voor hardware-implementaties van cryptografische hashfuncties met hoge doorvoer. Onze hardware-uitvoering van het RIPEMD-160 hashalgoritme vertegenwoordigt de snelste implementatie in de literatuur. Als bijdrage aan de SHA-3 competitie georganiseerd door het National Institute of Standards and Technology (NIST), definiëren we een

gestandardiseerd testkader voor een uitgebreide hardware-evaluatie van de veertien SHA-3 kandidaten uit de tweede ronde.

Tot slot bespreken we de recente ontwikkelingen in de lichtgewichtcryptografie. Onze bijdrage aan dit veld is KATAN & KTANTAN – een familie van kleine, zeer efficiënte, hardware-georiënteerde blokcijfers. De familie bestaat uit zes ontwerpen, waarvan de kleinste een grootte van slechts 462 NAND-poort equivalenties (GE) heeft. KATAN & KTANTAN is de kleinste familie van cryptografische primitieven die geschikt zijn voor de huidige CMOS-technologie gepubliceerd in de literatuur.

# Резиме

Захваљујући револуцији информационих и комуникационих технологија (ICT), савремено друштво доживљава значајне промене у начину комуникације и размене података. Интернет је свуда око нас и друштво врло лако прихвата предности које ICT револуција нуди. Подаци које размењујемо користећи глобалну мрежу, често су личне природе или спадају у домен врло поверљивих информација. Стога се јавља проблем заштите таквих података и потреба за његовим решењем постаје све већа. Бавећи се многим аспектима заштите информација, криптографија се у овом случају намеће као природно решење.

С почетка сматрана уметношћу тајног писања криптографија се, пратећи трендове ICT револуције, развила у мултидисциплинарну науку заштите тајности дигиталних података. У ту сврху криптографија користи комплексне математичке објекте због чега често представља "уско грло" хардверских и софтверских имплементација. Истраживање којим се бави ова теза разматра управо ефикасне хардверске имплементације криптографских алгоритама.

Први део тезе посвећен је ефикасним имплементацијама аритметике коначних поља са применом у криптографским системима јавних кључева. Фокус је стављен на тренутно најнапредније алгоритме за модуларно множење. Након тога уводимо неколико скупова модула за које модуларно множење постаје знатно ефикасније. Такође, комбиновањем неколико већ постојећих алгоритама уводимо нови, такозвани "*tripartite modular multiplication*" алгоритам. Предложени алгоритам смањује комплексност модуларног множења и посебно је погодан за системе у којима је могуће извршавати више паралелних операција истовремено.

Други део тезе уводи технике за ефикасно имплементирање хеш функција. Имплементација RIPEMD-160 хеш алгоритма која је описана у овом поглављу представља тренутно најбржу имплементацију овог алгоритма објављену у академским круговима. Као допринос такмичењу за избор новог SHA-3 стандарда, организованог од стране америчког Националног Института за Стандарде и Технологију (NIST), ми уводимо нови метод за тестирање и комплетну хардвер оцену перформанси свих четрнаест SHA-3 кандидата.

Коначно, у последњем поглављу ове тезе разматрана су најновија открића у пољу "*lightweight*" криптографије. Као допринос том пољу ми предлажемо KATAN & KTANTAN – фамилију малих, врло ефикасних блок шифара која је посебно погодна за компактне хардвер имплементације. Фамилија садржи шест верзија, од којих је најмања еквивалент величине свега 462 NAND интегрисаних кола. KATAN & KTANTAN тренутно представљају најкомпактнију фамилију блок шифара погодну за имплементације у постојећој CMOS технологији.

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

In our ever growing world of technology and communication, the amount of information we share with the rest of the digital universe is constantly increasing. Advances in the field of digital signal processing, bringing together the power of audio and visual experiences, are establishing the way digital information is shared. With the appearance of cloud computing, a technology that offers online data and application maintenance, we distribute our computational tasks and data storage over a broad network of computers. The speed the data is transmitted with, as well as the execution time of the remote applications play a vital role in having the cloud computing concept widely acceptable.

The rapid employment of Radio Frequency Identification (RFID) tags, on the other side, goes along with the expansion of ubiquitous computing, a model in which information processing becomes thoroughly integrated into everyday's objects and activities. With the appearance of smart devices we are witnessing the presence of a digital continuum, a concept that irreversibly closes the gap between high-speed and low-cost electronics. We are already surrounded by billions of embedded devices and that number is rapidly increasing. Besides being transparent to the end-users, the protocols carrying out the heavy computational tasks need to be power and energy efficient in this case. The power consumption, therefore, represents a limiting factor for designing these low-cost devices.

We often communicate private information using the benefits of cloud and ubiquitous computing. Neither of the two technologies is designed with security in mind and our demands to conceal confidential information are now, more than ever, being manifested and become very important. By ensuring confidentiality, entity and data authentication, access control, privacy protection, and many other information security objectives, cryptography seems to be a natural choice for addressing the issue of security. Based on hard mathematical problems, cryptography often

requires highly intensive computations which, in fact, represent the main restriction for its wide application in cloud and ubiquitous computing. If not fast enough, cryptography is simply not accepted on the Internet. In order to be transparent while providing security and data integrity, cryptography needs to follow trends driven by the continuous need for high speed and low power. The complexity of cryptographic algorithms is illustrated in Table 1.1, where the current secret-key and public-key cryptographic standards are compared with respect to their algorithm-specific characteristics. The state and the operands size are devised to provide security equivalent of 128-bit symmetric key ('ECRYPT II Yearly Report on Algorithms and Keysizes' [1]).

Table 1.1: Complexity of the current cryptographic standards (equivalent symmetric key size of 128 bits†).

| Algorithm | AES-128 | SHA-256 | ECC | RSA |
|---|---|---|---|---|
| State/Operand Size [bit] | 128 | 256 | 256 | 3,248 |
| Critical Operation | S-box | Modular Addition | Modular Multiplication | Modular Multiplication |
| Number of Rounds/ Modular Multiplications | 10 | 64 | $\approx$ 3,000 | $\approx$ 5,000 |

†'ECRYPT II Yearly Report on Algorithms and Keysizes' [1].

The smallest variant of the Advanced Encryption Standard (AES-128) [39] requires only 10 rounds to encrypt a message block of 128 bits, having an S-box as the critical operation. This indeed is a reason why AES achieves rather good performance in hardware. As shown by Hodjat and Verbauwhede [70], a fully pipelined architecture of AES implemented in 0.18 $\mu$m technology approaches a throughput of 70 Gb/s. The fully pipelined architectures comes at high price though, requiring more than 250,000 NAND gate equivalences (GE). On the other hand, relatively small state of 128 bits results in rather compact implementation of AES, requiring only 3,100 GE in 0.13 $\mu$m CMOS technology (Hämäläinen et al. [62]).

The Secure Hash Standard (SHA-256) [76] requires 64 rounds to produce a digest of a 512-bit message block. The state size of SHA-256 is 256 bits, while the critical operation is modular addition. The performance loss in comparison to AES is obvious, and therefore the fastest hardware implementation of SHA-256 provides a throughput of 7,420 Mb/s in 0.13 $\mu$m CMOS technology (Dadda, Macchetti, and Owen [38]). Due to the larger state size and more complex circuitry of the round function, the smallest implementation of SHA-256 consumes 8,588 GE (Kim and Ryou [89]). As a side-note, we mention that by the end of 2012, the National Institute of Standards and Technology (NIST) will replace the current SHA-256 standard with the new SHA-3 standard [127].

Elliptic Curve Cryptography (ECC) [120, 99] over a finite field of size 256 bits provides security equivalent of 128-bit symmetric key. The basic operation of ECC is a point multiplication, an operation that is heavily based on modular multiplication, i.e. approximately 3,000 modular multiplications are necessary for performing one ECC-256 point multiplication. Implemented in hardware, the fastest point multiplication of ECC over 256-bit prime field needs approximately 40 $\mu$s on Xilinx Virtex-4 SX55 FPGA board, i.e. about 25,000 point multiplications per second (Güneysu and Paar [59]). This, however, comes at high price utilizing 24,574 logic slices and 512 DSP units of the board. The compact architectures of ECC reported in the literature are exclusively dealing with smaller fields, typically 163-bit binary fields. That is somewhat obvious choice due to the increased complexity of public-key circuitry. Therefore the best result in this area comes from Hein, Wolkerstorfer, and Felber [66] where the actual circuit performing ECC over GF($2^{163}$) was fabricated, consuming around 15,000 GE in 0.18 $\mu$m CMOS technology. The synthesis results of the same circuit without the key storage lead to the area consumption of 11,904 GE. Another notable result is a recent work of Lee et al. [103] where the ECC based security processor for RFID, implemented in 0.13 $\mu$m CMOS technology, requires 12,506 GE. Finally, the fastest implementation of ECC over GF($2^{163}$) reported up to date comes from Sakiyama [146] and requires only 12 $\mu$s for a single point multiplication, i.e. about 83,300 point multiplications per second. The circuit is implemented in 0.13 $\mu$m CMOS technology and requires approximately 154,000 GE.

Although ECC is becoming increasingly popular especially for low-cost devices, RSA [144] is still the most widely used public-key algorithm today. To achieve the security equivalent of 128-bit symmetric key, RSA needs to operate with large numbers – up to 3,248 bits. That requires approximately 5,000 modular multiplications for performing a single modular exponentiation, which is the basic operation of RSA. This indeed is a reason for RSA being relatively slow compared to other cryptographic primitives. State of the art commercially available solutions offer at most 50 1024-bit modular exponentiations per second, running above 300 MHz and consuming around 40,000 GE in 0.13 $\mu$m CMOS technology with additional 12 kbits of the RAM storage (Hellion ModExp Core, STD256 [3]). The smallest design, on the other hand, consumes less than 8,000 GE including 10 kbits of RAM, but can execute only 5 modular exponentiation per second (Helion ModExp Core, TINY32). The fastest result available in the literature comes from Zhao et al. [181], and reports a 1024-bit modular exponentiation within 157.4 $\mu$s, i.e. approximately 6,350 modular exponentiations per second. This, however, comes at large area overhead resulting in a circuit of 923,000 GE, which running at 140 MHz consumes 1.619 W in 0.18 $\mu$m CMOS technology. For the 3,248-bit RSA we expect the results to be significantly worse with respect to overall performance.

Today's broadband Internet is reaching the rates of 100 Gb/s (100 Gigabit Ethernet [4]), while the arrival of its successor, i.e. Terabit Ethernet, is on

the horizon already. As mentioned above, only AES with highly parallelized implementation is currently able to fully satisfy the needs of the global network. Other cryptographic standards are still too far away from the performance goals driven by the Internet.

Moreover, the current standard solutions do not meet the extreme constraints of RFIDs and therefore, it is even more important to design alternatives. In order to ensure security for tiny pervasive devices, e.g. Electronic Product Codes (EPC) [2], we may devote only several hundreds of gates of the chip for that purpose (Juels and Weis [82]). AES is again the only candidate that comes close to this bound, still requiring several thousands of gates instead.

The need for efficiency is therefore a driving force which considerably influences a design of cryptographic primitives. Being formally defined as an accomplishment of, or ability to accomplish a task with a minimum expenditure of time and energy, the efficiency is one of the key requirements in assuring that the cryptographic primitives become ubiquitous and available on multiple platforms. We consider security as the most important measure while still trying to achieve cutting edge performance of the cryptographic primitive. The challenge now becomes a multi-dimensional space where different trade-offs are to be explored. Moving along the axes of the aforementioned space is the main topic of this dissertation.

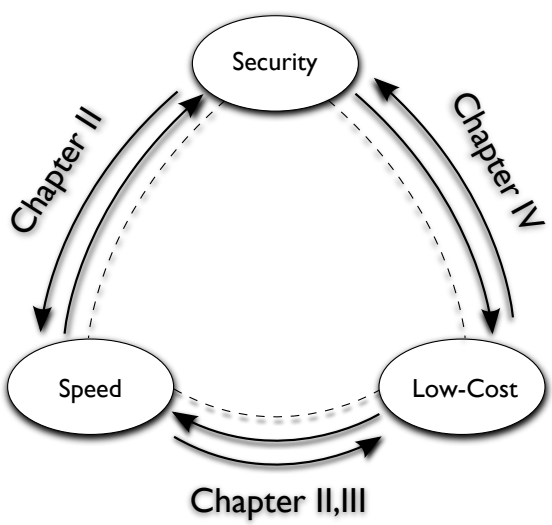


Figure 1.1: Outline of the thesis.

As outlined in Fig. 1.1, we dedicate three chapters of the thesis to investigate the trade-offs most of the engineers are faced with when implementing a cryptographic system. We explore a multitude of compromises, all having the ultimate goal of

ensuring a highly secure cryptosystem that runs at maximum achievable speed and consumes the minimum amount of energy. As the ultimate goal is unfortunately infeasible, we consider all the possible trade-offs and are only able to come close to an optimal implementation for a certain, specific application. We propose novelties on the algorithmic level, therefore enabling further performance improvements by exploring the lower levels of abstraction.

## 1.1 Outline and Summary of Contributions

This section outlines the structure of the thesis and details the personal contributions. The thesis is organized in five chapters.

**Chapter 1**: The first chapter provides a brief introduction to efficient implementations of cryptographic primitives. We also outline a summary and the contribution of each chapter separately.

**Chapter 2**: The second chapter entitled *"Efficient Hardware Implementations of Finite Field Arithmetic"* provides a brief introduction to modular arithmetic and brings forward the building blocks that are used later throughout the chapter. The main contributions of the chapter have been published in the proceedings of several peer-reviewed international conferences and international journals. Section 2.4 of this chapter deals with faster digit-serial modular multiplication based on the Barrett and Montgomery algorithms, representing the results published by Knežević, Vercauteren, and Verbauwhede [96] in the IEEE Transactions on Computers journal and by the same authors at the International Workshop on the Arithmetic of Finite Fields (WAIFI 2010) [97]. Section 2.5 discusses in detail how the bit-parallel version of a modular multiplication based on the Barrett and Montgomery algorithms can be performed without precomputation. The results are presented by Knežević, Sakiyama, Fan, and Verbauwhede [93] at WAIFI 2008, and by Knežević, Batina, and Verbauwhede [91] at the IEEE International Symposium on Circuits and Systems (ISCAS 2009). Finally, the last section of the chapter introduces a new modular multiplication algorithm, a so-called tripartite modular multiplication. The contribution of this work is a result of Sakiyama, Knežević, Fan, Preneel, and Verbauwhede [147] and will appear in Integration, the VLSI journal.

**Chapter 3**: The third chapter entitled *"High-Throughput Hardware Implementations of Cryptographic Hash Functions"* initially describes the basic throughput improving techniques that are widely used in digital signal processing (DSP) systems. The application of these, as well as several other algorithm-specific techniques are used later throughout the chapter for the purpose of high-throughput hardware implementations of cryptographic hash functions. Section 3.4 describes how the hardware implementation of the well-known RIPEMD-160 hash algorithm can be optimized by using iteration bound analysis and how its throughput can be

further increased by applying some of the algorithm-specific techniques. The contribution of this work has been published by Knežević, Sakiyama, Lee, and Verbauwhede [94] in the IEEE International Conference on Application-specific Systems, Architectures and Processors (ASAP 2008). Section 3.5 provides a comprehensive hardware evaluation of fourteen second-round SHA-3 candidates. This work by Knežević et al. [92] represents the final outcome of the collaboration of three universities and two international institues and will appear in the IEEE Transactions on Very Large Scale Integration (VLSI) Systems journal. In the last section of the chapter, as a result of the work published by Knežević and Verbauwhede [95] in the Workshop on Embedded Systems Security (WESS 2009), the hardware evaluation of the Luffa hash family is provided.

**Chapter 4**: The fourth chapter, entitled *"Lightweight Cryptography – A Battle for a Single Gate"*, deals with the concept of lightweight cryptography, specifically targeting the low-cost block ciphers. After mentioning related work, we present KATAN & KTANTAN – a family of small and efficient, hardware-oriented block ciphers. The contribution of this work has been published by De Cannière, Dunkelman, and Knežević [27] at the Workshop on Cryptographic Hardware and Embedded Systems (CHES 2009). We further provide an updated key schedule for the KTANTAN family, as a result of Bogdanov, Dunkelman, Knežević, Rechberger, and Verbauwhede [21]. The manuscript is submitted to the IEEE Communications Letters.

**Chapter 5**: The last chapter concludes and provides some directions for future work.

# Chapter 2

# Efficient Hardware Implementations of Finite Field Arithmetic

## 2.1 Introduction

Finite field arithmetic and its hardware and software implementations received considerable attention due to their extensive use in the theory of error correction codes. After Diffie and Hellman [44] introduced the concept of public-key cryptography (PKC) in the mid 70's, efficient implementations of the arithmetic of finite fields became a very exciting topic in this field.

The best-known public-key cryptosystems are based on factoring, i.e. RSA [144] and on the discrete logarithm problem in a large prime field (Diffie-Hellman [44], ElGamal [50], Schnorr [155], DSA [129]) or on an elliptic (hyper-elliptic) curve over a finite field (ECC/HECC) [120, 99]. Based on the hardness of the underlying mathematical problem, PKC usually deals with large numbers ranging from a few hundreds to a few thousands of bits in size ('ECRYPT II Yearly Report on Algorithms and Keysizes' [1]). Consequently, efficient implementations of PKC primitives have always been a challenge.

An efficient implementation of the aforementioned cryptosystems highly depends on the efficient implementation of modular arithmetic. More precisely, the modular multiplication forms the basis of a modular exponentiation which is the core operation of the RSA cryptosystem. It is also present in many other cryptographic algorithms including those based on ECC and HECC. Implementing

an efficient modular multiplication for PKC has been a great challenge for both software and hardware platforms because one has to deal with at least 1024-bit integers for RSA and 160 bits or more for ECC. Chapter 14 of the 'Handbook of Applied Cryptography' by Menezes, van Oorschot, and Vanstone [116] provides a comprehensive overview of common techniques for efficient implementations of modular arithmetic. Efficient hardware implementations of finite fields with applications in cryptography are further treated by Guajardo et al. in [58].

Figure 2.1 illustrates the main direction of this chapter. Some of the methods for accelerating modular multiplication, proposed in this chapter, require fixing a portion of bits of the RSA secret parameters. State of the art attacks on RSA benefit exactly from this fact and therefore the theoretical security of RSA decreases. This is why we claim that the trade-off between security and speed is scrutinized. However, we argue that, if the parameters are chosen carefully, the practical security of RSA remains unaffected against these attacks. On the other hand, a typical trade-off between a high-speed and a low-cost implementation is also considered by exploring different levels of parallelism and using a digit-serial approach. Furthermore, the tripartite modular multiplication, introduced in Section 2.6, comes with a new level of parallelism at the algorithmic level and therefore provides a possibility to fully explore the trade-off between speed and area.

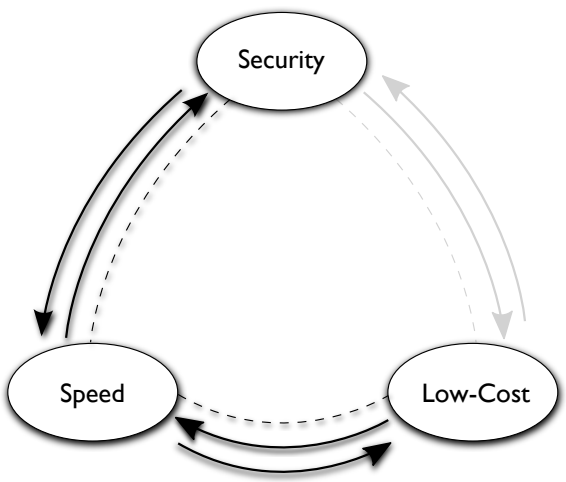


Figure 2.1: Security versus speed and speed versus low-cost trade-off.

After outlining some basics of finite field arithmetic, we discuss in detail our contributions. Section 2.4 deals with faster digit-serial modular multiplication based on the Barrett and Montgomery reduction methods. Besides accelerating

classical and Montgomery multiplication, we also show how bipartite modular multiplication can perform faster for a certain class of moduli. In Section 2.5, we discuss bit-parallel algorithms and reveal how they can be adapted for special classes of moduli, such that the precomputation can be omitted. Due to the importance of binary fields in a multitude of hardware architectures, we devote special attention to this class of algorithms as well. Finally, in Section 2.6 we introduce a new efficient method for modular multiplication, so-called, tripartite modular multiplication.

## 2.2 Preliminaries

This introductory section serves to provide some basic algebraic concepts that will be used throughout the chapter. We start with the basic definitions from the theory of finite fields. A detailed treatment of this topic can be found in 'Introduction to Finite Fields and Their Applications', by Lidl and Niederreiter [108].

Let $S$ be a set and let $S \times S$ denote the set of ordered pairs $(s, t)$ with $s, t \in S$. Then a mapping from $S \times S$ into $S$ will be called a (*binary*) *operation* on $S$.

**Definition 2.1.** A *group* is a set $G$ together with a binary operation * on $G$ such that the following three properties hold:

1. * is *associative*; that is, for any $a, b, c \in G$ holds $a * (b * c) = (a * b) * c$.
2. There is an *identity* (or *unity*) *element* $e$ in $G$ such that for all $a \in G$ holds $a * e = e * a = a$.
3. For each $a \in G$ there exists an inverse element $a^{-1} \in G$ such that $a * a^{-1} = a^{-1} * a = e$.

If the group also satisfies

4. For all $a, b \in G$ holds $a * b = b * a$,

then the group is called *abelian* (or *commutative*).

**Definition 2.2.** A *ring* $(R, +, \cdot)$ is a set $R$, together with two binary operations, denoted by $+$ and $\cdot$, such that:

1. $R$ is an abelian group with respect to $+$ .
2. $\cdot$ is associative – that is, $(a \cdot b) \cdot c = a \cdot (b \cdot c)$ for all $a, b, c \in R$.

3. The *distributive* laws hold; that is, for all $a, b, c \in R$ we have $a \cdot (b+c) = a \cdot b + a \cdot c$ and $(b + c) \cdot a = b \cdot a + c \cdot a$.

We call the operations + and · addition and multiplication, respectively, but we stress that those operations are not necessarily the ordinary operations with numbers. In the following convention, we use 0 (called the *zero element*) to denote the identity element of the abelian group $R$ with respect to addition.

A ring with a finite number of elements is called a *finite ring*. The ring of integers modulo an integer $M$, denoted with $\mathbb{Z}_M$, is a common example of a finite ring.

**Definition 2.3.**

(i) A ring is called a *ring with identity* if the ring has a multiplicative identity – that is, if there is an element $e$ such that $a \cdot e = e \cdot a = a$ for all $a \in R$.

(ii) A ring is called *commutative* if · is commutative.

(iii) A ring is called an *integral domain* if it is a commutative ring with identity $e \neq 0$ in which $a \cdot b = 0$ implies $a = 0$ or $b = 0$.

(iv) A ring is called a *division ring* if the nonzero elements of $R$ form a group under ·.

(v) A commutative division ring is called a *field*.

A field with finitely many elements is called a *finite field* or *Galois field*. The number of elements in the field represents the *order* of the field.

**Definition 2.4.** Let $F$ be a field. A subset $K$ of $F$ that is itself a field under the operations of $F$ will be called a *subfield* of $F$. In this context, $F$ is called an *extension field* of $K$. If $K \neq F$, we say that $K$ is a *proper subfield* of $F$.

**Definition 2.5.** A field containing no proper subfields is called a *prime field*.

The field of integers modulo a prime number $p$ is one of the most familiar examples of finite fields. It is also a prime field, often denoted as $\mathbb{F}_p$ or $\mathrm{GF}(p)$, and its extension field is denoted as $\mathbb{F}_{p^m}$ or $\mathrm{GF}(p^m)$ where $m \in \mathbb{N}$ denotes the *degree of the extension*. The prime $p$ is called the *characteristic* of $\mathrm{GF}(p^m)$. This field is of a fundamental importance and represents a basic building block for many cryptographic primitives.

Polynomials are very often used for representing the elements of finite fields. Let $\mathrm{GF}(p)$ be the finite field of characteristic $p$ and $\mathrm{GF}(p^m)$ its extension field of degree $m$. The extension field is defined with an irreducible polynomial $P(x) = \sum_{i=0}^{m} P_i x^i = (P_m \dots P_0)$, such that $P_i \in \mathrm{GF}(p)$. The elements of $\mathrm{GF}(p^m)$ are of the form $A(x) = \sum_{i=0}^{m-1} A_i x^i = (A_{m-1} \dots A_0)$ where $A_i \in \mathrm{GF}(p)$.

Given two elements $A(x), B(x) \in \mathrm{GF}(p^m)$, the addition in the extension field is now defined as:

$$C(x) = A(x) + B(x) \ ,$$

where $C_i = A_i + B_i \bmod p$.

Finite field multiplication can be defined using the above notation. Given an irreducible polynomial $P(x)$ and two elements $A(x), B(x) \in \mathrm{GF}(p^m)$, the finite field multiplication can be defined as:

$$A(x) \cdot B(x) = A(x)B(x) \bmod P(x) \ ,$$

where $A(x)B(x)$ is ordinary polynomial multiplication.

Furthermore, when dealing with the finite rings $\mathbb{Z}_M$, we can formally define a modular multiplication as follows. Given a modulus $M$ and two elements $A, B \in \mathbb{Z}_M$, the ordinary modular multiplication is defined as:

$$A \cdot B = AB \bmod M \ .$$

It is clear that for $M = p$, $p$ prime, $\mathbb{Z}_M$ is equivalent to $\mathrm{GF}(p)$ and the ordinary modular multiplication is in fact a finite field multiplication.

## 2.3 Efficient Modular Arithmetic

Modern computers usually have a word size of 8, 16, 32 or 64 bits but many other sizes have been used. The word sizes of 8 and 16 bits have especially been interesting for embedded computing platforms. However, for highly customized hardware designs, such as application specific integrated circuits (ASICs), an arbitrary word size is often used. In order to get familiar with the notation used throughout the chapter, we first review the word representation of the operands used in modular arithmetic.

A multiple-precision $n$-bit integer $A$ is represented in radix $r$ representation as $A = (A_{n_w-1} \dots A_0)_r$ where $r = 2^w$; $n_w$ represents the number of words and is equal to $\lceil n/w \rceil$ where $w$ is a *word-size*; $A_i$ is called a *digit* and $A_i = (a_{w-1} \dots a_0) \in [0, r-1]$. A special case is when $r = 2$ ($w = 1$) and the representation of $A = (a_{n-1} \dots a_0)_2$ is called a *bit* representation. A multiplication of two digits is further referred to as a *single-precision* (SP) multiplication. Sometimes we refer to it as a *digit* multiplication. We define a *partial product* as a product of a single digit and an $n_w$-digit integer. Similar to the case of integers, we also denote the $n$-bit polynomial $A(x)$ as $A(x) = \sum_{i=0}^{n_w-1} A_i x^{wi} = (A_{n_w-1} \dots A_0)$ where $A_i = \sum_{i=0}^{w-1} a_i x^i = (a_{w-1} \dots a_0)$.

In the previous section we defined addition and multiplication in finite fields. Other basic operations, such as subtraction, field inversion, and exponentiation are carried

out by using these two operations. The complexity of basic operations in GF($p^m$) is summarized in Table 2.1 ('Handbook of Applied Cryptography', Chapter 14 [116]). Additionally, we assume that the size of $p$ is $n$ bits.

Table 2.1: Complexity of basic operations in GF($p^m$).

| Operation | | Number of GF($p$) operations | Total complexity |
|---|---|---|---|
| Addition | $A(x) + B(x)$ | $\mathcal{O}(m)$ | $\mathcal{O}(mn)$ |
| Subtraction | $A(x) - B(x)$ | $\mathcal{O}(m)$ | $\mathcal{O}(mn)$ |
| Multiplication | $A(x) \cdot B(x)$ | $\mathcal{O}(m^2)$ | $\mathcal{O}(m^2n^2)$ |
| Inversion | $A(x)^{-1}$ | $\mathcal{O}(m^2)$ | $\mathcal{O}(m^2n^2)$ |
| Exponentiation | $A(x)^k, k < p^m$ | $\mathcal{O}(m^3n)$ | $\mathcal{O}(m^3n^3)$ |

Since finite field multiplication (modular multiplication, in general) is considerably more expensive than addition, it is indeed the right operation to be optimized. There exist many different algorithms for modular multiplication and depending on the implementation type, we can distinguish two main groups:

- *Bit-parallel algorithms*: these are the algorithms highly optimized for speed, which calculate the result with a time complexity $\mathcal{O}(n)$ where $n$ is the number of bits of the operands. Their main disadvantage is a large area overhead with a complexity of $\mathcal{O}(n^2)$. They often result in high-speed, but very expensive implementations. Due to the large operands used in e.g. RSA and ECC, these algorithms are often impractical on the majority of embedded platforms.
- *Digit-serial algorithms*: sometimes referred to as *sequential algorithms*, these algorithms trade speed for area. Multiplication is interleaved with reduction, and hence the whole calculation is spread over many cycles, resulting in a more compact, but slower implementation.

In this section, we outline the classical modular multiplication algorithm and two algorithms that had a significant impact on the efficient implementation of modular arithmetic – due to Barrett and Montgomery. As mentioned above, we distinguish between bit-parallel and digit-serial algorithms. Therefore, we outline the algorithms first in bit-parallel and then in their digit-serial form.

### 2.3.1 Bit-Parallel Algorithms

Algorithm 1 summarizes the classical modular multiplication in its bit-parallel form. The calculation of the intermediate quotient $q_C$ at step two of the algorithm is done

**Algorithm 1** Classical modular multiplication algorithm (bit-parallel version).

INPUT: $A = (a_{n-1} \dots a_0)$, $B = (b_{n-1} \dots b_0)$, $M = (m_{n-1} \dots m_0)$ where $0 \leq A, B < M$, $2^{n-1} \leq M < 2^n$.
OUTPUT: $C = AB \bmod M$.
1: $C \Leftarrow AB$
2: $q_C \Leftarrow \lfloor C/M \rfloor$
3: $C \Leftarrow C - q_C M$
4: Return $C$.

by utilizing integer division, which is an expensive operation. The idea of using the precomputed reciprocal of the modulus $M$ and simple shift and multiplication operations instead of division originally comes from Barrett [18]. To explain the basic idea, we rewrite the intermediate quotient $q_C$ as:

$$q_C = \left\lfloor \frac{C}{M} \right\rfloor = \left\lfloor \frac{\frac{C}{2^{n-1}} \frac{2^{2n}}{M}}{2^{n+1}} \right\rfloor \geq \left\lfloor \frac{\left\lfloor \frac{C}{2^{n-1}} \right\rfloor \left\lfloor \frac{2^{2n}}{M} \right\rfloor}{2^{n+1}} \right\rfloor = \hat{q} \ . \tag{2.1}$$

The value $\hat{q}$ represents an estimation of $q_C$. In most cryptographic applications, the modulus $M$ is fixed during many modular multiplications and hence the value $\lfloor 2^{2n}/M \rfloor$ can be precomputed and reused multiple times. Furthermore, an integer division by a power of 2 is a simple shift operation. Since the value of $\hat{q}$ is an estimated value, several correction steps at the end of the modular multiplication have to be performed. In what follows, we shall often refer to the classical modular multiplication based on Barrett reduction simply as Barrett multiplication.

In 1985, Montgomery [121] opened a new direction in the field of efficient modular multiplication. The algorithm became widely used and is certainly one of the most deployed algorithms today. We first describe its bit-parallel version as provided in Alg. 2. Given an $n$-bit odd modulus $M$ and an integer $U \in \mathbb{Z}_M$, the image or the Montgomery residue of $U$ is defined as $A = UR \bmod M$ where $R$, the Montgomery radix, is a constant relatively prime to $M$. If $A$ and $B$ are, respectively, the images of $U$ and $V$, the Montgomery multiplication of these two images is defined as:

$$A * B = ABR^{-1} \bmod M \ .$$

The result is the image of $UV \bmod M$ and needs to be converted back at the end of the process. For the sake of efficient implementation, one usually uses $R = 2^n$. Similar to Barrett multiplication, this algorithm uses a precomputed value $M' = -M^{-1} \bmod R$.

**Algorithm 2** Montgomery multiplication algorithm (bit-parallel version).

INPUT: $A = (a_{n-1} \dots a_0)$, $B = (b_{n-1} \dots b_0)$, $M = (m_{n-1} \dots m_0)$, $M' = -M^{-1} \bmod R$ where $0 \leq A, B < M$, $2^{n-1} \leq M < 2^n$, $R = 2^n$, and $\gcd(M, R)$=1.
OUTPUT: $C = ABR^{-1} \bmod M$.
1: $C \Leftarrow AB$
2: $q_M \Leftarrow CM' \bmod R$
3: $C \Leftarrow (C + q_M M)/R$
4: **if** $C \geq M$ **then**
5: $\quad C \Leftarrow C - M$
6: **end if**
7: Return $C$.

### 2.3.2 Digit-Serial Algorithms

Let the modulus $M$ be an $n_w$-digit integer, where the radix of each digit is $r = 2^w$. The classical, digit-serial, modular multiplication algorithm computes $AB \bmod M$ by interleaving the multiplication and modular reduction phases as it is shown in Alg. 3.

**Algorithm 3** Classical modular multiplication algorithm (digit-serial version).

INPUT: $A = (A_{n_w-1} \dots A_0)_r$, $B = (B_{n_w-1} \dots B_0)_r$, $M = (M_{n_w-1} \dots M_0)_r$ where $0 \leq A, B < M$, $2^{n-1} \leq M < 2^n$, $r = 2^w$ and $n_w = \lceil n/w \rceil$.
OUTPUT: $C = AB \bmod M$.
1: $C \Leftarrow 0$
2: **for** $i = n_w - 1$ downto 0 **do**
3: $\quad C \Leftarrow Cr + AB_i$
4: $\quad q_C \Leftarrow \lfloor C/M \rfloor$
5: $\quad C \Leftarrow C - q_C M$
6: **end for**
7: Return $C$.

Similar to its bit-parallel version, the calculation of the intermediate quotient $q_C$ at step four of the algorithm can also be done by following the approach of Barrett [18]. In his thesis, Dhem [42] generalizes this idea and provides the following relation:

$$q_C = \left\lfloor \frac{C}{M} \right\rfloor = \left\lfloor \frac{\frac{C}{2^{n+\beta}} \frac{2^{n+\alpha}}{M}}{2^{\alpha-\beta}} \right\rfloor \geq \left\lfloor \frac{\left\lfloor \frac{C}{2^{n+\beta}} \right\rfloor \left\lfloor \frac{2^{n+\alpha}}{M} \right\rfloor}{2^{\alpha-\beta}} \right\rfloor = \hat{q} \ . \tag{2.2}$$

He determines the values of $\alpha = w + 3$ and $\beta = -2$ for which Barrett multiplication needs at most one correction step at the end of the algorithm. Again, the modulus $M$ is fixed during many modular multiplications and hence the value $\mu = \lfloor 2^{n+\alpha}/M \rfloor$ is precomputed and reused multiple times.

Montgomery's algorithm has a sequential version as well. The algorithm is summarized in Alg. 4.

**Algorithm 4** Montgomery multiplication algorithm (digit-serial version).

INPUT: $A = (A_{n_w-1} \dots A_0)_r$, $B = (B_{n_w-1} \dots B_0)_r$, $M = (M_{n_w-1} \dots M_0)_r$, $M' = -M_0^{-1} \bmod r$ where $0 \leq A, B < M$, $2^{n-1} \leq M < 2^n$, $r = 2^w$, $\gcd(M, r)$=1 and $n_w = \lceil n/w \rceil$.
OUTPUT: $C = ABr^{-n_w} \bmod M$.
1: $C \Leftarrow 0$
2: **for** $i = 0$ to $n_w - 1$ **do**
3: $\quad C \Leftarrow C + AB_i$
4: $\quad q_M \Leftarrow CM' \bmod r$
5: $\quad C \Leftarrow (C + q_M M)/r$
6: **end for**
7: **if** $C \geq M$ **then**
8: $\quad C \Leftarrow C - M$
9: **end if**
10: Return $C$.

## 2.4 Faster Digit-Serial Modular Multiplication Based on Barrett and Montgomery Reduction Methods

### Publication Data

M. Knežević, F. Vercauteren, and I. Verbauwhede, "Faster Interleaved Modular Multiplication Based on Barrett and Montgomery Reduction Methods," *IEEE Transactions on Computers*, vol. 59, no. 12, pp. 1715–1721, 2010.

#### Personal Contributions

- Principal author.

Our novel contribution consists of proposing two interleaved modular multiplication algorithms based on Barrett and Montgomery reductions. Four large sets of moduli for which the novel algorithms apply are proposed and analyzed from a security point of view. We propose a hardware architecture for the modular multiplier that is based on our methods. The results show that, concerning the speed, our proposed architecture outperforms the modular multiplier based on standard

modular multiplication. Additionally, our design consumes less area compared to the standard solutions.

### 2.4.1 Related Work

Before introducing related work we note here that for the moduli used in all common ECC cryptosystems, the modular reduction can be done much faster than the one proposed by Barrett or Montgomery, even without any multiplication. This is the main reason behind standardizing generalized Mersenne prime moduli (sums/differences of a few powers of 2). Standards such as FIPS 186-3 [129], ANSI [10], and SEC2 [158] address this topic in detail.

The idea of simplifying the intermediate quotient evaluation was first presented by Quisquater [139] at the rump session of Eurocrypt '90. The method is similar to the one of Barrett except that the modulus $M$ is preprocessed before the modular multiplication in such a way that the evaluation of the intermediate quotient basically comes for free. Preprocessing requires some extra memory and computational time, but the latter is negligible when many modular multiplications are performed using the same modulus.

Lenstra [107] points out that choosing moduli with a predetermined portion is beneficial both for storage and computational requirements. He proposes a way to generate RSA moduli with any number of predetermined leading (trailing) bits, with the fraction of specified bits only limited by security considerations. Furthermore, Lenstra discusses security issues and concludes that the resulting moduli do not seem to offer less security than regular RSA moduli. In [81], Joye enhances the method for generating RSA moduli with a predetermined portion proposed by Lenstra in [107].

In [64], Hars proposes a long modular multiplication method that also simplifies an intermediate quotient evaluation. The method is based on Quisquater's algorithm and requires preprocessing of the modulus by increasing its length. The algorithm contains conditional branches that depend on the sign of the intermediate remainder. This increases the complexity of the algorithm, especially concerning the hardware implementation where additional control logic needs to be added.

### 2.4.2 The Proposed Modular Multiplication Methods for Integers

In this section we propose four sets of moduli that specifically target efficient modular multiplications by means of the classical modular multiplication based on Barrett reduction and Montgomery multiplication. In addition to simplified quotient evaluation, our algorithms do not require any additional preprocessing.

The algorithms are simple and especially suitable for hardware implementations. They contain no conditional branches inside the loop and hence require a very simple control logic.

The methods describing how to generate such moduli in case of RSA are discussed by Lenstra [107] and Joye [81]. Furthermore, from the sets proposed in this section one can also choose the primes that generate the RSA modulus to speed up decryption of RSA by means of the Chinese Remainder Theorem (CRT). In Section 2.4.8, we discuss security issues concerning this optimisation.

In both Barrett and Montgomery multiplications, the precomputed values of either the modulus reciprocal ($\mu$) or the modulus inverse ($M'$) are used in order to avoid multiple-precision divisions. However, single-precision multiplications still need to be performed (step four of Alg. 3 and Alg. 4). This especially concerns hardware implementations, as multiplication with the precomputed values often occurs within the critical path of the whole design. Section 2.4.6 discusses this issue in more detail.

Let us, for now, assume that the precomputed values $\mu$ and $M'$ are both of type $\pm 2^{\delta} - \varepsilon$ where $\delta \in \mathbb{Z}$ and $\varepsilon \in \{0, 1\}$. By tuning $\mu$ and $M'$ to be of this special type, we transform a single-precision multiplication with these values into a simple shift operation in hardware. Therefore, we find sets of moduli for which the precomputed values are both of type $\pm 2^{\delta} - \varepsilon$.

### 2.4.3 Speeding Up Classical Modular Multiplication

In what follows, we assume that the classical modular multiplication is implemented by means of Alg. 3, where step four of the algorithm is evaluated as summarized in Eq. 2.2. To make the following discussion easier, we provide an analysis of Alg. 3 that is based on the work of Dhem in [42].

*Analysis of Alg. 3.* We assume that step four of Alg. 3 is performed according to Eq. 2.2. Let us first consider the first iteration of Alg. 3 ($i = 0$). We can find an integer $\gamma$ such that $C_0 = AB_{n_w-1} < 2^{n+\gamma}$. This represents an upper bound of $C$ ($C_0$ for $i = 0$). The quotient $q = \left\lfloor \frac{C_0}{M} \right\rfloor$ can now be written as

$$q = \left\lfloor \frac{C_0}{M} \right\rfloor = \left\lfloor \frac{\frac{C_0}{2^{n+\beta}} \frac{2^{n+\alpha}}{M}}{2^{\alpha-\beta}} \right\rfloor ,$$

where $\alpha$ and $\beta$ are two variables. The estimation of the given quotient is now equal to

$$\hat{q} = \left\lfloor \frac{\left\lfloor \frac{C_0}{2^{n+\beta}} \right\rfloor \left\lfloor \frac{2^{n+\alpha}}{M} \right\rfloor}{2^{\alpha-\beta}} \right\rfloor = \left\lfloor \frac{\left\lfloor \frac{C_0}{2^{n+\beta}} \right\rfloor \mu}{2^{\alpha-\beta}} \right\rfloor ,$$

where $\mu = \left\lfloor \frac{2^{n+\alpha}}{M} \right\rfloor$ is constant and may be precomputed. Let us now define the quotient error as a function of the variables $\alpha$, $\beta$ and $\gamma$

$$e = e(\alpha, \beta, \gamma) = q - \hat{q} \ .$$

Since $\frac{X}{Y} \geq \left\lfloor \frac{X}{Y} \right\rfloor > \frac{X}{Y} - 1$ for any $X, Y \in \mathbb{Z}$, we can write the following inequality

$$\begin{aligned} q = \left\lfloor \frac{C_0}{M} \right\rfloor \geq \hat{q} &> \frac{\left\lfloor \frac{C_0}{2^{n+\beta}} \right\rfloor \left\lfloor \frac{2^{n+\alpha}}{M} \right\rfloor}{2^{\alpha-\beta}} - 1 \\ &> \frac{\left(\frac{C_0}{2^{n+\beta}} - 1\right)\left(\frac{2^{n+\alpha}}{M} - 1\right)}{2^{\alpha-\beta}} - 1 \\ &= \frac{C_0}{M} - \frac{C_0}{2^{n+\alpha}} - \frac{2^{n+\beta}}{M} + \frac{1}{2^{\alpha-\beta}} - 1 \\ &\geq \left\lfloor \frac{C_0}{M} \right\rfloor - \frac{C_0}{2^{n+\alpha}} - \frac{2^{n+\beta}}{M} + \frac{1}{2^{\alpha-\beta}} - 1 \\ &= q - \frac{C_0}{2^{n+\alpha}} - \frac{2^{n+\beta}}{M} + \frac{1}{2^{\alpha-\beta}} - 1 \ . \end{aligned}$$

Now, since $e \in \mathbb{Z}$, the quotient error can be estimated as

$$e = e(\alpha, \beta, \gamma) \leq \left\lfloor 1 + \frac{C_0}{2^{n+\alpha}} + \frac{2^{n+\beta}}{M} - \frac{1}{2^{\alpha-\beta}} \right\rfloor \ .$$

According to Alg. 3 we have $C_0 < 2^{n+\gamma}$ and $M \geq 2^{n-1}$. Hence, we can evaluate the quotient error as

$$e = e(\alpha, \beta, \gamma) \leq \left\lfloor 1 + 2^{\gamma-\alpha} + 2^{\beta+1} - \frac{1}{2^{\alpha-\beta}} \right\rfloor \ .$$

Following the previous inequality, it is obvious that for $\alpha \geq \gamma + 1$ and $\beta \leq -2$ we have $e = 1$.

Next, we need to ensure that the intermediate remainder $C_i$ does not grow uncontrollably as $i$ increases. Since $A < M$, $B_i < 2^w$, $C_i < M + eM$ and $M < 2^n$, after $i$ iterations we have

$$\begin{aligned} C_i &= C_{i-1} 2^w + AB_i \\ &< (M + eM)2^w + M2^w \\ &< (2 + e)2^{n+w} \ . \end{aligned}$$

Since we want to use the same value for $e$ during the algorithm, the next condition must hold

$$C_i < (2 + e)2^{n+w} < 2^{n+\gamma} \ .$$

To minimize the quotient error ($e = 1$), we must choose $\gamma$ such that

$$3 \cdot 2^w < 2^\gamma \ .$$

In other words, we choose $\gamma \geq w + 2$. Now, according to the previous analysis we conclude that for $\alpha \geq \gamma + 1$, $\beta \leq -2$ and $\gamma \geq w + 2$ we may realize a modular multiplication with only one correction step at the end of the whole process. □

It is interesting here to evaluate the size of the intermediate quotient $\hat{q}$ and the precomputed value $\mu$ as a function of the parameters $\alpha$ and $\beta$. As will be seen later, this indeed represents a major drawback of the generalized Barrett multiplication and is a reason why Montgomery's method is superior in this case. Let us write the following relation:

$$\mu = \left\lfloor \frac{2^{n+\alpha}}{M} \right\rfloor < \left\lfloor \frac{2^{n+\alpha}}{2^{n-1}} \right\rfloor = 2^{\alpha+1} \ .$$

Obviously, the size of $\mu$ is at most $\lambda = \alpha + 1$ bits. Similarly, we can evaluate the size of $\hat{q}$:

$$\hat{q} = \left\lfloor \frac{\left\lfloor \frac{C}{2^{n+\beta}} \right\rfloor \left\lfloor \frac{2^{n+\alpha}}{M} \right\rfloor}{2^{\alpha-\beta}} \right\rfloor < \left\lfloor \left\lfloor \frac{C}{2^{n+\beta}} \right\rfloor 2^{\beta+1} \right\rfloor < \left\lfloor \left\lfloor \frac{3 \cdot 2^{n+w}}{2^{n+\beta}} \right\rfloor 2^{\beta+1} \right\rfloor \ ,$$

which, for $\beta \leq w + 1$, can be further simplified to $\hat{q} < 3 \cdot 2^{w+1}$. If chosen as suggested earlier, the parameters $\alpha = w + 3$ and $\beta = -2$ determine the size of $\hat{q}$ and $\mu$ to be at most $w + 3$ bits and $\lambda = w + 4$ bits, respectively. This introduces an additional overhead for software implementations, while it can be easily overcome in hardware.

Before describing the proposed algorithm, we provide two lemmata to make the following discussion easier.

**Lemma 2.1.** *Let $M = 2^n - \Delta$ be an $n$-bit positive integer and let $\mu = \lfloor 2^{n+\alpha}/M \rfloor$ where $\alpha \in \mathbb{N}$. If $0 < \Delta \leq \lfloor \frac{2^n}{1+2^\alpha} \rfloor$, then $\mu = 2^\alpha$.*

*Proof.* Rewrite $2^{n+\alpha}$ as $2^{n+\alpha} = M2^\alpha + 2^\alpha\Delta$. Since it is given that $0 < \Delta \leq \lfloor \frac{2^n}{1+2^\alpha} \rfloor$, we conclude that $0 < 2^\alpha\Delta < M$. By definition of Euclidean division, this shows that $\mu = 2^\alpha$. □

**Lemma 2.2.** *Let $M = 2^{n-1} + \Delta$ be an $n$-bit representation and let $\mu = \lfloor 2^{n+\alpha}/M \rfloor$ where $\alpha \in \mathbb{N}$. If $0 < \Delta \leq \lfloor \frac{2^{n-1}}{2^{\alpha+1}-1} \rfloor$, then $\mu = 2^{\alpha+1} - 1$.*

*Proof.* Rewrite $2^{n+\alpha}$ as $2^{n+\alpha} = M(2^{\alpha+1}-1) + 2^{n-1} - \Delta(2^{\alpha+1}-1)$. Since $0 < \Delta \leq \lfloor \frac{2^{n-1}}{2^{\alpha+1}-1} \rfloor$, we conclude that $0 \leq 2^{n-1} - \Delta(2^{\alpha+1}-1) < M$. By definition of Euclidean division this shows that $\mu = 2^{\alpha+1} - 1$. □

The interleaved modular multiplication algorithm based on general Barrett reduction is given in Section 2.3.2. Now, according to Lemmata 2.1 and 2.2 we can define two sets of moduli for which the modular multiplication based on Barrett reduction can be improved. These sets are of type

$$\begin{aligned} \mathrm{S}_1: &\quad M = 2^n - \Delta_1 \qquad \text{where } 0 < \Delta_1 \leq \lfloor \frac{2^n}{1+2^\alpha} \rfloor \ ; \\ \mathrm{S}_2: &\quad M = 2^{n-1} + \Delta_2 \quad \text{where } 0 < \Delta_2 \leq \lfloor \frac{2^{n-1}}{2^{\alpha+1}-1} \rfloor \ . \end{aligned} \tag{2.3}$$

Figure 2.2 further illustrates the properties of the two proposed sets $\mathrm{S}_1$ and $\mathrm{S}_2$. As we can see from the figure, approximately $\alpha$ bits of the modulus are fixed to be all 0's or all 1's, while the other $n-\alpha$ bits are arbitrarily chosen[1].

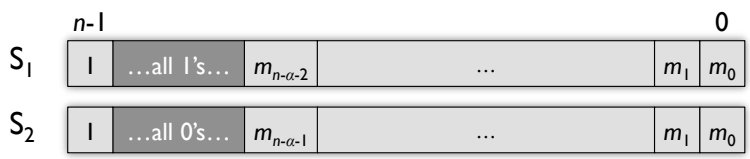


Figure 2.2: Binary representation of the proposed sets $\mathrm{S}_1$ and $\mathrm{S}_2$.

The proposed modular multiplication algorithm is shown in Alg. 5. The parameters $\alpha$ and $\beta$ are important for the quotient evaluation. As we show later, to minimize the error in quotient evaluation, $\alpha$ and $\beta$ are chosen such that $\alpha = w + 3$ and $\beta = -2$.

In contrast to Barrett multiplication where the quotient is evaluated as $\hat{q} = \left\lfloor \frac{\lfloor \frac{C}{2^{n+\beta}} \rfloor \lfloor \frac{2^{n+\alpha}}{M} \rfloor}{2^{\alpha-\beta}} \right\rfloor$, in our proposed algorithm the evaluation basically comes for free:

$$\hat{q} = \begin{cases} \lfloor \frac{C}{2^n} \rfloor & \text{if } M \in \mathrm{S}_1 \ ; \\ \lfloor \frac{C}{2^{n-1}} \rfloor & \text{if } M \in \mathrm{S}_2 \ . \end{cases}$$

This saves one single-precision multiplication and additionally increases the speed of the proposed modular multiplication algorithm.

[1] If $m_{n-\alpha-2} = 1$ for $M \in \mathrm{S}_1$ ($m_{n-\alpha-1} = 0$ for $M \in \mathrm{S}_2$), then the remaining $n-\alpha-2$ ($n-\alpha-1$) least significant bits can be arbitrarily chosen. Otherwise, if $m_{n-\alpha-2} = 0$ ($m_{n-\alpha-1} = 1$), then the remaining $n-\alpha-2$ ($n-\alpha-1$) least significant bits are chosen such that Eq. 2.3 is satisfied.

**Algorithm 5** Proposed interleaved modular multiplication based on generalized Barrett reduction ($\alpha = w + 3$ and $\beta = -2$).

INPUT: $A = (A_{n_w-1} \dots A_0)_r$, $B = (B_{n_w-1} \dots B_0)_r$, $M \in \mathrm{S}_1 \cup \mathrm{S}_2$ where $0 \leq A, B < M$, $r = 2^w$ and $n_w = \lceil n/w \rceil$.
OUTPUT: $C = AB \bmod M$.

```
C ⇐ 0
for i = n_w − 1 downto 0 do
    C ⇐ Cr + AB_i
    q̂ = ⌊C / 2^n⌋      if M ∈ S_1
        ⌊C / 2^(n−1)⌋  if M ∈ S_2
    C ⇐ C − q̂M
end for
if C ≥ M then
    C ⇐ C − M // At most 1 subtraction is needed.
end if
while C < 0 do
    C ⇐ C + M // At most 2 additions are needed.
end while
return C.
```

*Proof of Alg. 5.* To prove the correctness of the algorithm, we need to show that there exist $\alpha, \beta \in \mathbb{Z}$, such that $\hat{q}$ can indeed be represented as

$$\hat{q} = \begin{cases} \left\lfloor \frac{C}{2^n} \right\rfloor & \text{if } M \in \mathrm{S}_1 \ ; \\ \left\lfloor \frac{C}{2^{n-1}} \right\rfloor & \text{if } M \in \mathrm{S}_2 \ . \end{cases}$$

As shown in the analysis of Alg. 3, to have the minimized quotient error, the parameters $\alpha$ and $\beta$ need to be chosen such that $\alpha \geq w + 3$ and $\beta \leq -2$. Let us first assume that $M \in \mathrm{S}_1$. According to Lemma 2.1 it follows that $\mu = 2^\alpha$. Now, $\hat{q}$ becomes equal to

$$\hat{q} = \left\lfloor \frac{\left\lfloor \frac{C}{2^{n+\beta}} \right\rfloor \mu}{2^{\alpha-\beta}} \right\rfloor = \left\lfloor \frac{\left\lfloor \frac{C}{2^{n+\beta}} \right\rfloor 2^\alpha}{2^{\alpha-\beta}} \right\rfloor = \left\lfloor \left\lfloor \frac{C}{2^{n+\beta}} \right\rfloor 2^\beta \right\rfloor \ .$$

For $\beta \leq 0$ the previous equation becomes equivalent to

$$\hat{q} = \left\lfloor \frac{C}{2^n} \right\rfloor \ .$$

For the case where $M \in \mathrm{S}_2$ we have, according to Lemma 2.2, that $\mu = 2^{\alpha+1} - 1$. Now, $\hat{q}$ becomes equal to

$$\hat{q} = \left\lfloor \frac{\left\lfloor \frac{C}{2^{n+\beta}} \right\rfloor (2^{\alpha+1} - 1)}{2^{\alpha-\beta}} \right\rfloor = \left\lfloor \left\lfloor \frac{C}{2^{n+\beta}} \right\rfloor 2^{\beta+1} \left(1 - \frac{1}{2^{\alpha+1}}\right) \right\rfloor \ .$$

To further simplify the proof we choose $\beta = -2$ and the previous equation becomes equivalent to

$$\hat{q} = \left\lfloor \left\lfloor \frac{C}{2^{n-2}} \right\rfloor \frac{1}{2}\left(1 - \frac{1}{2^{\alpha+1}}\right) \right\rfloor \ .$$

If we choose $\alpha$ such that

$$2^{\alpha+1} > \max\left\{ \left\lfloor \frac{C}{2^{n-2}} \right\rfloor \right\} \ , \tag{2.4}$$

the expression of $\hat{q}$ simplifies to

$$\hat{q} = \begin{cases} \left\lfloor \frac{C}{2^{n-1}} \right\rfloor - 1 & \text{if } 2 \mid \left\lfloor \frac{C}{2^{n-2}} \right\rfloor \ ; \\ \left\lfloor \frac{C}{2^{n-1}} \right\rfloor & \text{if } 2 \nmid \left\lfloor \frac{C}{2^{n-2}} \right\rfloor \ . \end{cases} \tag{2.5}$$

The inequality (2.4) can be written as

$$2^{\alpha+1} > \left\lfloor \frac{\max\{C\}}{2^{n-2}} \right\rfloor \ ,$$

where $\max\{C\}$ is evaluated in the analysis of Alg. 3 and given as $\max\{C\} = (2+e)2^{n+w}$. To have the minimal error, we choose $e = 1$ and get the following relation

$$2^{\alpha+1} > \left\lfloor \frac{3 \cdot 2^{n+w}}{2^{n-2}} \right\rfloor = \left\lfloor 3 \cdot 2^{w+2} \right\rfloor \ .$$

The latter inequality is satisfied for $\alpha \geq w + 3$.

If instead of Eq. 2.5 we use only $\hat{q} = \left\lfloor \frac{C}{2^{n-1}} \right\rfloor$, the evaluation of the intermediate quotient $\hat{q}$ will, for $2 \mid \left\lfloor \frac{C}{2^{n-2}} \right\rfloor$, become greater than or equal to the real intermediate quotient $q$. Due to this fact $C$ can become negative at the end of the current iteration. Hence, we need to consider the case where $C < 0$. Let us prevent $C$ from an uncontrollable decrease by putting a lower bound with $C > -2^{n+\gamma}$ where $\gamma \in \mathbb{Z}$. Since $\frac{X}{Y} \geq \left\lfloor \frac{X}{Y} \right\rfloor > \frac{X}{Y} - 1$ for any $X, Y \in \mathbb{Z}$, we can write the following

inequality (note that $C < 0$ and $M > 0$)

$$\begin{aligned}
\hat{q} = \left\lfloor \frac{\left\lfloor \frac{C}{2^{n+\beta}} \right\rfloor \left\lfloor \frac{2^{n+\alpha}}{M} \right\rfloor}{2^{\alpha-\beta}} \right\rfloor &\leq \frac{\left\lfloor \frac{C}{2^{n+\beta}} \right\rfloor \left\lfloor \frac{2^{n+\alpha}}{M} \right\rfloor}{2^{\alpha-\beta}} \\
&< \frac{\frac{C}{2^{n+\beta}} \left( \frac{2^{n+\alpha}}{M} - 1 \right)}{2^{\alpha-\beta}} \\
&= \frac{C}{M} - \frac{C}{2^{n+\alpha}} \\
&< \left\lfloor \frac{C}{M} \right\rfloor + 1 - \frac{C}{2^{n+\alpha}} \\
&= q + 1 - \frac{C}{2^{n+\alpha}} \\
&< q + 1 + 2^{\gamma-\alpha} \ .
\end{aligned}$$

Now, since $q$, $\hat{q}$, $e \in \mathbb{Z}$, we choose $\alpha \geq \gamma + 1$ and the quotient error gets estimated as $-1 \leq e \leq 0$. If in the next iteration it again happens that $2 \mid \left\lfloor \frac{C}{2^{n-2}} \right\rfloor$, the quotient error will become $-2 \leq e \leq 0$.

Finally, to assure that $C$ will remain within the bounds during the $i$-th iteration we write

$$\begin{aligned}
C_i &= C_{i-1} 2^w + AB_i \\
&= (C_{i-2} - qM + eM) 2^w + AB_i \\
&> (0 + eM) 2^w + 0 \\
&> e 2^{n+w} > -2^{n+\gamma} \ .
\end{aligned}$$

The worst case is when $e = -2$ and then it must hold $\gamma > w + 1$. By choosing $\alpha = w + 3$ and $\beta = -2$ all conditions are satisfied and hence, $\hat{q}$ is indeed a good estimate of $q$. At most one subtraction or 2 additions at the correction step are required to obtain $C = AB \bmod M$. □

### 2.4.4 Speeding Up Montgomery Multiplication

Similar to Lemmata 2.1 and 2.2 we also provide Lemmata 2.3 and 2.4 that are at the heart of the proposed modular multiplication algorithm based on Montgomery reduction.

**Lemma 2.3.** *Let $M = \Delta 2^w + 1$ be an $n$-bit positive integer, i.e. $2^{n-w-1} \leq \Delta < 2^{n-w}$, and let $M' = -M^{-1} \bmod 2^w$ where $w \in \mathbb{N}$, then $M' = -1$.*

*Proof.* Since $M = 1 \bmod 2^w$ we clearly have $-M^{-1} = -1 \bmod 2^w$. □

**Lemma 2.4.** *Let $M = \Delta 2^w - 1$ be an $n$-bit positive integer, i.e. $2^{n-w-1} < \Delta \leq 2^{n-w}$ and let $M' = -M^{-1} \bmod 2^w$ where $w \in \mathbb{N}$, then $M' = 1$.*

*Proof.* Since $M = -1 \bmod 2^w$ we clearly have $-M^{-1} = 1 \bmod 2^w$. □

According to the previous two lemmata we can easily find two sets of moduli for which the precomputational step in Montgomery multiplication can be excluded. The resulting algorithm is shown in Alg. 6. The proposed sets are of type

$$\begin{array}{lll} \mathrm{S}_3: & M = \Delta_3 2^w + 1 & \text{where } 2^{n-w-1} \leq \Delta_3 < 2^{n-w} \ ; \\ \mathrm{S}_4: & M = \Delta_4 2^w - 1 & \text{where } 2^{n-w-1} < \Delta_4 \leq 2^{n-w} \ . \end{array} \tag{2.6}$$

Figure 2.3 further illustrates the properties of the two proposed sets $\mathrm{S}_3$ and $\mathrm{S}_4$. As we can see from the figure, $w - 1$ bits of the modulus are fixed to be all 0's or all 1's, while the other $n - w + 1$ bits are arbitrarily chosen. To fulfill the condition $\gcd(M, r) = 1$ (see Alg. 4), the least significant bit of $M$ is set to 1.

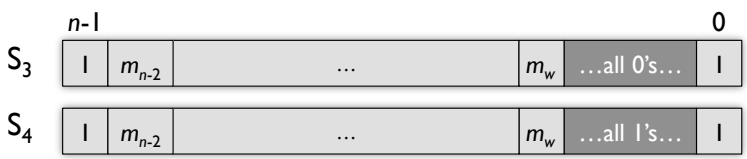


Figure 2.3: Binary representation of the proposed sets $\mathrm{S}_3$ and $\mathrm{S}_4$.

Due to the use of special type of moduli, the evaluation of the intermediate Montgomery quotient is simplified compared to the original algorithm given in Alg. 4. As in our case the value of $M'$ is simply equal to 1 or $-1$, the Montgomery quotient $q_M = (C \bmod r)M' \bmod r$ becomes now

$$q_M = \begin{cases} -C \bmod r & \text{if } M \in \mathrm{S}_3 \ ; \\ C \bmod r & \text{if } M \in \mathrm{S}_4 \ . \end{cases}$$

Since $r = 2^w$, the calculation of $q_M$ basically comes for free.

*Proof of Alg. 6.* Follows immediately from Lemma 2.3 and Lemma 2.4. □

**Algorithm 6** Proposed interleaved modular multiplication based on Montgomery reduction.

INPUT: $A = (A_{n_w-1} \dots A_0)_r$, $B = (B_{n_w-1} \dots B_0)_r$, $M \in \mathrm{S}_3 \cup \mathrm{S}_4$ where $0 \leq A, B < M$, $r = 2^w$ and $n_w = \lceil n/w \rceil$.
OUTPUT: $C = ABr^{-n_w} \bmod M$.

```
C ⇐ 0
for i = 0 to n_w − 1 do
    C ⇐ C + AB_i
    q_M = { −C mod r   if M ∈ S_3
          {  C mod r   if M ∈ S_4
    C ⇐ (C + q_M M)/r
end for
if C ≥ M then
    C ⇐ C − M
end if
return C.
```

### 2.4.5 Speeding Up Bipartite Modular Multiplication

## Publication Data

M. Knežević, F. Vercauteren, and I. Verbauwhede, "Speeding Up Bipartite Modular Multiplication," in *Arithmetic of Finite Fields, Third International Workshop – WAIFI 2010*, vol. 6087 of *Lecture Notes in Computer Science*, pp. 166–179, Springer, 2010.

#### Personal Contributions

- Principal author.

Our novel contribution consists of proposing a new set of moduli for which the bipartite modular multiplication performs faster. We also analyze the proposed set from a security point of view and propose a novel architecture of the modular multiplier that efficiently handles the proposed algorithm.

An algorithm that efficiently combines classical and Montgomery multiplication, in finite fields of characteristic 2, was independently proposed by Potgieter [136] and Wu [176] in 2002. Published in 2005, a bipartite modular multiplication (BMM) by Kaihara and Takagi [83] extended this approach to the ring of integers. The method efficiently combines classical modular multiplication with Montgomery's multiplication algorithm. It splits the operand multiplier into two parts that can be processed separately in parallel, increasing the calculation speed. The calculation is

performed using Montgomery residues defined by a modulus $M$ and a Montgomery radix $R$, $R < M$. Next, we outline the main idea of the BMM method.

Let the modulus $M$ be an $n_w$-digit integer, where the radix of each digit is $r = 2^w$ and let $R = r^k$ where $0 < k < n_w$. Consider the multiplier $B$ to be split into two parts $B_H$ and $B_L$ so that $B = B_H R + B_L$. Then, the Montgomery multiplication modulo $M$ of the integers $A$ and $B$ can be computed as follows:

$$\begin{aligned} A * B &= ABR^{-1} \bmod M \\ &= A(B_H R + B_L)R^{-1} \bmod M \\ &= \big((AB_H \bmod M) + (AB_L R^{-1} \bmod M)\big) \bmod M \ . \end{aligned}$$

The left term of the previous equation, $AB_H \bmod M$, can be calculated using the classical modular multiplication that processes the upper part of the split multiplier $B_H$. The right term, $AB_L R^{-1} \bmod M$, can be calculated using Montgomery multiplication that processes the lower part of the split multiplier $B_L$. Both calculations can be performed in parallel. Since the split operands $B_H$ and $B_L$ are shorter in length than $B$, the calculations $AB_H \bmod M$ and $AB_L R^{-1} \bmod M$ are performed faster than $ABR^{-1} \bmod M$.

Similar to the previous section, we propose a large set of moduli, for which the speed of bipartite modular multiplication, where the Barrett and Montgomery algorithms are the main ingredients, significantly increases. As will be shown later, we consider state of the art attacks on public-key cryptosystems, and show that the proposed set is safe to use in practice for both ECC/HECC and RSA cryptosystems. We propose a hardware architecture for the modular multiplier that outperforms the multiplier based on the standard BMM method.

Since the BMM method utilizes both Barrett and Montgomery multiplication algorithms, one needs to precompute both $\mu = \lfloor 2^{n+\alpha}/M \rfloor$ and $M' = -M_0^{-1} \bmod r$. Let us, for now, assume that the precomputed values are both of type $2^\gamma$ where $\gamma \in \mathbb{Z}$. By tuning $\mu$ and $M'$ to be of this special type, we transform a single-precision multiplication with these values into a simple shift operation in hardware. Therefore, we find a set of moduli for which the precomputed values are both of type $2^\gamma$. A lemma that defines this set is given below:

**Lemma 2.5.** *Let $M = 2^n - \Delta 2^w - 1$ be an $n$-bit positive integer in radix $r = 2^w$ representation with $\Delta \in \mathbb{Z}$, $w \in \mathbb{N}$ and $w < n$. Now, let $\mu = \lfloor 2^{n+\alpha}/M \rfloor$ where $\alpha \in \mathbb{N}$ and $M' = -M_0^{-1} \bmod r$. The following statement holds:*

$$\mu = 2^\alpha \ \wedge \ M' = 1 \quad \Rightarrow \quad 0 \leq \Delta \leq \left\lfloor \frac{2^n - 2^\alpha - 1}{2^w(2^\alpha + 1)} \right\rfloor \ .$$

*Proof.* To prove the lemma, we first rewrite $2^{n+\alpha}$ as $2^{n+\alpha} = M2^{\alpha} + \Delta 2^{w+\alpha} + 2^{\alpha}$. Now, the reciprocal $\mu$ of the modulus $M$ can be written as:

$$\mu = \left\lfloor \frac{2^{n+\alpha}}{M} \right\rfloor = 2^{\alpha} + \left\lfloor \frac{\Delta 2^{w+\alpha} + 2^{\alpha}}{M} \right\rfloor = 2^{\alpha} + \left\lfloor \frac{\theta}{M} \right\rfloor \ .$$

Having that $\mu = 2^{\alpha}$, the inequality $0 \leq \theta < M$ must hold. By solving the left part of inequality ($\theta \geq 0$) we get:

$$\Delta \geq -2^{-w} \ . \tag{2.7}$$

Similar, for the right part of inequality ($\theta < M$) we get:

$$\Delta < \frac{2^n - 2^{\alpha} - 1}{2^w\left(2^{\alpha} + 1\right)} \ . \tag{2.8}$$

From the condition $M' = -M_0^{-1} \bmod r = 1$ it follows that $M = -1 \bmod r$. This is true for all $\Delta \in \mathbb{Z}$. Finally, a condition that the modulus $M$ is an $n$-bit integer ($2^{n-1} \leq M < 2^n$) makes the last condition for $\Delta$:

$$-2^{-w} < \Delta \leq 2^{n-w-1} - 2^{-w} \ . \tag{2.9}$$

Now, from the inequalities (2.7), (2.8), (2.9) and the fact that $\Delta \in \mathbb{Z}$, follows the final condition for $\Delta$:

$$0 \leq \Delta \leq \left\lfloor \frac{2^n - 2^{\alpha} - 1}{2^w(2^{\alpha} + 1)} \right\rfloor \ .$$

□

The previous theorem defines a set of moduli for which both conditions $\mu = 2^{\alpha}$ and $M' = 1$ are satisfied. As mentioned earlier, to minimize the number of correction steps in the Barrett algorithm, we choose $\alpha = w + 3$. Finally, the proposed set is defined as:

$$\mathrm{S}_5: \quad M = 2^n - \Delta_5 2^w - 1 \quad \text{where } 0 \leq \Delta_5 \leq \left\lfloor \frac{2^n - 2^{w+3} - 1}{2^w(2^{w+3} + 1)} \right\rfloor \ .$$

Figure 2.4 further illustrates the properties of the proposed set. As can be seen, the $w$ least significant bits and the $w + 3$ most significant bits are fixed to be all 1's while the other $n - 2w - 3$ bits can be randomly chosen.

The evaluation of the intermediate quotient for the Barrett algorithm, $\hat{q}$, now becomes equal to:

$$\hat{q} = \left\lfloor \frac{\left\lfloor \frac{C}{2^{n+\beta}} \right\rfloor \mu}{2^{\alpha-\beta}} \right\rfloor = \left\lfloor \frac{\left\lfloor \frac{C}{2^{n+\beta}} \right\rfloor 2^{\alpha}}{2^{\alpha-\beta}} \right\rfloor = \left\lfloor \left\lfloor \frac{C}{2^{n+\beta}} \right\rfloor 2^{\beta} \right\rfloor \ .$$

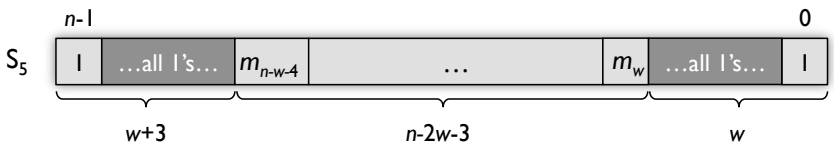


Figure 2.4: Binary representation of the proposed set $S_5$.

For $\beta \leq 0$, the previous equation simplifies and is equivalent to $\hat{q} = \left\lfloor \frac{C}{2^n} \right\rfloor$. Since $M' = 1$, the intermediate quotient for the Montgomery multiplication also gets simplified: $q_M = C \bmod r$.

Finally, the bipartite modular multiplication for the proposed set of moduli is given in Alg. 7. After the final addition is performed, one more correction step might be necessary since $0 \leq C_H + C_L < 2M$.

**Algorithm 7** BMM algorithm for the proposed set of moduli.

INPUT: $A = (A_{n_w-1} \dots A_0)_r$, $B = (B_{n_w-1} \dots B_0)_r = B_H r^k + B_L$, $M = (M_{n_w-1} \dots M_0)_r \in S_5$, where $0 \leq A, B < M$, $r = 2^w$, $0 < k < n_w$ and $n_w = \lceil n/w \rceil$.

**Output:** $C = ABr^{-k} \bmod M$**.**

1: $C_H \Leftarrow 0$
2: **for** $i = n_w - 1$ downto $k$ **do**
3: $\quad C_H \Leftarrow C_H r + AB_i$
4: $\quad \hat{q} \Leftarrow \lfloor C_H / 2^n \rfloor$
5: $\quad C_H \Leftarrow C_H - \hat{q}M$
6: **end for**
7: **if** $C_H \geq M$ **then**
8: $\quad C_H \Leftarrow C_H - M$
9: **end if**

1: $C_L \Leftarrow 0$
2: **for** $i = 0$ to $k - 1$ **do**
3: $\quad C_L \Leftarrow C_L + AB_i$
4: $\quad q_M \Leftarrow C_L \bmod r$
5: $\quad C_L \Leftarrow (C_L + q_M M)/r$
6: **end for**
7: **if** $C_L \geq M$ **then**
8: $\quad C_L \Leftarrow C_L - M$
9: **end if**

Return $C \Leftarrow C_H + C_L$.

### 2.4.6 Hardware Implementation of the Proposed Algorithms Based on Barrett and Montgomery Reduction Methods

To verify our approach in practice, we implement a set of multipliers that are based on our proposal and compare them with the multipliers that support the original Barrett and Montgomery algorithms. In order to have an objective comparison

between different designs, we define a relative throughput as

$$T_r = \frac{f_{\max}}{N} \ ,$$

where $f_{\max}$ is a maximum frequency and $N$ is a number of clock cycles. The total throughput is then obtained as $T = BT_r$, where $B$ is the number of bits processed in $1/T_r$ time.

To maximize the throughput, one obviously needs to decrease $N$ and increase $f_{\max}$. Typically, there are plenty of trade-offs to explore in order to make an optimal (in this case fastest) design. To make an objective comparison, we distinguish between designs that aim at the shortest critical path and the ones that achieve the minimum number of clock cycles. We address each of them separately, in the coming subsections.

### Optimization Goal: Shortest Critical Path

A modular multiplier with the shortest critical path (bold line) is depicted in Fig. 2.5 and consists of two multiple-precision multipliers ($\pi_1, \pi_2$). Apart from the multipliers, the architecture contains an additional adder denoted with $\Sigma$. Having two multiple-precision multipliers may seem redundant at first glance, but the multiplier $\pi_1$ uses data from $A$ and $B$ that are fixed during a single modular multiplication. Now, by running $\pi_1$ and $\pi_2$ in parallel, we speed-up the whole multiplication process. Both Barrett and Montgomery algorithms can be implemented based on this architecture. If the target is a more compact design, one can also use a single multiple-precision multiplier which does not reduce the generality of our discussion.

The critical path of the whole design occurs from the output of the register $C$ to the input of the temporary register in $\pi_2$, passing through one multiplexer, one single-precision multiplier and one adder (bold line).

### Optimization Goal: Minimum Number of Clock Cycles

In order to minimize the number of clock cycles needed for one modular multiplication, the architecture from Fig. 2.5 is modified as depicted in Fig. 2.6. Another single-precision multiplier ($\pi_3$), consisting only of pure combinational logic, is added without requiring any clock cycles for calculating its product.

Multipliers $\pi_1$ and $\pi_2$ perform multiplications at lines three and five of both Alg. 3 and Alg. 4, respectively. A multiplication performed in step four of both algorithms is done by multiplier $\pi_3$. An eventual shift of the register $C$ is handled by the controller. The exact schedule of the functional parts of the multiplier is as follows:

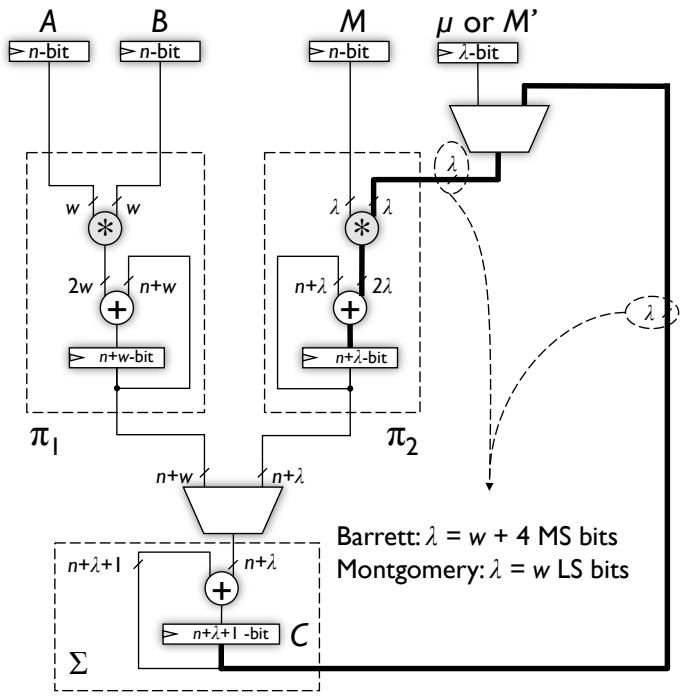


Figure 2.5: Datapath of the modular multiplier with the shortest critical path.

$\pi_1 \rightarrow \Sigma \rightarrow \pi_1\pi_2\pi_3 \rightarrow \Sigma \rightarrow \Sigma \rightarrow \pi_1\pi_2\pi_3 \rightarrow \Sigma \rightarrow \Sigma \rightarrow \ldots$ In case of generalized Barrett reduction, the precomputed value $\mu$ is $\lambda = w + 4$-bits long, while for the case of Montgomery the precomputed value $M'$ is $\lambda = w$-bits long. Due to the generalized Barrett algorithm, the multiplier $\pi_2$ uses the most significant $\lambda$ bits[2] of the product calculated by $\pi_3$, while for the case of Montgomery, it uses the least significant $\lambda$ bits of the same product. This is indeed a reason for Montgomery's multiplier being superior compared to the one of Barrett.

The critical path of the whole design occurs from the output of the register $C$ to the input of the temporary register in $\pi_2$, passing through two single-precision multipliers and one adder (bold line).

[2]As previously discussed, not all of the $\lambda = w + 4$ bits are necessary. Instead, one can use only $w + 3$ bits.

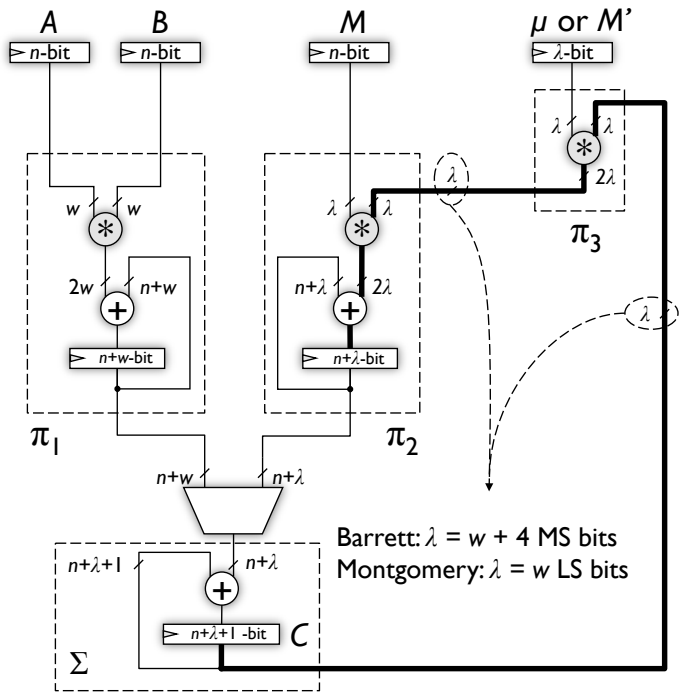


Figure 2.6: Datapath of the modular multiplier with the minimized number of clock cycles.

### Proposed Multiplier

A major improvement of the new algorithms is the simplified quotient evaluation. This fact results in the new proposed architecture for the efficient modular multiplier as shown in Fig. 2.7. It consists of two multiple-precision multipliers ($\pi_1$ and $\pi_2$) only. The most important difference is that there are no multiplications with the precomputed values and hence, the critical path contains one single-precision multiplier and one adder only (bold line). To compare the performance with the architectures proposed in Fig. 2.5 and Fig. 2.6, we have synthesized a number of multipliers.

### Results

To show this in practice, we have synthesized 192-bit, 512-bit and 1024-bit multipliers, each with digit size of 32 bits. The code was first written in GEZEL [154]

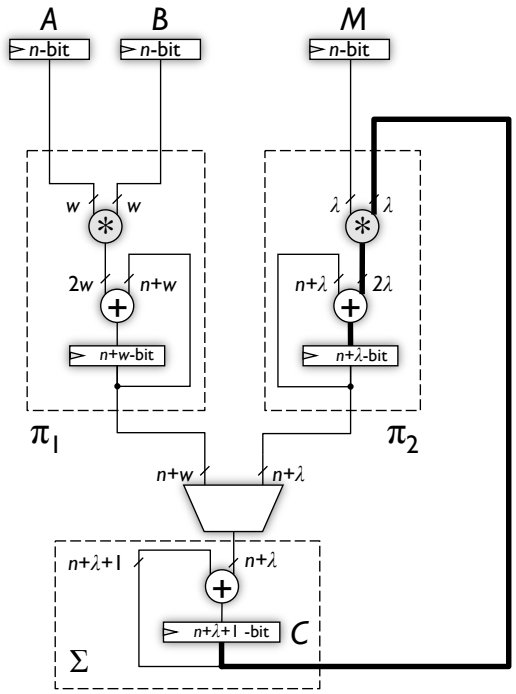


Figure 2.7: Datapath of our proposed multiplier.

and tested for its functionality and then translated to VHDL and synthesized using the Synposys Design Compiler version C-2009.06-SP3. We used a UMC 0.13 $\mu$m CMOS High-Speed standard cell library and the results can be found in Table 2.2. The size of the designs is given as the number of NAND gate equivalences (GE).

The designs that are based on Barrett and Montgomery algorithms, with the minimum number of clock cycles are outperformed up to 62 % and 17 %, respectively. The same architecture outperforms the modular multiplier based on standard Barrett reduction with the shortest critical path up to 11 %. The architecture with the shortest critical path based on Montgomery reduction is outperformed up to 12 %.

Additionally, designs based on our algorithms demonstrate area savings in comparison to the standard algorithms. Note that the obtained results are based on the synthesis only. After the place and route is performed we expect a decrease of the performance for all implemented multipliers and hence we believe that the relative speed-up will approximately remain the same.

Table 2.2: Hardware architectures of 192-bit, 512-bit and 1024-bit modular multipliers with digit size of $w = 32$ bits (Synopsys Design Compiler version C-2009.06-SP3, synthesis results).

| Algorithm | Design | $n$ [bit] | Area [kGE] | Frequency [MHz] | $N$ [cycles] | $T_r$ [kHz] |
|---|---|---|---|---|---|---|
| Classical | Fig. 2.5 | 192 | 42.57 | 303 | 61 | 4.967 |
| | | 512 | 63.21 | 280 | 321 | 0.872 |
| | | 1024 | 99.46 | 270 | 1,153 | 0.234 |
| | Fig. 2.6 | 192 | 46.14 | 187 | 55 | 3.400 |
| | | 512 | 65.65 | 182 | 305 | 0.597 |
| | | 1024 | 98.89 | 178 | 1,121 | 0.159 |
| | Fig 2.7 | 192 | 42.08 | 304 | 55 | 5.527 |
| | | 512 | 62.52 | 280 | 305 | 0.918 |
| | | 1024 | 90.39 | 274 | 1,121 | 0.244 |
| Montgomery | Fig. 2.5 | 192 | 34.06 | 305 | 61 | 5.000 |
| | | 512 | 59.67 | 287 | 321 | 0.894 |
| | | 1024 | 92.37 | 267 | 1,153 | 0.232 |
| | Fig. 2.6 | 192 | 40.32 | 262 | 55 | 4.764 |
| | | 512 | 55.49 | 254 | 305 | 0.833 |
| | | 1024 | 85.19 | 255 | 1,121 | 0.227 |
| | Fig. 2.7 | 192 | 37.30 | 308 | 55 | 5.600 |
| | | 512 | 59.07 | 292 | 305 | 0.957 |
| | | 1024 | 90.21 | 282 | 1,121 | 0.252 |

Finally, it is interesting to consider the choice of the digit size. As will be discussed in Section 2.4.8, the upper bound of the digit size is decided by security margins. A typical digit size of 8, 16, 32, 64 or even 128 bits seems to provide a reasonable security margin for an RSA modulus of 1024 bits or more. On the other side, with the increase of digit size, the number of cycles decreases for the whole design and the overall speed-up is increasing. It is also obvious that a larger digit size implies a larger circuit and thus the performance trade-off concerning throughput and area would be interesting to explore.

### 2.4.7 Hardware Implementation of the Proposed Algorithm Based on Bipartite Modular Multiplication

Obviously, the goal of the BMM algorithm is to utilize parallel computation and hence, increase the speed of modular multiplication. Therefore, in order to compare

different designs with the same input size, we again use the relative throughput as a measure for comparison.

Similar to the previous subsection, we again distinguish between designs that aim at the shortest critical path and the ones that achieve the minimum number of clock cycles. We address each of them separately, in the next subsections.

### Optimization Goal: Shortest Critical Path

A modular multiplier based on the BMM algorithm, depicted in Fig. 2.8, consists of four multiple-precision multipliers ($\pi_{H1}, \pi_{H2}, \pi_{L1}, \pi_{L2}$). Apart from the multipliers, the architecture contains some additional adders ($\Sigma_L$, $\Sigma_H$ and $\Sigma$). The multiple-precision multipliers are implemented in a digit-serial manner which typically provides a good trade-off between area and speed. The multipliers $\pi_{H1}$ and $\pi_{H2}$ assemble together the Barrett modular multiplier that processes the most significant half of $B$ (that is $B_H$). Similarly, the multipliers $\pi_{L1}$ and $\pi_{L2}$ form the Montgomery multiplier that processes the least significant half of $B$ (that is $B_L$). The results of both multipliers are finally added together, resulting in $C = ABr^{-k} \bmod M$. The parameters $k$ and $\alpha$ are chosen such that the execution speed is maximized and the number of correction steps is minimized: $k = \lfloor n_w/2 \rfloor$ and $\alpha = w + 3$.

A choice of the specific architecture is based on the following criteria. The two levels of parallelism are exploited such that the number of clock cycles needed for one modular multiplication is minimized. First, the BMM algorithm itself is constructed such that the Barrett part and the Montgomery part of the multiplier work independently, in parallel. Second, the multiple-precision multipliers $\pi_{H1}$ and $\pi_{H2}$ in the Barrett part, and $\pi_{L1}$ and $\pi_{L2}$ in the Montgomery part operate with independent data such that they run in parallel and speed-up the whole multiplication process. The critical path is minimized and consists of one multiplexer, a single-precision multiplier and an adder (bold line, Fig. 2.8).

In order to avoid any ambiguity we provide a graph in Fig. 2.9 which shows the exact timing schedule of separate blocks inside the multiplier. With $i$ ($0 \leq i < k$) we denote the current iteration of the algorithm. Each iteration consists of $n_w + 3$ clock cycles except the first iteration that lasts for $n_w + 1$ cycles.

### Optimization Goal: Minimum Number of Clock Cycles

In order to minimize the number of clock cycles needed for one modular multiplication, the architecture from Fig. 2.8 is modified as depicted in Fig. 2.10. Two single-precision multipliers ($\pi_{H3}$ and $\pi_{L3}$), consisting only of pure combinational logic, are added without requiring any clock cycles for calculating their products.

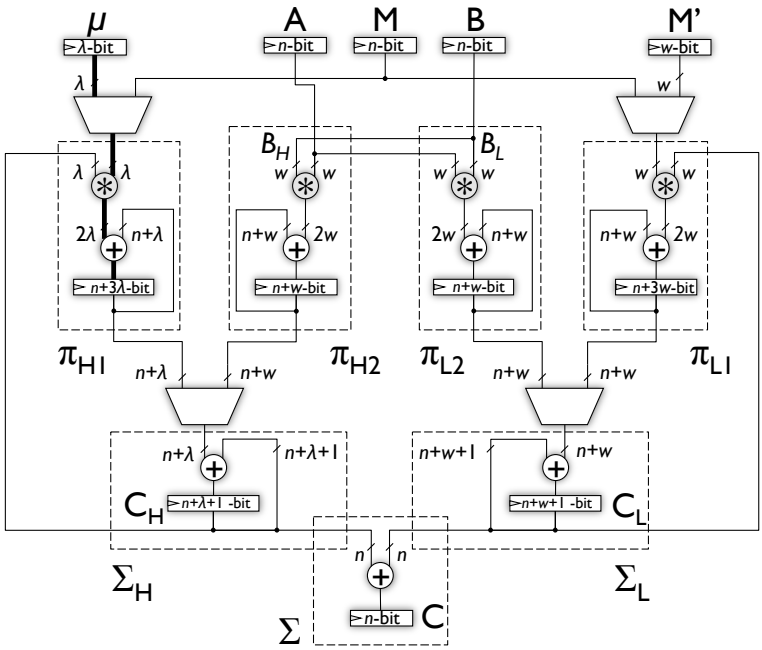


Figure 2.8: Datapath of the modular multiplier with the shortest critical path based on the BMM method.

We again provide a graph in Fig. 2.11 which shows the timing schedule of the multiplier. Each iteration now consists of $n_w + 2$ clock cycles except the first that lasts for $n_w + 1$ cycles.

The critical path of the whole design occurs from the output of the register $Z_H$ to the input of the temporary register in $\pi_{H1}$, passing through two single-precision multipliers and one adder (bold line).

### Proposed Multiplier

An architecture of the modular multiplier based on the BMM method with the moduli from the proposed set (see Alg. 7) is shown in Fig. 2.12. The most important difference is that there are no multiplications with the precomputed values and hence, the critical path contains one single-precision multiplier and one adder only. A full timing schedule of the multiplier is given in Fig. 2.13. The number of cycles remains the same as in the architecture from Fig. 2.10 while the critical path reduces.

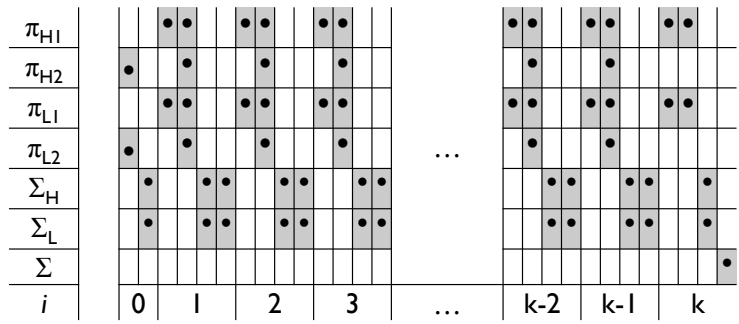


Figure 2.9: Timing schedule of the BMM multiplier with the shortest critical path.

#### Results

To show this in practice, we have synthesized 192-bit, 512-bit and 1024-bit multipliers, each with the digit size of 16, 32 and 64 bits. The designs were synthesized using UMC 0.13 $\mu$m CMOS High-Speed standard cell library with Synopsys Design Compiler version C-2009.06-SP3. The results are given in Table 2.3.

Observing the implementation results, we conclude that our proposed design outperforms the standard BMM design with the shortest critical path by 17 %. A design that is based on standard BMM with the minimum number of clock cycles is outperformed by at most 68 %. Furthermore, our design consumes less area than all its counterparts.

### 2.4.8 Security Considerations

In this section we analyze the security implications of choosing primes in one of the sets $S_1 \ldots S_5$ for use in ECC/HECC and in RSA.

In the current state of the art, the security of ECC/HECC over finite fields GF($p$) only depends on the extension degree of the field (see 'Handbook of Elliptic and Hyperelliptic Curve Cryptography' by Avanzi et al. [12]). Therefore, the security does not depend on the precise structure of the prime $p$. This is illustrated by the particular choices for $p$ that have been made in several standards such as SEC [158], NIST [129], ANSI [10]. In particular, the following primes have been proposed: $p_{192} = 2^{192} - 2^{64} - 1$, $p_{224} = 2^{224} - 2^{96} + 1$, $p_{256} = 2^{256} - 2^{224} + 2^{192} + 2^{96} - 1$, $p_{384} = 2^{384} - 2^{128} - 2^{96} + 2^{32} - 1$, and $p_{521} = 2^{521} - 1$. It is easy to verify that for $w \leq 28$ all primes are in one of the proposed sets. As such at least one of our

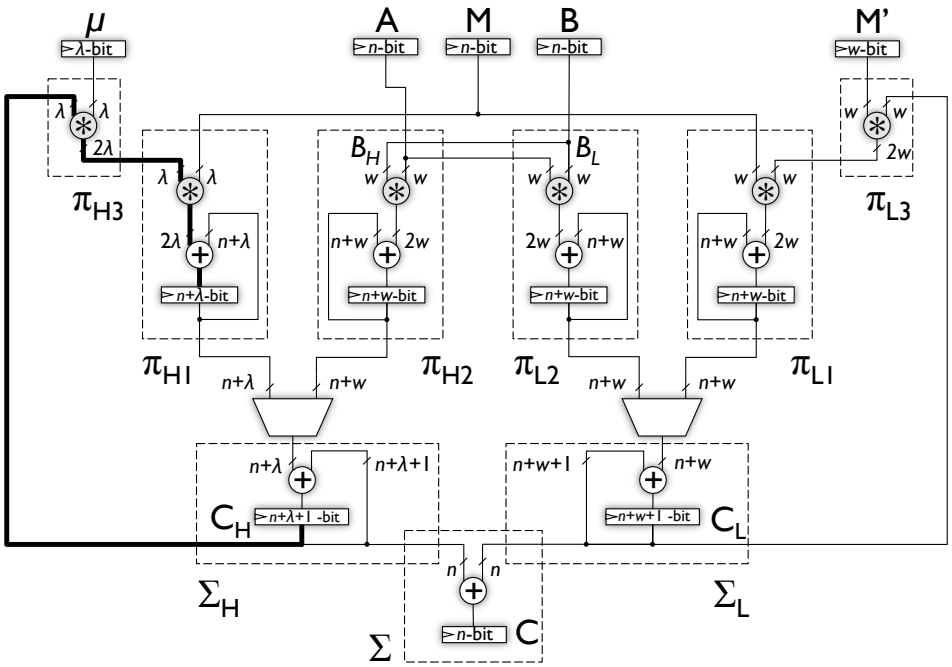


Figure 2.10: Datapath of the modular multiplier with the minimized number of clock cycles based on the BMM method.

methods applies for all primes included in the standards. In conclusion: choosing a prime of prescribed structure has no influence on the security of ECC/HECC.

The case of RSA requires a more detailed analysis than ECC/HECC. First, we assume that the modulus $N$ is chosen from one of the proposed sets. This is a special case of the security analysis given by Lenstra in [107] followed by the conclusion that the resulting moduli do not seem to offer less security than regular RSA moduli.

Next, we assume that the primes $p$ and $q$, which constitute the modulus $N = pq$, are both chosen in one of the sets $S_i$. To analyze the security implications of the restricted choice of $p$ and $q$, we first make a trivial observation. The number of $n$-bit primes in the sets $\mathrm{S}_1 \ldots \mathrm{S}_4$ for $n > 259 + w$ ($n > 259 + 2w$ for $\mathrm{S}_5$) is large enough such that exhaustive listing of these sets is impossible, since a maximum of $w + 3$ ($2w + 3$ for $\mathrm{S}_5$) bits are fixed.

The security analysis then corresponds to attacks on RSA with partially known

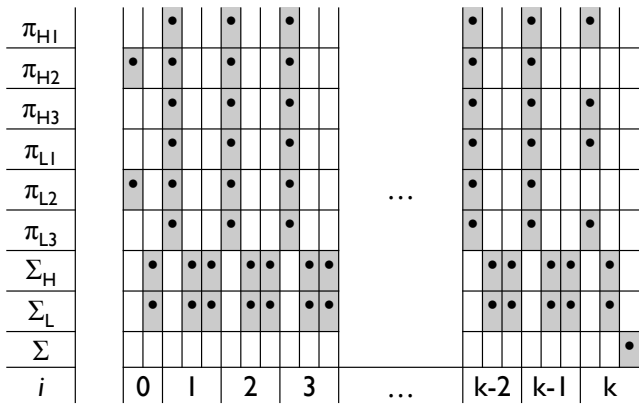


Figure 2.11: Timing schedule of the BMM multiplier with the minimized number of clock cycles.

factorization. This problem has been analyzed extensively in the literature and the first results come from Rivest and Shamir [143] in 1985. They describe an algorithm that factors $N$ in polynomial time if 2/3 of the bits of $p$ or $q$ are known. In 1995, Coppersmith [34] improved this bound to 3/5.

Today's best attacks all rely on variants of Coppersmith's algorithm published in 1996 [36, 35]. A good overview of these algorithms is given by May in [112, 113]. The best results in this area are as follows. Let $N$ be an $n$ bit number, which is a product of two $n/2$-bit primes. If half of the bits of either $p$ or $q$ (or both) are known, then $N$ can be factored in polynomial time. If less than half of the bits are known, say $n/4 - \varepsilon$ bits, then the best algorithm simply guesses $\varepsilon$ bits and then applies the polynomial time algorithm, leading to a running time exponential in $\varepsilon$. In practice, the values of $w$ (typically $w \leq 128$) and $n$ ($n \geq 1024$) are always such that our proposed moduli remain secure against Coppersmith's factorization algorithm, since at most $w + 3$ ($2w + 3$ for $S_5$) bits of $p$ and $q$ are known.

Finally, we consider a similar approach extended to the multi-prime RSA, a special fast RSA-type of cryptosystem introduced by Takagi [161]. Specifically, we consider moduli of the form $N = p^r q$ where $p$ and $q$ have the same bit-size. This attack is an extension of Coppersmith's work and was proposed by Boneh, Durfee and Howgrave-Graham [24]. Assuming that $p$ and $q$ are of the same bit-size, one needs a $1/(r+1)$-fraction of the most significant bits of $p$ in order to factor $N$ in polynomial time. In other words, for the case $r = 1$, we need half of the bits, whereas for, e.g. $r = 2$ we need only a third of the most significant bits of $p$. These results show that the primes $p, q \in S$, assembling an RSA modulus of the form $N = p^r q$, should be used with care. This is especially true when $r$ is large. Note that if $r \approx \log p$,

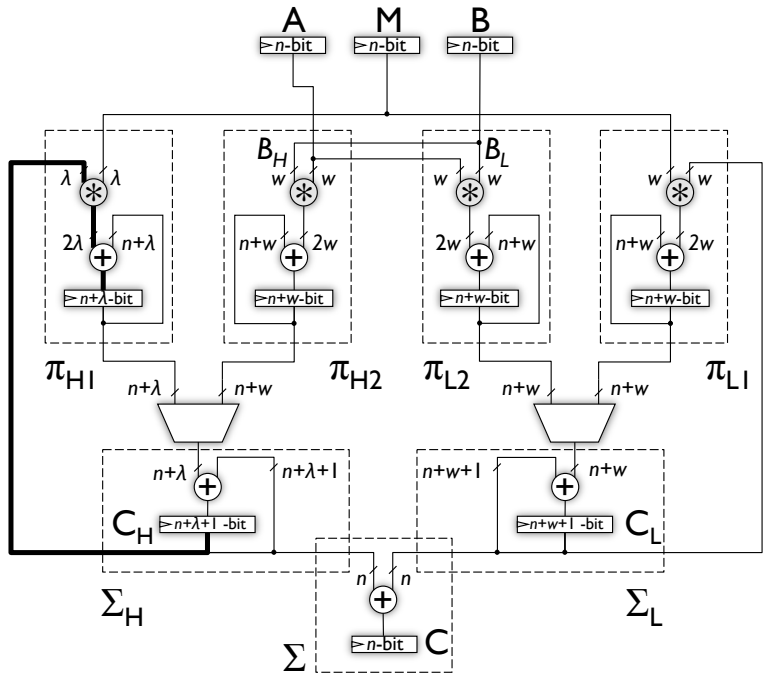


Figure 2.12: Datapath of the modular multiplier based on the BMM method with a modulus from the proposed set.

the latter factoring method factors $N$ in polynomial time for any primes $p, q \in \mathbb{N}$.

### 2.4.9 The Proposed Multiplication Methods in GF($2^n$)

Following the same principles described in the previous sections, we provide a special set of moduli for which the digit-serial multiplication in GF($2^n$) based on Barrett reduction has no precomputation and has a simplified quotient evaluation. Second, we show how the interleaved digit-serial multiplication based on Montgomery reduction for a complementary set of moduli, can also be performed without precomputation and with simplified quotient evaluation. As these algorithms operate in a binary field, they are specially suitable for efficient hardware implementations.

To make the following discussion easier we introduce the floor function for polynomials in the following manner. Let $M(x)$ and $U(x)$ be polynomials over GF(2) with $\deg(M(x)) = n$ and $\deg(U(x)) > n$, then there exist unique polynomials $q(x)$

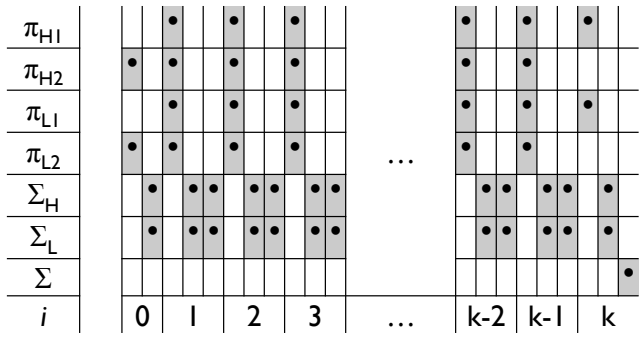


Figure 2.13: Timing schedule of the proposed BMM multiplier.

and $Z(x)$ over GF(2) where $\deg(Z(x)) \leq n-1$, such that $U(x) = q(x)M(x) + Z(x)$. The polynomial $q(x)$ is called the quotient and is denoted by the floor function as

$$q(x) = \left\lfloor U(x)/M(x) \right\rfloor = U(x) \operatorname{div} M(x) \ . \tag{2.10}$$

The following lemma is the analogue of Lemma 2.2.

**Lemma 2.6.** *Let $M(x) = x^n + \Delta(x)$ be an irreducible polynomial over* GF(2) *such that $\Delta(x) = \sum_{i=0}^{n-w} m_i x^i$ where $1 < w < n$, $m_i \in GF(2)$ and let $\mu(x) = \left\lfloor x^{n+w-1}/M(x) \right\rfloor$. Then it holds $\mu(x) = x^{w-1}$.*

*Proof.* Rewrite $x^{n+w-1}$ as $x^{n+w-1} = x^{w-1}M(x) + x^{w-1}\Delta(x)$. Since $\deg(x^{w-1}\Delta(x)) \leq n-1$ and $\deg(M(x)) = n$, we conclude that the quotient is indeed $\mu(x) = x^{w-1}$. □

Based on Lemma 2.6, we define a set of irreducible polynomials for which the digit-serial multiplication in GF($2^n$) based on Barrett reduction has no precomputation and has a simplified quotient evaluation. The set is illustrated in Fig. 2.14

$$\mathrm{F}_1: \quad M(x) = x^n + \Delta_1(x) \quad \text{where } \Delta_1(x) = \sum_{i=0}^{n-w} m_i x^i \ . \tag{2.11}$$

In [43], Dhem presents a digit-serial multiplication in GF($2^n$) based on Barrett modular reduction. Based on it, we propose a digit-serial multiplication algorithm for the proposed set in Alg. 8.

Table 2.3: Hardware architectures of 192-bit, 512-bit and 1024-bit modular multipliers (Synopsys Design Compiler version C-2009.06-SP3, synthesis results).

| Design | $n$ [bit] | $w$ [bit] | Area [kGE] | Frequency [MHz] | $N$ [cycles] | $T_r$ [MHz] |
|---|---|---|---|---|---|---|
| Fig. 2.8 | 192 | 16 | 48.20 | 340 | 103 | 3.301 |
| | | 32 | 85.66 | 224 | 34 | 6.588 |
| | | 64 | 212.40 | 137 | 16 | 8.563 |
| | 512 | 16 | 96.31 | 315 | 593 | 0.531 |
| | | 32 | 134.10 | 209 | 169 | 1.237 |
| | | 64 | 259.84 | 134 | 53 | 2.528 |
| | 1024 | 16 | 177.93 | 300 | 2,209 | 0.136 |
| | | 32 | 208.59 | 193 | 593 | 0.325 |
| | | 64 | 356.37 | 134 | 169 | 0.793 |
| Fig. 2.10 | 192 | 16 | 50.17 | 230 | 97 | 2.371 |
| | | 32 | 84.25 | 147 | 31 | 4.742 |
| | | 64 | 220.73 | 82 | 14 | 5.857 |
| | 512 | 16 | 97.44 | 234 | 577 | 0.406 |
| | | 32 | 127.33 | 144 | 161 | 0.894 |
| | | 64 | 271.54 | 80 | 49 | 1.633 |
| | 1024 | 16 | 169.49 | 223 | 2,177 | 0.102 |
| | | 32 | 198.01 | 145 | 577 | 0.251 |
| | | 64 | 341.59 | 82 | 161 | 0.509 |
| Fig. 2.12 | 192 | 16 | 44.65 | 343 | 97 | 3.536 |
| | | 32 | 83.14 | 240 | 31 | 7.742 |
| | | 64 | 204.73 | 138 | 14 | 9.857 |
| | 512 | 16 | 95.23 | 323 | 577 | 0.560 |
| | | 32 | 137.41 | 229 | 161 | 1.422 |
| | | 64 | 247.35 | 134 | 49 | 2.735 |
| | 1024 | 16 | 183.01 | 316 | 2,177 | 0.145 |
| | | 32 | 211.07 | 212 | 577 | 0.367 |
| | | 64 | 346.40 | 134 | 161 | 0.832 |

*Proof of Alg. 8.* The correctness of the algorithm follows directly from Lemma 2.6 since $\mu = x^{w-1}$ and the quotient evaluation thus becomes

$$\hat{q}(x) = \left\lfloor \frac{\left\lfloor \frac{C(x)}{x^{n-1}} \right\rfloor \mu(x)}{x^{w-1}} \right\rfloor = \left\lfloor \frac{\left\lfloor \frac{C(x)}{x^{n-1}} \right\rfloor x^{w-1}}{x^{w-1}} \right\rfloor = \left\lfloor \frac{C(x)}{x^{n-1}} \right\rfloor \ .$$

□


Figure 2.14: Binary representation of the proposed set $\mathrm{F}_1$.

**Algorithm 8** Proposed digit-serial multiplication in GF($2^n$) based on Barrett reduction.

INPUT: $A(x) = \sum_{i=0}^{n_w-1} A_i(x)x^{iw}$, $B(x) = \sum_{i=0}^{n_w-1} B_i(x)x^{iw}$, $M(x) \in \mathrm{F}_1$, $n_w = \lceil n/w \rceil$.
OUTPUT: $C(x) = A(x)B(x) \bmod M(x)$.
  $C(x) \Leftarrow 0$
  **for** $i = n_w - 1$ downto 0 **do**
    $C(x) \Leftarrow C(x)x^w + A(x)B_i(x)$
    $\hat{q}(x) \Leftarrow \left\lfloor \frac{C(x)}{x^{n-1}} \right\rfloor$
    $C(x) \Leftarrow C(x) + \hat{q}(x)M(x)$
  **end for**
  return $C(x)$.

Similar to Lemma 2.6 we also give Lemma 2.7 that is at the heart of the proposed multiplication in GF($2^n$) based on Montgomery reduction.

**Lemma 2.7.** *Let* $M(x) = x^n + x^w\Delta(x) + 1$ *be an irreducible polynomial over* GF(2) *such that* $\Delta(x) = \sum_{i=0}^{n-w-1} m_i x^i$ *where* $1 < w < n$, $m_i \in GF(2)$ *and let* $M'(x) = M(x)^{-1} \bmod x^w$. *Then it holds* $M'(x) = 1$

*Proof.* Note that $M(x) \equiv 1 \bmod x^w$, which shows immediately that $M'(x) = 1$. □

We now define a set of irreducible polynomials for which the digit-serial multiplication in GF($2^n$) based on Montgomery reduction has no precomputation and has a simplified quotient evaluation. Figure 2.15 illustrates its binary representation.

$$\mathrm{F}_2: \quad M(x) = x^n + x^w\Delta_2(x) + 1 \quad \text{where } \Delta_2(x) = \sum_{i=0}^{n-w-1} m_i x^i \ . \tag{2.12}$$

The Montgomery multiplication algorithm for finite fields of characteristic 2 was proposed by Koç and Acar in [31]. Based on this algorithm we propose a digit-serial multiplication in GF($2^n$) which skips the precomputation and has a simplified quotient evaluation (see Alg. 9).

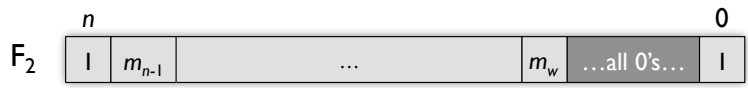


Figure 2.15: Binary representation of the proposed set $\mathrm{F}_2$.

**Algorithm 9** Proposed digit-serial multiplication in GF($2^n$) based on Montgomery reduction.

INPUT: $A(x) = \sum_{i=0}^{n_w-1} A_i(x)x^{iw}$, $B(x) = \sum_{i=0}^{n_w-1} B_i(x)x^{iw}$, $M(x) \in \mathrm{F}_2$, $r(x) = x^w$, $n_w = \lceil n/w \rceil$.
OUTPUT: $C(x) = A(x)B(x)r(x)^{-n_w} \bmod M(x)$.
  $C(x) \Leftarrow 0$
  **for** $i = 0$ to $n_w - 1$ **do**
    $C(x) \Leftarrow C(x) + A_i(x)B(x)$
    $q(x) \Leftarrow C(x) \bmod r(x)$
    $C(x) \Leftarrow (C(x) + M(x)q(x))/r(x)$
  **end for**
  return $C(x)$.

*Proof of Alg. 9.* The correctness follows immediately from Lemma 2.7 since $M'(x) = 1$. □

### 2.4.10 Summary

In this section we proposed two interleaved modular multiplication algorithms based on Barrett and Montgomery reductions. We introduced two sets of moduli for the algorithm based on Barrett's and two sets of moduli for the algorithm based on Montgomery's algorithm. Another set of moduli for which the performance of bipartite modular multiplication considerably increases is also proposed. These sets contain moduli with a prescribed number (typically the digit-size) of zero/one bits, either in the most significant or least significant part. Due to this choice, our algorithms have no precomputational phase and have a simplified quotient evaluation, which makes them more efficient than existing solutions. Since the security level of ECC/HECC does not depend at all on the precise structure of the prime $p$, our proposed set is safe to be used for constructing underlying fields in elliptic curves cryptography. The case of RSA is also discussed and if used with care ($w \leq 128$ and $n \geq 1024$) our proposed set does not decrease the security of RSA.

Moreover, Lenstra [107] and Joye [81] have already shown that the generation of RSA moduli with a predetermined portion can be as efficient as a regular generation

of RSA moduli. Generating primes with a predetermined portion is therefore trivial and is straightforward for implementation.

Furthermore, following a similar approach, we defined two sets of moduli for finite fields of characteristic 2 for which the modular multiplication over GF($2^n$) becomes faster and simplified.

## 2.5 Bit-Parallel Modular Multiplication Based on Barrett and Montgomery Reduction Methods Without Precomputation

### Publication Data

M. Knežević, L. Batina, and I. Verbauwhede, "Modular Reduction Without Precomputational Phase," in *IEEE International Symposium on Circuits and Systems – ISCAS 2009*, pp. 1389–1392, IEEE, 2009.

M. Knežević, K. Sakiyama, J. Fan, and I. Verbauwhede, "Modular Reduction in GF($2^n$) Without Precomputational Phase," in *Arithmetic of Finite Fields, Second International Workshop – WAIFI 2008*, vol. 5130 of *Lecture Notes in Computer Science*, pp. 77–87, Springer, 2008.

#### Personal Contributions

- Principal author.

Our novel contribution consists of proposing several sets of moduli for which the precomputational phase in Barrett and Montgomery reduction algorithms can be avoided. We discuss the case of integers and finite fields of characteristic 2.

In the previous section, we discussed in detail the approach of accelerating digit-serial multiplication algorithms. As outlined in Section 2.3, the bit-parallel algorithms represent another important aspect of implementing efficient modular multiplication. These algorithms normally use significantly more area and provide much faster multiplication as a result. In this section, we deal with the bit-parallel algorithms and propose the sets of integers and the sets of polynomials for which the precomputational stage in both Barrett and Montgomery algorithms can be omitted.

In Section 2.3.1, we have outlined the bit-parallel modular multiplication algorithms that are based on Barrett and Montgomery reduction. In this section, we first

provide two special sets of moduli for which the precomputational step in Barrett multiplication can be avoided. Second, we propose a modular multiplication algorithm that is based on Barrett reduction and show how using a modulus from the defined sets can be beneficial for skipping the precomputational step. Then, we show how Montgomery multiplication, with using complementary sets of moduli, can also be performed very efficiently without precomputation. A modular multiplication based on these algorithms can be implemented at a very high throughput in hardware. Finally, by providing two additional sets of moduli, we extend the same approach to finite fields of characteristic 2.

Starting with the basic idea of the proposed modular multiplication based on Barrett's algorithm we give two lemmata as follows.

**Lemma 2.8.** *Let $M = r^{n_w-1} + \Delta$ be an $n_w$-digit positive integer in radix $r$ representation, such that $0 \leq \Delta < \lfloor r^{(n_w-1)/2} \rfloor$ and $\mu = \lfloor r^{2n_w-2}/M \rfloor$. Then it holds $\mu = r^{n_w-1} - \Delta$.*

*Proof.* Rewrite $r^{2n_w-2}$ as $r^{2n_w-2} = (r^{n_w-1} - \Delta)M + \Delta^2$. Since it is given that $0 \leq \Delta < \lfloor r^{(n_w-1)/2} \rfloor$, we conclude that $0 \leq \Delta^2 < M$. By definition of Euclidean division, this shows that $\mu = r^{n_w-1} - \Delta$. □

**Lemma 2.9.** *Let $M = r^{n_w} - \Delta$ be an $n_w$-digit positive integer in radix $r$ representation, such that $0 < \Delta < \lfloor r^{n_w/2} \rfloor$ and $\mu = \lfloor r^{2n_w}/M \rfloor$. Then it holds $\mu = r^{n_w} + \Delta$.*

*Proof.* Rewrite $r^{2n_w}$ as $r^{2n_w} = (r^{n_w} + \Delta)M + \Delta^2$. Since it is given that $0 < \Delta < \lfloor r^{n_w/2} \rfloor$, we conclude that $0 < \Delta^2 < M$. By definition of Euclidean division, this shows that $\mu = r^{n_w} + \Delta$. □

The bit-parallel classical modular multiplication based on Barrett reduction is given in Section 2.3.1. Now, according to Lemmata 2.8 and 2.9, we can define two sets of primes for which the Barrett reduction can be performed without using a precomputational phase. These sets are of type:

$$\begin{array}{lll} \mathrm{S}_6: & M = r^{n_w-1} + \Delta_6 & \text{where } 0 \leq \Delta_6 < \lfloor r^{(n_w-1)/2} \rfloor \ ; \\ \mathrm{S}_7: & M = r^{n_w} - \Delta_7 & \text{where } 0 < \Delta_7 < \lfloor r^{n_w/2} \rfloor \ . \end{array} \tag{2.13}$$

To further illustrate the properties of the two proposed sets, we give Fig. 2.16 where the moduli from each set are represented in radix 2 representation. Note that here $k = \lfloor \frac{n-1}{2} \rfloor$, $m_i \in \{0, 1\}$ and, additionally, Eq. (2.13) has to be satisfied.

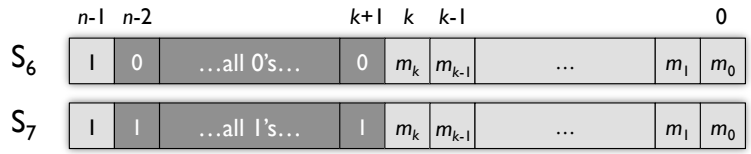


Figure 2.16: Binary representation of the proposed sets $S_6$ and $S_7$.

The precomputed reciprocal needed for Barrett's algorithm can be easily formed together with the modulus as:

$$\mu = \begin{cases} r^{n_w-1} - \Delta_6 & \text{if } M \in S_6 \ ; \\ r^{n_w} + \Delta_7 & \text{if } M \in S_7 \ . \end{cases}$$

The proposed modular multiplication based on Barrett algorithm is shown in Alg. 10.

**Algorithm 10** Modular multiplication without precomputation based on Barrett algorithm.

INPUT: $A = (A_{n_w-1} \dots A_0)_r$, $B = (B_{n_w-1} \dots B_0)_r$, $M \in S_6 \cup S_7$, $\mu$, $r \geq 3$, $0 \leq A, B < M$.
OUTPUT: $C = AB \bmod M$.

$C = AB$

$$\hat{q} \Leftarrow \begin{cases} \left\lfloor \frac{\lfloor \frac{C}{r^{n_w-1}} \rfloor \mu}{r^{n_w-1}} \right\rfloor & \text{if } M \in S_1 \\ \left\lfloor \frac{\lfloor \frac{C}{r^{n_w-1}} \rfloor \mu}{r^{n_w+1}} \right\rfloor & \text{if } M \in S_2 \end{cases}$$

$C \Leftarrow C \bmod r^{n_w+1} - M\hat{q} \bmod r^{n_w+1}$
**if** $C < 0$ **then**
  $C \Leftarrow C + r^{n_w+1}$
**end if**
**while** $C \geq M$ **do**
  $C \Leftarrow C - M$
**end while**
RETURN: $C$.

*Proof of Alg. 10.* To show the correctness of the algorithm we first assume that $M \in S_6$. The case $M \in S_7$ is completely analogous. Let $q = \lfloor C/M \rfloor$ and $r = C \bmod M = C - qM$. In the algorithm, $\hat{q}$ is an estimate of $q$ since

$$\frac{C}{M} = \frac{C}{r^{n_w-1}} \frac{r^{2n_w-2}}{M} \frac{1}{r^{n_w-1}} \ .$$

We now show that $q-3 \leq \hat{q} \leq q$. The right part of the inequality is straightforward to prove as it is

$$\hat{q} = \left\lfloor \frac{\left\lfloor \frac{C}{r^{n_w-1}} \right\rfloor \mu}{r^{n_w-1}} \right\rfloor \leq \left\lfloor \frac{C}{r^{n_w-1}} \frac{r^{2n_w-2}}{M} \frac{1}{r^{n_w-1}} \right\rfloor = \left\lfloor \frac{C}{M} \right\rfloor = q \ .$$

Next, we prove the left part of the inequality. Since $\frac{X}{Y} \geq \left\lfloor \frac{X}{Y} \right\rfloor > \frac{X}{Y} - 1$ for any $X, Y \in \mathbb{N}$, we can write the following inequality

$$\begin{aligned} q &= \left\lfloor \frac{\frac{C}{r^{n_w-1}} \frac{r^{2n_w-2}}{M}}{r^{n_w-1}} \right\rfloor \\ &\leq \left\lfloor \frac{\left(\left\lfloor \frac{C}{r^{n_w-1}} \right\rfloor + 1\right)\left(\left\lfloor \frac{r^{2n_w-2}}{M} \right\rfloor + 1\right)}{r^{n_w-1}} \right\rfloor \\ &= \left\lfloor \frac{\left\lfloor \frac{C}{r^{n_w-1}} \right\rfloor \mu}{r^{n_w-1}} + \frac{\left\lfloor \frac{C}{r^{n_w-1}} \right\rfloor + \left\lfloor \frac{r^{2n_w-2}}{M} \right\rfloor + 1}{r^{n_w-1}} \right\rfloor \ . \end{aligned}$$

Since $C < M^2$ and $M = r^{n_w-1} + \Delta_6 \geq r^{n_w-1}$, where $0 \leq \Delta_6 < \left\lfloor r^{(n_w-1)/2} \right\rfloor$, it follows that

$$\begin{aligned} \left\lfloor \frac{C}{r^{n_w-1}} \right\rfloor + \left\lfloor \frac{r^{2n_w-2}}{M} \right\rfloor + 1 &\leq \left\lfloor \frac{M^2}{r^{n_w-1}} \right\rfloor + r^{n_w-1} + 1 \\ &= r^{n_w-1} + 2\Delta_6 + \left\lfloor \frac{\Delta_6^2}{r^{n_w-1}} \right\rfloor + r^{n_w-1} + 1 \\ &\leq 2r^{n_w-1} + 2\Delta_6 + 2 \ . \end{aligned}$$

Finally, we have

$$\begin{aligned} q &\leq \left\lfloor \frac{\left\lfloor \frac{C}{r^{n_w-1}} \right\rfloor \mu}{r^{n_w+1}} + \frac{2r^{n_w-1} + 2\Delta_6 + 2}{r^{n_w-1}} \right\rfloor \\ &= \left\lfloor \frac{\left\lfloor \frac{C}{r^{n_w-1}} \right\rfloor \mu}{r^{n_w+1}} + 2 + \frac{2\Delta_6 + 2}{r^{n_w-1}} \right\rfloor \\ &\leq \hat{q} + 3 \ . \end{aligned}$$

Similarly, for the case when $M \in \mathrm{S}_7$ we have

$$\begin{aligned} q &= \left\lfloor \frac{\frac{C}{r^{n_w-1}} \frac{r^{2n_w}}{M}}{r^{n_w+1}} \right\rfloor \\ &\leq \left\lfloor \frac{\left(\left\lfloor \frac{C}{r^{n_w-1}} \right\rfloor + 1\right)\left(\left\lfloor \frac{r^{2n_w}}{M} \right\rfloor + 1\right)}{r^{n_w+1}} \right\rfloor \\ &\leq \left\lfloor \frac{\left\lfloor \frac{C}{r^{n_w-1}} \right\rfloor \mu}{r^{n_w+1}} + 2 + \frac{2\Delta_7 r + 2}{r^{n_w+1}} \right\rfloor \\ &\leq \hat{q} + 3 \ . \end{aligned}$$

Hence, $\hat{q}$ is indeed a good estimate of $q$ and at most 3 subtractions at the correction step are required to obtain $C = AB \bmod M$. This concludes the proof. □

In contrast to the original Barrett algorithm for integers, our proposed algorithm differs not only in the lack of the precomputational phase, but also in the number of correction steps. While in the original Barrett reduction algorithm, the number of correction steps is at most 2, in our modified reduction algorithm this number can be at most 3. One can further reduce the number of redundant subtractions by increasing the precision of $\mu$ by two or more digits. The same approach was applied by Dhem [42] to the original Barrett algorithm, resulting in the improved Barrett reduction where at most one subtraction needs to be performed at the correction step.

Similar to Lemmata 2.8 and 2.9 we also give Lemmata 2.10 and 2.11 that are the starting points for the proposed modular reduction based on Montgomery algorithm.

**Lemma 2.10.** *Let $M = \Delta r^k + 1$ be an $n_w$-digit positive integer in radix $r$ representation where $r^{n_w-k-1} \leq \Delta < r^{n_w-k}$, $k = \left\lceil \frac{n_w-1}{2} \right\rceil$ and let $M' = -M^{-1} \bmod r^{n_w-1}$. Then it holds $M' = \Delta r^k - 1$.*

*Proof.* In order to prove the lemma we need to show that $MM' = -1 \bmod r^{n_w-1}$. Indeed, if we express the product $MM'$ as

$$\begin{aligned} MM' &= (\Delta r^k + 1)(\Delta r^k - 1) \\ &= \Delta^2 r^{2k} - 1 \ , \end{aligned}$$

it becomes obvious that $MM' = -1 \bmod r^{n_w-1}$. □

**Lemma 2.11.** *Let $M = \Delta r^k - 1$ be an $n_w$-digit positive integer in radix $r$ representation where $r^{n_w-k-1} < \Delta \leq r^{n_w-k}$, $k = \left\lceil \frac{n_w-1}{2} \right\rceil$ and let $M' = -M^{-1} \bmod r^{n_w-1}$. Then it holds $M' = \Delta r^k + 1$.*

*Proof.* Analogous to the proof of Lemma 2.10, we write

$$\begin{aligned} MM' &= (\Delta r^k - 1)(\Delta r^k + 1) \\ &= \Delta^2 r^{2k} - 1 \\ &= -1 \bmod r^{n-1} \ . \end{aligned}$$

□

According to Lemmata 2.10 and 2.11, we can easily find two sets of moduli for which the precomputational step in Montgomery reduction can be excluded. The proposed sets are of type

$$\begin{array}{lll} \mathrm{S}_8: & M = \Delta_8 r^k + 1 & \text{where } r^{n_w-k-1} \le \Delta_8 < r^{n_w-k} \ ; \\ \mathrm{S}_9: & M = \Delta_9 r^k - 1 & \text{where } r^{n_w-k-1} < \Delta_9 \le r^{n_w-k} \ , \end{array} \tag{2.14}$$

where $k = \left\lceil \frac{n_w-1}{2} \right\rceil$. To further illustrate the properties of the two proposed sets, we give Fig. 2.17 where the moduli from each set are represented in radix 2 representation. Note that here $k = \left\lceil \frac{n-1}{2} \right\rceil$ and $m_i \in \{0, 1\}$.


Figure 2.17: Binary representation of the proposed sets $\mathrm{S}_8$ and $\mathrm{S}_9$.

The precomputed inverse needed for the Montgomery algorithm can be easily formed together with the modulus as:

$$M' = \begin{cases} \Delta_8 r^k - 1 & \text{if } M \in \mathrm{S}_8 \ ; \\ \Delta_9 r^k + 1 & \text{if } M \in \mathrm{S}_9 \ . \end{cases}$$

The modified algorithm is shown in Alg. 11.

### 2.5.1 On the Security of the Proposed Sets

As we can see from Figs. 2.16 and 2.17, one half of the modulus (either the most or the least significant half) is always filled with all 0's or all 1's. In the current state of the art, the security of ECC/HECC over prime fields GF($p$) does not depend

**Algorithm 11** Modular multiplication without precomputation based on Montgomery algorithm.

INPUT: $A = (A_{n_w-1} \dots A_0)_r$, $B = (B_{n_w-1} \dots B_0)_r$, $M \in \mathrm{S}_8 \cup \mathrm{S}_9$, $M'$, $R = r^{n_w}$, $0 \leq A, B < M$.
OUTPUT: $C = ABR^{-1} \bmod M$.
  $q_M \Leftarrow ABM' \bmod R$
  $C \Leftarrow (AB + q_M M)/R$
  **if** $C \geq M$ **then**
    $C \Leftarrow C - M$
  **end if**
  RETURN: $C$.

at all on the precise structure of the prime $p$. Hence, choosing a prime from the proposed sets has no influence on the security of ECC/HECC.

The security analysis for the RSA corresponds to attacks on RSA with partially known factorization. This problem has been analyzed extensively in the literature and the best attacks all rely on variants of Coppersmith's algorithm [35]. The best results in this area are as follows: let $N$ be an $n$-bit number, which is a product of two $n/2$-bit primes ($p$ and $q$). If half of the bits of either $p$ or $q$ (or both) are known, then $N$ can be factored in polynomial time. Hence, it is important to stress here that, due to the Coppersmith's method of factoring, the proposed sets of integers ($\mathrm{S}_6$, $\mathrm{S}_7$, $\mathrm{S}_8$, and $\mathrm{S}_9$) must not be used as prime factors for the RSA moduli. Clearly, our proposed method for bit-parallel modular multiplication is not applicable to RSA.

### 2.5.2 Bit-Parallel Finite Field Multiplication without Precomputation in GF($2^n$)

The ideas described in the previous section are further applied to finite fields of characteristic 2. A set of irreducible polynomials, for which the precomputational phase in Barrett reduction over GF($2^n$) is not needed, is defined by the following lemma. This lemma is the analogue of Lemma 2.8.

**Lemma 2.12.** *Let* $M(x) = x^n + \sum_{i=0}^{l} m_i x^i$ *and* $\mu(x) = x^{2n} \operatorname{div} M(x)$ *be polynomials over* GF(2)*, where* $l = \lfloor n/2 \rfloor$*. Then it holds* $\mu(x) = M(x)$*.*

*Proof.* In order to prove that the previous lemma holds we need to find a polynomial $B(x)$ of degree $n-1$ or less that satisfies $x^{2n} = M(x)^2 + B(x)$. Indeed, if we write

$x^{2n}$ as

$$x^{2n} = M(x)^2 + B(x)$$
$$= x^{2n} + \sum_{i=0}^{l} m_i x^{2i} + \sum_{i=0}^{n-1} b_i x^i \ ,$$

then we can choose coefficients $b_i$, $0 \leq i \leq n-1$, such that $b_{2j} = m_j$ and $b_{2j+1} = 0$, $0 \leq j \leq l$. This concludes the proof. □

Now, according to Lemma 2.12, we can define a set of moduli for which the Barrett reduction does not require a precomputational step. This set is of type

$$\mathrm{F}_3: \quad M(x) = x^n + \sum_{i=0}^{l} m_i x^i \quad \text{where } l = \left\lfloor \frac{n}{2} \right\rfloor \ ; \tag{2.15}$$

and the algorithm is shown in Alg. 12. Figure 2.18 illustrates its structure. It is interesting to note here that, for this special case, the irreducible polynomial can be chosen from the set that contains $2^{\lfloor n/2 \rfloor}$ different polynomials. This set represents the pool from which the irreducible polynomials are chosen.


Figure 2.18: Binary representation of the proposed set $\mathrm{F}_3$.

**Algorithm 12** Finite field multiplication based on Barrett reduction over GF($2^n$) without precomputation.

INPUT: $A(x) = \sum_{i=0}^{n-1} a_i x^i$, $B(x) = \sum_{i=0}^{n-1} b_i x^i$, $M(x) \in \mathrm{F}_3$, where $a_i, b_i \in \mathrm{GF}(2)$.
OUTPUT: $C(x) = A(x)B(x) \bmod M(x)$.
$C(x) = A(x)B(x) \operatorname{div} x^n$
$q(x) = C(x)M(x) \operatorname{div} x^n$
$C(x) = \big(A(x)B(x) + q(x)M(x)\big) \bmod x^n$
RETURN: $C(x)$.

In order to prove the correctness of Alg. 12, we introduce the following definition.

**Definition 2.6.** Let $P(x)$ and $Q(x)$ denote arbitrary polynomials of degree $p$ and $q$, respectively. We define $\Delta(n) = \frac{P(x)}{Q(x)}$ such that $n = p - q$ and $n \in \mathbb{Z}$. In other words, with $\Delta(n)$ we denote an arbitrary element from the set of all rational functions of degree n.

*Proof of Alg. 12.* We provide the proof for any modulus $M(x)$ where $\mu(x) = x^{2n} \operatorname{div} M(x)$. Consequently, the proof holds for our special case of $M(x) \in \mathrm{F}_3$ and $\mu(x) = M(x)$. Let us first introduce the following notations: $Q(x) = A(x)B(x) \operatorname{div} M(x)$, $Q_1(x) = A(x)B(x) \operatorname{div} x^n$, $Q_2(x) = \mu(x)Q_1(x)$, and $Q_3(x) = Q_2(x) \operatorname{div} x^n$.

Using notation from Def. 2.6 and starting from the original Barrett reduction we can write

$$\begin{aligned}\mu(x) &= x^{2n} \operatorname{div} M(x) \\ &= \frac{x^{2n}}{M(x)} + \Delta(-1) \ .\end{aligned}$$

Similarly, we can express $Q_1(x)$ as

$$\begin{aligned}Q_1(x) &= A(x)B(x) \operatorname{div} x^n \\ &= \frac{A(x)B(x)}{x^n} + \Delta(-1) \\ &= \left(\frac{Q(x)M(x)}{x^n} + \Delta(-1)\right) + \Delta(-1) \\ &= \frac{Q(x)M(x)}{x^n} + \Delta(-1) \ ,\end{aligned}$$

Using the previous equations, $Q_2(x)$ and $Q_3(x)$ can be written as

$$\begin{aligned}Q_2(x) &= \mu(x)Q_1(x) \\ &= \left(\frac{x^{2n}}{M(x)} + \Delta(-1)\right)\left(\frac{Q(x)M(x)}{x^n} + \Delta(-1)\right) \\ &= Q(x)x^n + \frac{x^{2n}}{M(x)}\Delta(-1) + \frac{Q(x)M(x)}{x^n}\Delta(-1) + \Delta(-1)\Delta(-1) \\ &= Q(x)x^n + \Delta(n-1) + \Delta(n-1) + \Delta(-2) \\ &= Q(x)x^n + \Delta(n-1) \ , \\ Q_3(x) &= Q_2(x) \operatorname{div} x^n \\ &= \Big(Q(x)x^n + \Delta(n-1)\Big) \operatorname{div} x^n \\ &= Q(x) \ .\end{aligned}$$

Finally, we can evaluate $C(x) = A(x)B(x) \bmod M(x)$ as

$$\begin{aligned} C(x) &= A(x)B(x) \bmod x^n + M(x)\big(A(x)B(x) \operatorname{div} M(x)\big) \bmod x^n \\ &= A(x)B(x) \bmod x^n + M(x)Q(x) \bmod x^n \\ &= A(x)B(x) \bmod x^n + M(x)Q_3(x) \bmod x^n \ . \end{aligned}$$

This concludes the proof. □

Lemma 2.13 defines the set of irreducible polynomials for which the precomputational step in Montgomery reduction over GF($2^n$) is not needed.

**Lemma 2.13.** *Let* $M(x) = \sum_{i=l}^{n} m_i x^i + 1$ *and* $M'(x) = M(x)^{-1} \bmod x^n$ *be polynomials over GF*(2)*, where* $l = \lceil n/2 \rceil$*. Then it holds* $M'(x) = M(x)$*.*

*Proof.* In order to prove the lemma we need to show that $M(x)^2 = 1 \bmod x^n$. Indeed, if we write $M(x)^2$ as

$$\begin{aligned} M(x)^2 &= M(x)M(x) \\ &= \sum_{i=l}^{n} m_i x^{2i} + 1 \ , \end{aligned}$$

it becomes obvious that $M(x)^2 = 1 \bmod x^n$, since $l = \lceil n/2 \rceil$. This concludes the proof. □

The set is now defined as

$$\mathrm{F}_4: \quad M(x) = \sum_{i=l}^{n} m_i x^i + 1 \quad \text{where } l = \left\lceil \frac{n}{2} \right\rceil \ ; \tag{2.16}$$

and the algorithm is shown in Alg. 13. Its illustration is given in Fig. 2.19. Similar to the case of Barrett, this set also contains $2^{\lfloor n/2 \rfloor}$ elements, out of which the irreducible polynomials are chosen.

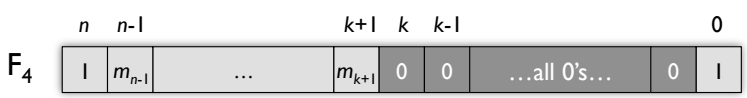


Figure 2.19: Binary representation of the proposed set $\mathrm{F}_4$.

**Algorithm 13** Montgomery multiplication over GF($2^n$) without precomputation.

INPUT: $A(x) = \sum_{i=0}^{n-1} a_i x^i$, $B(x) = \sum_{i=0}^{n-1} b_i x^i$, $M(x) \in \mathrm{F}_4$, $R(x) = x^n$, where $a_i, b_i \in \mathrm{GF}(2)$.
OUTPUT: $C(x) = A(x)B(x)R^{-1}(x) \bmod M(x)$.
$C(x) = A(x)B(x)$
$q_M(x) = C(x)M(x) \bmod R(x)$
$C(x) = \big(C(x) + q_M(x)M(x)\big) \operatorname{div} R(x)$
RETURN: $C(x)$.

*Proof of Alg. 13.* Similar to the case of finite field multiplication based on Barrett reduction, here we also provide the proof for any modulus $M(x)$ where $M'(x) = M(x)^{-1} \bmod R(x)$. Let $D(x) = A(x)B(x)$ and let $Q_1(x) = D(x) \bmod R(x)$, $Q_2(x) = M'(x)Q_1(x) \bmod R(x)$, and $Q_3(x) = M(x)Q_2(x)$.

The polynomial $D(x)$ can be written as $D(x) = D_1(x)R(x) + D_0(x)$, where $R(x) = x^n$. There exists a polynomial $S(x)$ of degree $n-1$ such that

$$D_0(x) + M(x)S(x) = 0 \bmod R(x) \ ,$$

In other words, $S(x)$ can be expressed as

$$S(x) = -D_0(x)M(x)^{-1} \bmod R(x) \ .$$

At the same time it holds

$$D(x) + M(x)S(x) = D(x) \bmod M(x) \ ,$$

$$D(x) + M(x)S(x) = 0 \bmod R(x) \ .$$

Finally, we have

$$\begin{aligned} C(x) &= \big(D(x) + M(x)S(x)\big) \operatorname{div} R(x) \\ &= D(x)R(x)^{-1} \bmod M(x) \ . \end{aligned}$$

Now, it is obvious that:

$$M'(x) = -M^{-1}(x) \bmod R(x)$$

$$Q_1(x) = D_0(x)$$

$$Q_2(x) = S(x)$$

$$Q_3(x) = M(x)S(x) \ .$$

This concludes the proof. □

### 2.5.3 Summary

Four distinct sets of primes and two distinct sets of irreducible polynomials, for which the precomputational step in bit-parallel modular multiplication algorithms can be excluded, are introduced in this section. The proposed methods for modular multiplication are very suitable for fast hardware implementations of some public-key cryptosystems and in particular of Elliptic Curve Cryptography.

## 2.6 Tripartite Modular Multiplication

### Publication Data

K. Sakiyama, M. Knežević, J. Fan, B. Preneel, and I. Verbauwhede, "Tripartite Modular Multiplication," *Integration, the VLSI journal*, 14 pages, 2011. To appear.

#### Personal Contributions

- Involved in: Implementation; Cost and performance estimation; Text writing.

Our novel contribution consists of proposing a new modular multiplication algorithm. The algorithm is similar to the existing bipartite modular multiplication and by using the approach of Karatsuba, it introduces additional level of parallelism.

The bipartite modular multiplication was introduced in Section 2.4.5 and its performance was further improved by choosing a special set of moduli. By integrating the classical modular multiplication with Montgomery multiplication, the BMM algorithm achieves an additional level of parallelism, which is indeed the main feature of the algorithm. In this section we propose a new modular multiplication algorithm that effectively integrates three algorithms, a classical modular multiplication based on Barrett reduction, the Montgomery multiplication and the Karatsuba multiplication. The novelty comes at higher algorithmic level and the performance can be further improved by parallelizing any of its ingredients (Barrett, Montgomery or Karatsuba multiplication). The proposed algorithm minimizes the number of single-precision multiplications and enables more than 3-way parallel computation. This section investigates the cost and speed trade-offs for a hardware implementation of the proposed modular multiplication and compares the results with implementations of previous algorithms.

Among the integer or polynomial multiplication techniques, two important methods are Karatsuba algorithm [84] and its generalization – Toom-Cook's algorithm (sometimes known as Toom-3) [166]. They both reduce the number of single-precision multiplications by reusing the intermediate partial products. By

recursively using Karatsuba's method, one multiplication of two $n_w$-digit integers has complexity of $\mathcal{O}(n_w^{\log_2 3})$, while Toom-$k$ has complexity $\mathcal{O}(n_w^{\log_k 2k-1})$. Both algorithms can provide faster multiplication than the normal schoolbook method. Karatsuba's algorithm can accelerate multiplication by representing two $n_w$-digit integers as $A = A_1R + A_0$ and $B = B_1R + B_0$, where $R = 2^k$ is chosen for an efficient implementation. Then, the product of $AB$ can be computed as

$$AB = p_1R^2 + (p_2 - p_0 - p_1)R + p_0 \ , \tag{2.17}$$

where

$$p_0 = A_0B_0,\ p_1 = A_1B_1,\ p_2 = (A_0 + A_1)(B_0 + B_1) \ . \tag{2.18}$$

Therefore, we need only three sub-product multiplications while the standard, schoolbook multiplication needs four sub-products. The highest speed is achieved when choosing $k$ to be around $n_w/2$. By using Karatsuba's method recursively, the time complexity becomes $\mathcal{O}(n_w^{\log_2 3})$.

### 2.6.1 Overview of the Proposed Multiplication Algorithm

We explain the proposed algorithm by starting from the basic version that is based on the following equation derived from Karatsuba's algorithm.

$$\begin{aligned} ABR^{-1} \bmod M =& (A_1R + A_0)(B_1R + B_0)R^{-1} \bmod M \\ =& \left\{A_1B_1R + (A_1B_0 + A_0B_1) + A_0B_0R^{-1}\right\} \bmod M \\ =& \big\{p_1R \bmod M \\ & + (p_2 - p_0 - p_1) \bmod M \\ & + p_0R^{-1} \bmod M\big\} \bmod M \ , \end{aligned} \tag{2.19}$$

where

$$p_0 = A_0B_0,\ p_1 = A_1B_1,\ p_2 = (A_0 + A_1)(B_0 + B_1) \ . \tag{2.20}$$

In Eq. (2.19), $n_w$-digit inputs $A$ and $B$ are split into two blocks as $A = (A_1, A_0)_R$ and $B = (B_1, B_0)_R$, and then Karatsuba's method is applied for performing multiplication of $AB$. Here, $R$ is chosen as $R = r^k$ where $k = \lceil n_w/2 \rceil$ for an

efficient implementation although $k$ can be arbitrarily chosen. We call this case a $u$-split version where $u = 2$. In total, we have three terms that can be computed independently by using the existing algorithms described in the previous sections. To obtain a high-speed implementation, one can compute these three different terms in parallel. Figure 2.20 explains the main idea of the proposed algorithm and compares with the bipartite algorithm. For a modular multiplication with $n_w$-digit polynomial-basis inputs, $A(x) = A_1(x)R(x)+A_0(x)$ and $B(x) = B_1(x)R(x)+B_0(x)$ where $R(x) = x^k$, we can use the same sequence as shown in Eq. (2.19).

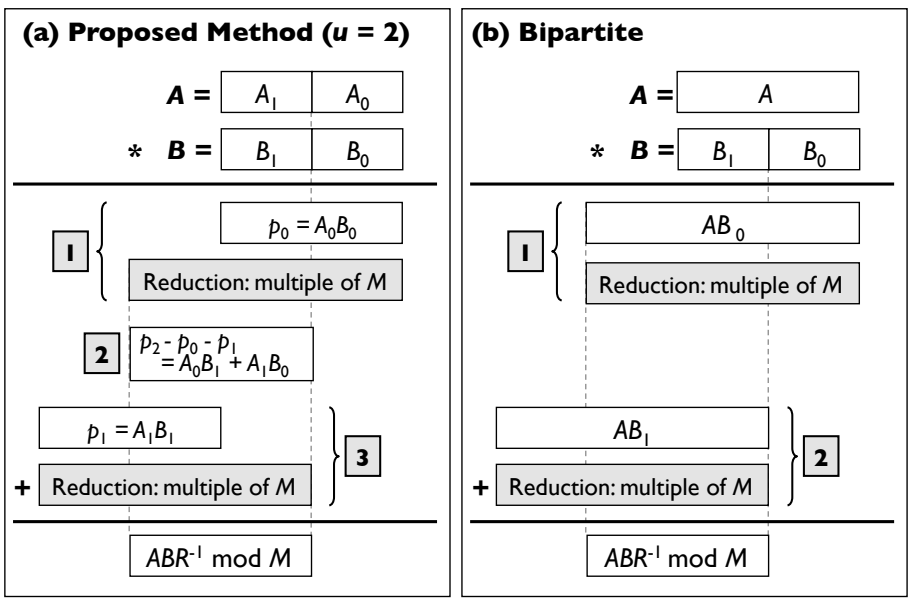


Figure 2.20: Procedure for modular multiplication. (a) our proposed method. (b) bipartite method.

## 2.6.2 Further Exploration of the Proposed Algorithm

For a further exploration of the proposed algorithm, we can split the inputs into more blocks, e.g. $A = (A_3, A_2, A_1, A_0)_R$ for an $n_w$-digit integer input and $A(x) = \sum_{i=0}^{3} A_i(x)R(x)^i$ for an $n_w$-digit polynomial-basis value, where $R = r^k$ and $R(x) = x^k$ where $k = \lceil n_w/4 \rceil$. In this example case of $u = 4$, we can explore

further parallelism as

$$\begin{aligned} ABR^{-2} \bmod M = \Big[ & p_3 R^4 \bmod M + (p_7 - p_2 - p_3) R^3 \bmod M \\ & + \big\{ (p_6 - p_1 + p_2 - p_3) R^2 \\ & + (p_8 + p_0 + p_1 + p_2 + p_3 - p_4 - p_5 - p_6 - p_7) R \\ & + (p_5 - p_0 + p_1 - p_2) \big\} \bmod M \\ & + (p_4 - p_0 - p_1) R^{-1} \bmod M + p_0 R^{-2} \bmod M \Big] \bmod M \ , \end{aligned} \tag{2.21}$$

where

$$\begin{aligned} & p_0 = A_0 B_0, \ p_1 = A_1 B_1, \ p_2 = A_2 B_2, \ p_3 = A_3 B_3 \ , \\ & p_4 = (A_0 + A_1)(B_0 + B_1), \ p_5 = (A_0 + A_2)(B_0 + B_2) \ , \\ & p_6 = (A_1 + A_3)(B_1 + B_3), \ p_7 = (A_2 + A_3)(B_2 + B_3) \ , \\ & p_8 = (A_0 + A_1 + A_2 + A_3)(B_0 + B_1 + B_2 + B_3) \ . \end{aligned} \tag{2.22}$$

This example case allows us to perform modular multiplication up to 5-way parallel computing as shown in Eq. (2.21). Parameters $p_0, p_1, \ldots, p_8$ in Eq. (2.22) are computed with complexity of 9 sub-product multiplications and 14 additions.

For the reduction steps, we apply Barrett and Montgomery reduction to the terms that require reduction (i.e. 1st, 2nd, 6th and 7th term in Eq. (2.21)). For the other terms, we use a simple modular addition or subtraction. The final reduction step is performed after adding up all the partial results derived from each term.

Due to the carry-free arithmetic, for a (modular) multiplication over a binary field, the reduction is only needed for terms that require Barrett and Montgomery reduction (again, these are 1st, 2nd, 6th and 7th term in Eq. (2.21)). Figure 2.21 a summarizes the 4-split version of the proposed algorithm.

For a better area-performance trade-off, the method described in Fig. 2.21 a can be modified as shown in Fig. 2.21 b. It illustrates a 3-way parallel computational sequence equivalent to the one in Fig. 2.21 a. The 3-way parallel version can save area cost by sharing the hardware for reduction. The critical path delay increases slightly due to the one extra addition before the reduction. However, this

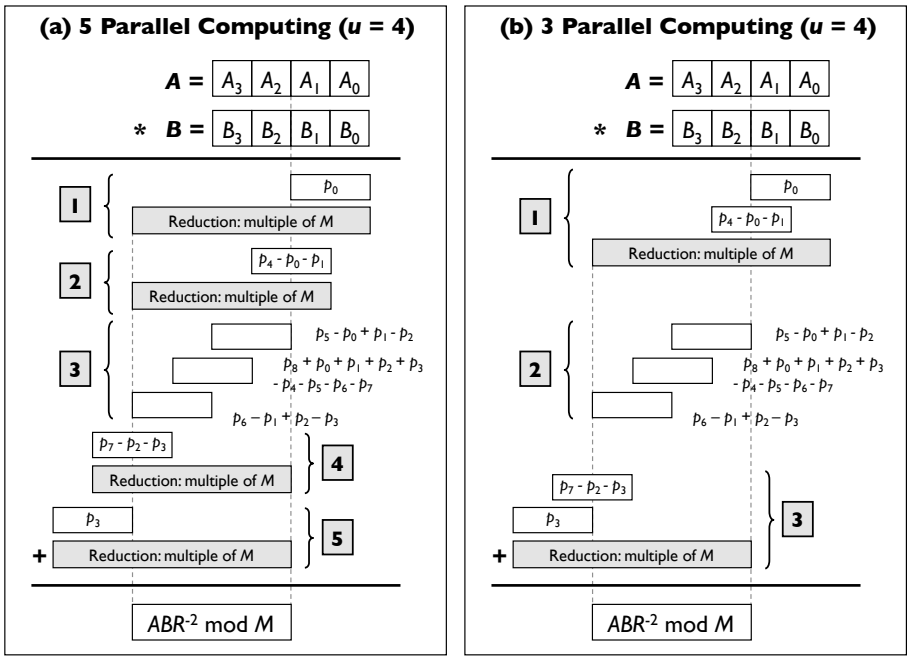


Figure 2.21: Procedure for modular multiplication for $u = 4$. (a) five-way parallel computation. (b) three-way parallel computation.

speed penalty occurs only for the case of integer multiplication where the carry propagation is present.

To illustrate further, we also mention the case of $u = 8$ for which we need 27 $\lceil n_w/8 \rceil \times \lceil n_w/8 \rceil$-digit multiplications to prepare the partial products $p_0, p_1, \ldots, p_{26}$ since each $\lceil n_w/4 \rceil \times \lceil n_w/4 \rceil$-digit multiplication in Eq. (2.22) can be computed with three $\lceil n_w/8 \rceil \times \lceil n_w/8 \rceil$-digit by using Karatsuba's method. In general, we need $3^v$ $\lceil n_w/u \rceil \times \lceil n_w/u \rceil$-digit multiplications for $u = 2^v$, where $v$ is a non-negative integer.

Finally, to give an idea of possible improvements of our algorithm, we provide Fig. 2.22 which represents the hierarchy of modular multiplication. In order to further improve the performance of tripartite (and bipartite) multiplication, one can use an optimized (pipelined, parallelized, etc) implementation of any of the levels below. For example, the pipelined implementation of Montgomery multiplication, as described by Suzuki [160], can be used to further parallelize our algorithm. However, in this section we focus on the higher algorithmic level and therefore we leave this practical question open for further analysis.

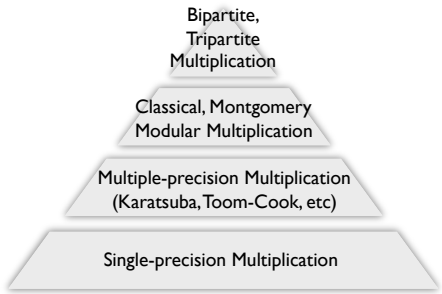


Figure 2.22: Hierarchy of the modular multiplication.

### 2.6.3 Cost and Performance Estimation

In order to have a fast multiplier while still preserving a relatively low area, we consider here a digit-serial approach. We further assume that the multiplier has enough parallel processing units such that the full degree of parallelism can be exploited both for bipartite and tripartite algorithms. In short, we consider here so-called serial-parallel multipliers.

To make a fair comparison of modular multipliers implemented using different algorithms we use the following criteria. A computational complexity of the algorithm is considered to be the number of single-precision multiplications necessary for performing the whole algorithm. Therefore, we assume that an addition can be implemented very efficiently in hardware. Since the only requirement for achieving full parallelism is to use a sufficient number of single-precision multipliers, we consider only the number of single-precision multipliers as the area estimation. We stress here that the size of the whole architecture will, of course, be only proportional and not equal to the size of all single-precision multipliers. The size of a single-precision multiplier is $w \times w$ bits.

We assume that, besides maximizing the throughput, an important goal is to keep the number of single-precision multipliers as small as possible. Hence, we consider the case with the minimal number of single-precision multipliers while still allowing fully parallel processing. The critical path delay is assumed to be the same in all cases and equal to the delay of one single-precision multiplier – $t_{SP}$. Finally, the total speed-up is considered for the case when $n_w \to \infty$. This is a theoretical result and will be different in practice as the final speed also depends on the physical size of a design.

The number of single-precision multiplications is the main criterium for the speed comparison. Therefore, we first calculate the computational complexities for the classical modular multiplication based on Barrett's algorithm, Montgomery

multiplication and bipartite modular multiplication, respectively. We assume that addition in hardware can be implemented in a single clock cycle at a reasonable hardware cost.

The complexity of the classical modular multiplication based on Barrett reduction can be simply obtained by analyzing Alg. 3. At step three of the algorithm we need to perform $n_w$ single-precision multiplications. Using the trick of Barrett, step four can be performed at the cost of a single digit multiplication. Finally, step five of the algorithm requires another $n_w$ digit multiplications. In total, to perform a classical modular multiplication, we need $2n_w^2 + n_w$ single-precision multiplications.

By analyzing Alg. 4 we conclude that the complexity of Montgomery multiplication is also equal to $2n_w^2 + n_w$ single-precision multiplications.

Assuming that we combine the classical modular multiplication based on Barrett reduction and Montgomery multiplication to implement the BMM algorithm, the complexity becomes equal to $n_w^2 + n_w/2$ single-precision multiplications (for the case of $k = \lceil n_w/2 \rceil$). Although it requires more resources, the BMM algorithm can speed up a modular multiplication by up to two times.

Next, we provide the analysis of computational complexity for the proposed, tripartite algorithm. Let us consider the general case, namely a $u$-split version of the proposed algorithm. Assuming the algorithm being fully parallelized, the most time-consuming part is the classical modular multiplication based on Barrett reduction and the Montgomery multiplication. As discussed in Section 2.3.2, in order to compute the $n_w \times n_w$-digit modular multiplication, they use $\lambda = w + 4$-bit and $\lambda = w$-bit digit single-precision multiplications, respectively. Step three of both Alg. 3 and Alg. 4 requires $n_w/u$ single-precision multiplications, while step four requires only one single-precision multiplication. Since the modulus $M$ is an $n_w$-digit integer, step five of both algorithms still requires $n_w$ single-precision multiplications. Finally, to perform a full $n_w \times n_w$-digit modular multiplication we need

$$\frac{n_w}{u}\left(\frac{n_w}{u} + 1 + n_w\right) = \frac{u+1}{u^2}n_w^2 + \frac{1}{u}n_w$$

single-precision multiplications.

The data in Table 2.4 only represents the theoretical result based on the number of single-precision multipliers used in our design. To show the practical value of our proposal, we implement an interleaved modular multiplier and compare it with the implementations of other algorithms. Next, we discuss the hardware implementation in detail.

Table 2.4: Cost and performance estimation for an interleaved modular multiplication.

| Algorithm | Number of SP multipliers† | Number of SP multiplications† | Critical Path Delay‡ | Speed-Up $[n \to \infty]$ |
|---|---|---|---|---|
| Classical | 1 | $2n_w^2 + n_w$ | $t_{SP}$ | 1 |
| Montgomery | 1 | $2n_w^2 + n_w$ | $t_{SP}$ | 1 |
| Bipartite | 2 | $n_w^2 + \frac{1}{2}n_w$ | $t_{SP}$ | 2 |
| Proposed ($u = 2$) | 3 | $\frac{3}{4}n_w^2 + \frac{1}{2}n_w$ | $t_{SP}$ | 2.67 |
| Proposed ($u = 4$) | 9 | $\frac{5}{16}n_w^2 + \frac{1}{4}n_w$ | $t_{SP}$ | 6.40 |
| Proposed ($u = 8$) | 27 | $\frac{9}{64}n_w^2 + \frac{1}{8}n_w$ | $t_{SP}$ | 14.22 |

†A single-precision multiplier is of size $w \times w$ bits.

‡A critical path delay is assumed to be the latency of a single-precision multiplier.

### 2.6.4 Hardware Implementation of the Proposed Algorithm

In order to verify our algorithm in practice, we implement the proposed solution on two different hardware platforms. First, we provide the figures for the Xilinx Virtex-5 FPGA board (xc5vlx50t-ff1136). Second, we provide the ASIC synthesis results using UMC 0.13 $\mu$m CMOS process and Synopsys Design Compiler version C-2009.06-SP3. Additionally, we implement the classical multiplier based on Barrett reduction, the Montgomery and the bipartite multipliers, and compare them with our results.

For the FPGA implementation, our strategy is to use dedicated DSP48E slices on the board for implementing single-precision multipliers. Implemented this way, the whole design achieves a higher speed and consumes less bit-slices. However, DSP48E slices are a limited resource on our testing FPGA board and hence the goal is to use the minimal number of single-precision multipliers, yet allowing the full parallelism on the algorithmic level.

The described strategy results in an architecture for classical and Montgomery algorithms as shown in Fig. 2.23. As mentioned in Section 2.4.3, we use $\lambda = w$ bits digit-size for the case of Montgomery and $\lambda = w + 4$ bits digit-size for the case of classical modular multiplication based on Barrett reduction. A single $\lambda \times \lambda$ single-precision multiplier is used in both cases (inside the digit-serial multiplier $\pi_1$).

An architecture for the bipartite method is described in Fig. 2.24. It consists of a classical modular multiplier based on Barrett reduction and a Montgomery multiplier. Two digit-serial multipliers ($\pi_1$ and $\pi_2$) were used, each containing one single-precision multiplier.

Finally, our proposed architecture for the $u = 2$-split version is depicted in Fig. 2.25.

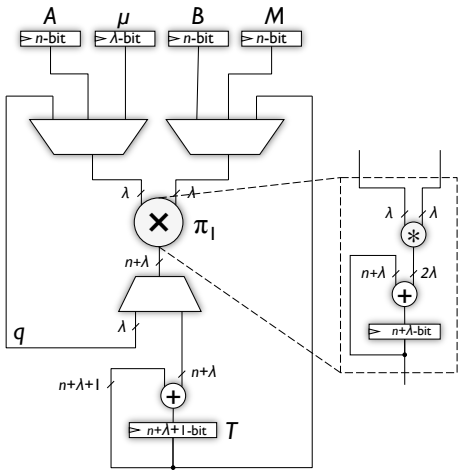


Figure 2.23: Datapath of the modular multiplier based on classical and Montgomery algorithms.

As discussed in Section 2.6.1, it consists of a classical modular multiplier based on Barrett reduction ($\pi_1$), Montgomery multiplier ($\pi_2$) and Karatsuba multiplier in the middle ($\pi_3$) – all running in parallel. Instead of computing $A_1B_1R \bmod M$, the classical modular multiplier on the left-hand side of Fig. 2.25 computes $A_1B_1R \bmod M - A_1B_1$ and stores the result in register $T_H$ where $A_1B_1$ is a partial product necessary for the Karatsuba part. The same holds for the Montgomery multiplier which instead of $A_0B_0R^{-1} \bmod M$ computes $A_0B_0R^{-1} \bmod M - A_0B_0$ and stores the result in register $T_L$. Finally, all three parts are added together, resulting in the final $n + 2$-bit product.

Since in most cryptographic applications the result of one modular multiplication is used as input to another, successive modular multiplication, the result needs to be reduced and remain strictly smaller than the modulus. At most three subtractions are necessary for this final reduction.

Table 2.5 shows the results of our FPGA implementations. We have implemented a $192 \times 192$-bit modular multiplier with digit size of $w = 32$ bits, based on all the mentioned algorithms and compared them with our solution. All the designs were implemented using maximum speed as the main constraint. The results clearly show that concerning speed, our algorithm outperforms all the previously proposed solutions. As the comparison criteria we use the relative throughput defined as:

$$T_r = \frac{\text{Frequency}}{\text{Number of cycles}} \ .$$

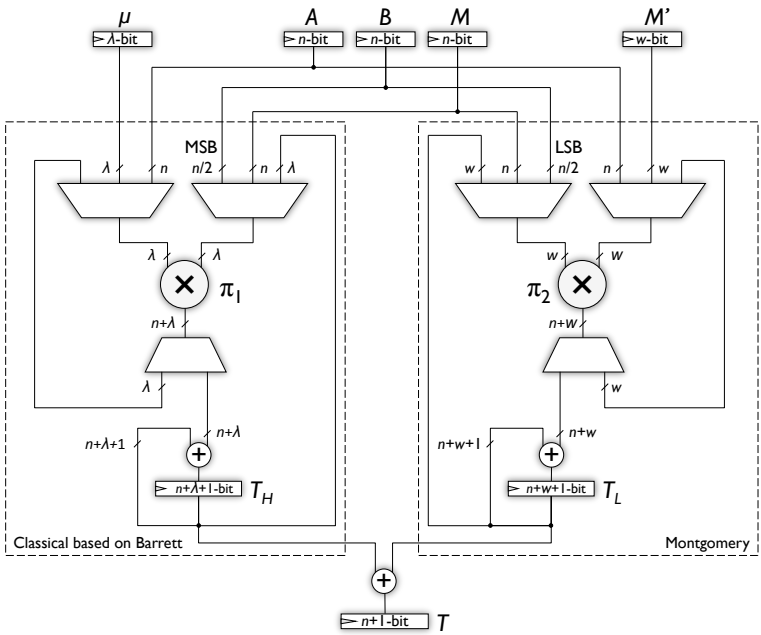


Figure 2.24: Datapath of the modular multiplier based on the bipartite algorithm.

An area overhead of about 660 bit-slices, compared to the bipartite design, is mainly due to the use of additional adders and multiplexers necessary for the control logic. A better resource sharing is also possible at the cost of maximum frequency and therefore we have obtained a design of 1408 bit-slices running at a frequency of 69 MHz. Finally, a speed-up of almost 150 % compared to the classical method and about 25 % compared to the bipartite method is obtained.

Table 2.5: Comparison of FPGA implementations for a 192 × 192-bit modular multiplier. Target platform: Xilinx Virtex 5 FPGA board (xc5vlx50t-ff1136).

| Algorithm | Bit slices | DSP48E slices | Number of cycles | Frequency [MHz] | $T_r$ [MHz] | Speed Up |
|---|---|---|---|---|---|---|
| Classical | 988 | 6 | 78 | 77 | 0.987 | 1 |
| Montgomery | 849 | 4 | 78 | 102 | 1.308 | 1.325 |
| Bipartite | 1313 | 10 | 39 | 77 | 1.974 | 2 |
| Proposed ($u = 2$) | 1979 | 14 | 30 | 74 | 2.467 | 2.499 |

Table 2.6 shows the results of our ASIC synthesis results. The same architectures

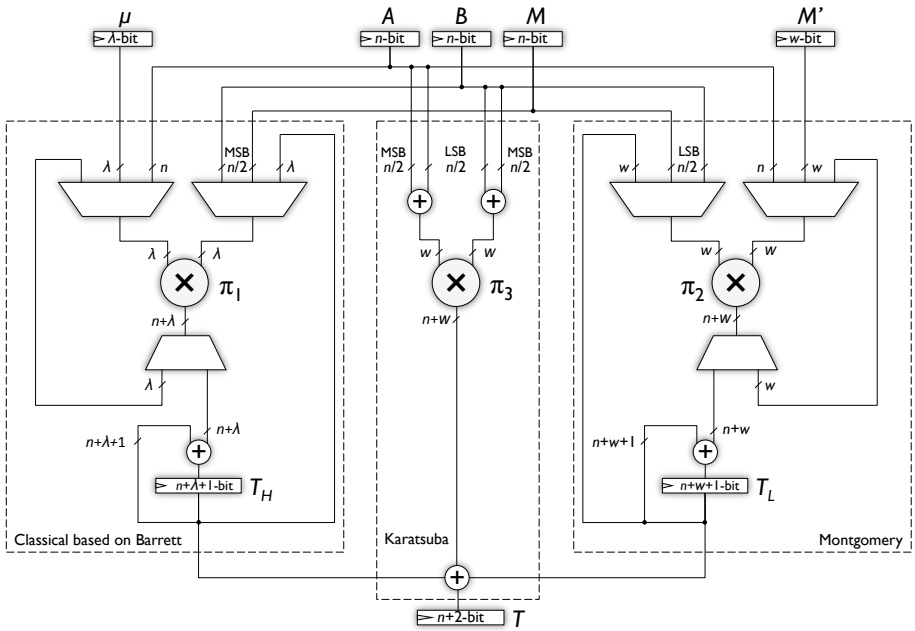


Figure 2.25: Datapath of the modular multiplier based on the proposed algorithm.

as in the case of FPGA were synthesized and compared to each other. As can be observed, concerning the speed performance, our proposal outperforms the classical modular multiplier by nearly 3 times and the bipartite multiplier by 45 %. However, this comes at a larger hardware cost and therefore our multiplier is around 2.38 times bigger than the classical one and about 56 % bigger than the bipartite multiplier.

Table 2.6: Comparison of ASIC implementations for a 192 × 192-bit modular multiplier. Target platform: UMC 0.13 $\mu$m CMOS technology (Synopsys Design Compiler version C-2009.06-SP3, synthesis results).

| Algorithm | Area [kGE] | Number of cycles | Frequency [MHz] | $T_r$ [MHz] | Speed Up |
|---|---|---|---|---|---|
| Classical | 42.33 | 78 | 191 | 2.448 | 1 |
| Montgomery | 31.70 | 78 | 229 | 2.936 | 1.199 |
| Bipartite | 64.45 | 39 | 193 | 4.949 | 2.022 |
| Proposed ($u = 2$) | 100.77 | 30 | 215 | 7.167 | 2.928 |

### 2.6.5 Summary

We have introduced a new modular multiplication algorithm suitable for efficient hardware implementations. We have also implemented a hardware modular multiplier based on the proposed algorithm that effectively integrates three different algorithms, the classical modular multiplication based on Barrett reduction, Montgomery multiplication and Karatsuba multiplication. The results show that, speed-wise, our proposed algorithm outperforms all the previously proposed solutions.

We believe that the proposed algorithm offers a new alternative for efficient modular multiplication on both software and hardware platforms. The cost and performance trade-offs when increasing the value of $u$ further (i.e. $u > 8$) still need to be explored. The software implementation and the scheduling problem on multicore platforms still remains a challenge. We also plan to implement the proposed algorithm for multiplication in binary field. Another direction for future work would be to use the Toom-Cook algorithm for the multiplication step instead of Karatsuba's method.

## 2.7 Conclusion

In this chapter we explicitly showed how some of the most adopted algorithms today, i.e. Barrett, Montgomery, and bipartite, can be further accelerated by using special classes of moduli. The algorithms have shown to be secure for use in today's public-key cryptosystems.

As a final contribution to this chapter, we proposed a new, so-called, tripartite modular multiplication, an algorithm that benefits from combining the algorithms of Barrett, Montgomery, and Karatsuba. By its construction, the algorithm provides ample parallelism and therefore offers a possibility to fully explore the trade-off between area and speed.

# Chapter 3

# High-Throughput Hardware Implementations of Cryptographic Hash Functions

## 3.1 Introduction

Hash functions are deterministic mathematical algorithms that map arbitrary length sequences of bits into a hash result of a fixed, limited length. The mapping is done in a way that these hash results can be considered as unique fingerprints.

Cryptographic hash algorithms are one of the most important primitives in security systems. They are most commonly used for digital signature algorithms [129], message authentication and as building blocks for other cryptographic primitives such as hash based block ciphers, e.g. Bear, Lion [9] and Shacal [63], stream ciphers and pseudo-random number generators.

The typical example of hash based message authentication is protecting the authenticity of the short hash result instead of protecting the authenticity of the whole message. Consequently, in digital signatures the signing algorithm is always applied to the hash result rather than to the original message. This ensures both performance and security benefits. Hash algorithms can also be used to compare two values without revealing them. The typical examples for this application are password authentication mechanisms.

At the same time the need for high-throughput optimized implementations of the hash functions is getting essential in almost every security application. Due to the

iterative mode of operations, an efficient implementation of the hash algorithm has always been a challenge. In this chapter we focus on hardware implementations of hash functions and we specifically target high-throughput implementations. We propose several techniques that result in a higher speed of the design, often coming at the cost of higher area requirements. Security, on the other side, remains unaffected. Hence, Fig. 3.1 illustrates a typical trade-off that is examined throughout this chapter.

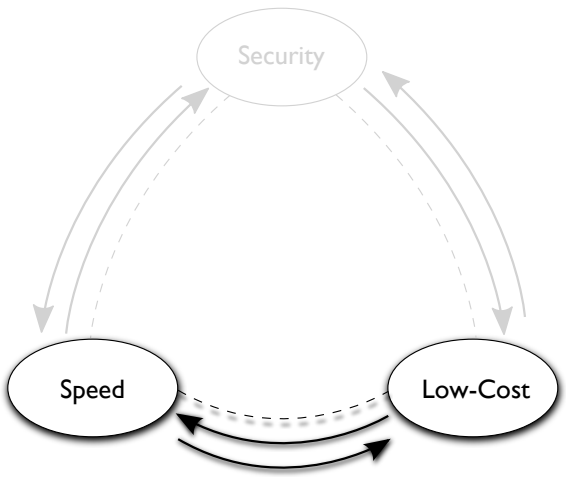


Figure 3.1: Speed versus low-cost trade-off.

As the hash algorithms are widely used in many security applications, it is very important that they fulfill certain security properties. Depending on the application the hash function is used in, some of the properties are more important than the others. Generally, it is accepted that a cryptographic hash function has to satisfy the following three requirements:

1. *Preimage resistance*: It must be hard to find any preimage for a given hash output, i.e. given a hash output H getting a message M such that $\mathrm{H} = Hash(\mathrm{M})$ must be hard.

2. *Second Preimage resistance*: It must be hard to find another preimage for a given input, i.e. given $\mathrm{M}_0$ and $Hash(\mathrm{M}_0)$ getting $\mathrm{M}_1 \neq \mathrm{M}_0$ such that $Hash(\mathrm{M}_0) = Hash(\mathrm{M}_1)$ must be hard.

3. *Collision resistance*: It must be hard to find two different inputs with the same hash output, i.e. getting $\mathrm{M}_0$ and $\mathrm{M}_1$ such that $Hash(\mathrm{M}_0) = Hash(\mathrm{M}_1)$ must be hard.

Furthermore, it should be easy to compute the hash value for any given message. A hash algorithm that is characterized by the first two properties is called a *One-Way Hash Algorithm*. If all three properties are met we say that the hash algorithm is *Collision Resistant*. Finding collisions of a specific hash algorithm is the most common way of attacking it.

There are a few different types of hash algorithms described in the literature. Some of them are based on block ciphers, modular arithmetic, cellular automata, knapsack and lattice problems, algebraic matrices, etc. Historically, most cryptographic hash functions were based on block ciphers, e.g. the early DES-based hash function proposed by Rabin [140] in 1978. Preneel et al. [138] described in a systematic way how a block cipher can be used to construct a hash function whose output size corresponds to the block size of the cipher. The most commonly used hash algorithms, known as *dedicated hash algorithms*, are especially designed for hashing and are not provably secure. In 1990, Rivest designed the dedicated hash function MD4 [141]. Nowadays, the biggest class of these algorithms is based on design principles of the MD4 family.

A brief overview of the basic hash applications as well as a detailed discussion about the possible hash constructions is given in the 'Encyclopedia of Cryptography and Security' by Tilborg [168], chapter 'Hash Functions'. More details concerning this topic can be found in chapter 9 of 'Handbook of Applied Cryptography' [116]. For a thorough survey of analysis and design of cryptographic hash functions we refer to Preneel's dissertation [137].

The chapter is organized as follows. Section 3.2 gives a brief historical overview of the most popular hash algorithms and their security considerations. Section 3.3 discusses several basic DSP techniques for improving the throughput. In Section 3.4 we give an example of speeding up an algorithm by using some of the algorithm-specific techniques and illustrate this approach by implementing the RIPEMD-160 hash function. The extensive hardware comparison of fourteen second-round SHA-3 candidates is treated in Section 3.5. Finally, in Section 3.6 we discuss in detail a hardware evaluation of the Luffa hash family.

## 3.2 Popular Hash Algorithms and Their Security Considerations

The design philosophy of the most commonly used hash algorithms such as MD5, the whole SHA family, and RIPEMD is based on design principles of the MD4 family. In this section we give a short overview and provide historical facts about existing attacks on these algorithms.

MD4 is a 128-bit cryptographic hash algorithm introduced by Rivest in 1990 [141].

The structure of MD4 is based on basic arithmetic operations and several Boolean functions. After it was proposed, several other hash algorithms were constructed based on the same design principles: a 256-bit extension of MD4 [141], MD5 [142], HAVAL [182], RIPEMD [26], RIPEMD-160 [48], SHA-0 [5], SHA-1 [6], SHA-256, SHA-384, SHA-512, SHA-224 [7], etc. The first attack on MD4 was published already in 1991 by Merkle, den Boer, and Bosselaers [40]. The attack was performed on a reduced version of MD4 (2 out of 3 rounds). Additionally, in November 1994, Vaudenay mounts an attack where the last round is omitted [169]. In Fall of 1995, Dobbertin finds collisions for all three rounds of MD4 [47]. A few years after Rivest designed the strengthened version MD5, it was shown by den Boer and Bosselaers [41] that the compression function of MD5 is not collision resistant. At the beginning of 1996, Dobbertin also found a free-start collision of MD5 in which the initial value of the hash algorithm is replaced by a non-standard value making the attack possible [46]. Finally, in the rump session of Crypto 2004 it was announced that collisions for MD4, MD5, HAVAL-128 and RIPEMD were found. In 2005, Wang et al. [172, 170, 173, 171] publish several cryptanalytic articles showing that the use of a differential attack can find a collision in MD5 in less than an hour while the same attack applied to MD4 can be performed in less than a fraction of a second. The latest results by Stevens et al. [159] in 2009, present a refined chosen-prefix collision for MD5 which requires only $2^{49}$ calls of the compression function. The same work also improves identical-prefix collisions for MD5 to about $2^{16}$ compression function calls.

The first version of the SHA family, known as SHA-0, was introduced by the American National Institute for Standards and Technology (NIST) in 1993 [5]. This standard is also based on the design principles of the MD4 family. One year after proposing SHA-0, NIST discovered a certificational weakness in the existing algorithm. By introducing a minor change they proposed the new Secure Hash standard known as SHA-1 [6]. The message digest size for both algorithms is 160 bits. The first attack on SHA-0 was published in 1998 by Chabaud and Joux [33] and was probably similar to the classified attack developed earlier (the attack that resulted in the upgrade to SHA-1). This attack shows that a collision in SHA-0 can be found after $2^{61}$ evaluations of the compression function. According to the birthday paradox, a brute force attack would require $2^{80}$ operations on average. In August 2004, Joux et al. [80] first showed a full collision on SHA-0 with a complexity of $2^{51}$ computations. Finally, in 2005 Wang, Yin, and Yu announced the full collision in SHA-0 in just $2^{39}$ hash operations [173] and reported that collision in SHA-1 can be found in complexity less than $2^{69}$ computations [171].

The following generation of SHA algorithms known as the SHA-2 family was introduced by NIST in 2000 and adopted as an ISO standard in 2003 [76]. All three hash algorithms (SHA-256, SHA-384 and SHA-512) have much larger message digest size (256, 384, and 512 bits, respectively). The youngest member of this family is SHA-224 and was introduced in 2004 as a Change Notice to FIPS 180-2 [7].

There are only a few security evaluations of the SHA-2 algorithms so far. The first security analysis was done by Gilbert and Handschuh [56] in 2003 and it showed that neither Chabaud and Joux's attack, nor Dobbertin-style attacks apply to these algorithms. However, they show that slightly simplified versions of the SHA-2 family are surprisingly weak. In the same year Hawkes and Rose [65] announced that second preimage attacks on SHA-256 are much easier than expected. A cryptanalysis of step-reduced SHA-2 is done by Aoki et al. [11], Indesteege et al. [74], Mendel et al. [114], Nikolić et al. [132], and Sanadhya et al. [150, 149, 151, 148]. Although pointing out possible weaknesses of the SHA-2 family, these analyses did not lead to actual attacks so far.

In [167], van Oorschot and Wiener showed that in 1994 a brute-force collision search for a 128-bit hash algorithm could be done in less than a month with a $10 million investment. Nowadays, according to Moore's law, the same attack could be performed in less than two hours. As a countermeasure to this attack, the size of the hash result has to be at least 160 bits. RIPEMD-160 is a hash algorithm with a message digest of 160 bits that was designed by Dobbertin, Bosselaers and Preneel in 1996 [48]. The intention was to make a stronger hash algorithm and replace the existing 128-bit algorithms such as MD4, MD5 and RIPEMD. To the best of our knowledge the only study concerning the security of RIPEMD-160 so far, is published by Rijmen et al. [115]. In this analysis, the authors extend existing approaches using recent results in cryptanalysis of hash algorithms. They show that methods successfully used to attack SHA-1 are not applicable to full RIPEMD-160. Additionally, they use analytical methods to find a collision in a 3-round variant of RIPEMD-160.

In response to the above mentioned attacks on the most popular hash algorithms, NIST launched a worldwide competition for the development of a new hash function. Similar to the development of the present block cipher standard – AES, NIST uses a competition model that has been proven to assure a fair selection among various candidates [127]. The competition is organized in three phases, with the final phase scheduled to complete by the end of 2012. Out of the original 64 submissions to the first phase, fourteen candidates have been selected for detailed analysis in the second phase (Blake, BMW, CubeHash, ECHO, Fugue, Grøstl, Hamsi, Keccak, JH, Luffa, Shabal, SHAvite-3, SIMD, Skein). On December 9, 2010, NIST announced five finalists that advanced to the third, and final round (Blake, Grøstl, JH, Keccak, and Skein).

The selection of the winning candidates is driven by considering security properties as well as implementation efficiency of the proposed hash algorithms both in hardware and software. However, a systematic cryptanalysis of hash functions is not well established. It is hard to measure the cryptographic strength of a hash function beyond obvious metrics such as digest length. For this reason, the implementation efficiency of hardware and software plays a vital role in the selection of the finalist.

## 3.3 Throughput Improving Techniques

In 1997, Bosselaers, Govaerts, and Vandewalle [25] analyzed the MD4-based designs with respect to their performance, especially emphasizing the potential of software parallelism. Their prediction that the future architectures will be able to exploit this potential actually came true with the appearance of the multi-core processors.

A systematic approach for designing the throughput optimal hardware architectures of the same class of hash functions is treated by Lee et al. [104, 105]. The authors show that the implementation of cryptographic hash functions has adapted techniques, more than expected, from the architectures, design methods, and tools of digital signal processing systems.

In this section we outline some of the standard DSP techniques such as *retiming*, *unrolling*, *pipelining*, and combinations of those, for the purpose of high-throughput hardware implementations of cryptographic hash functions.

A typical, somewhat simplified, hardware architecture of a cryptographic hash function is illustrated in Fig. 3.2. The black squares represent the sequential logic while the grey circles represent the combinational logic. The message block M, of size $B$ bits, is first stored in the input register, after which the core function F is iteratively applied $n$ times by updating its state register. Sometimes, the result needs to be processed by a so-called finalization function before it is stored in the output register, after which the final digest H is delivered. In order to simplify this discussion we ignore the eventual presence of the finalization function. Obviously, not all cryptographic hash function will necessarily follow the architecture given in Fig. 3.2. The figure rather serves to illustrate the principal usage of several basic throughput-improving DSP techniques.

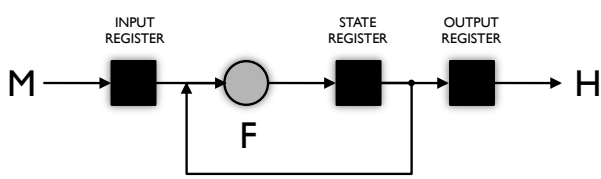


Figure 3.2: Typical hardware architecture of the cryptographic hash function.

The number of times the core function is applied is usually denoted as the number of rounds. Due to the large design space of hardware architectures, we rather use the number of clock cycles as a quantitative measure. We assume that $n$ rounds of the core function are applied to the state register during $I_n$ clock cycles. Furthermore, we recall the definition of the *critical path* as the longest path between any two storage elements. The critical path is therefore inversely proportional to the maximum frequency of the design ($f_{\max}$). The maximum throughput is now

estimated as:

$$T = \frac{f_{\max} B}{I_n} \ .$$

Obviously, in order to increase the overall throughput given the size of the input message block $B$, one needs to increase the maximum frequency of the design and/or to decrease the total number of clock cycles. Next, we outline some of the popular DSP techniques for improving the overall throughput of the design. Note that the techniques are described in their basic form, while more advanced applications depend on the characteristics of the algorithm itself. One such algorithm dependent application is given in Section 3.4, where a so-called *iteration bound analysis* is introduced and specifically used for improving the throughput of the MD4 based cryptographic hash functions. A detailed treatment of the iteration bound analysis can be found in the 'VLSI Digital Signal Processing Systems: Design and Implementation' book by Parhi [133].

### Retiming

Retiming, as first introduced by Leiserson et al. in 1983 [106], is a widely used technique in the design of DSP systems. It is a transformation technique that changes the locations of unit-delay elements (registers) in a circuit without affecting the input/output characteristic of the circuit. Unlike pipelining, retiming does not increase the latency of the system.

By observing Fig. 3.2, it is obvious that the critical path of the design is placed along the core function F, between the input and the state register. As already mentioned, the function F is purely combinational, and therefore the critical path can be shortened by splitting the core function into two functions $\mathrm{F}_1$ and $\mathrm{F}_2$ of a lower complexity, i.e. $\mathrm{F} = \mathrm{F}_2(\mathrm{F}_1)$, and placing the state register in between. This is illustrated in Fig. 3.3. Ideally, the critical path of $\mathrm{F}_1(\mathrm{F}_2)$ should be smaller than the critical path of F.

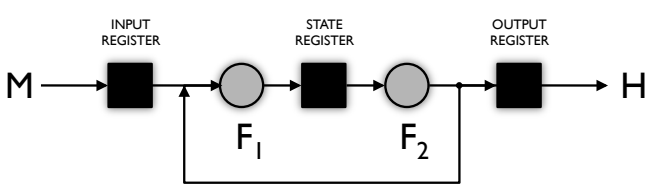


Figure 3.3: Retiming transformation.

Retiming is a technique that is also used in the iteration bound analysis. We discuss more about this method, which turns out to be very suitable for implementing the MD4-based algorithms, as introduced by Lee et al. [104, 105], in Section 3.4.

### Unrolling

Unrolling, sometimes referred to as loop unwinding, is a known transformation technique for improving the speed of an algorithm at the expense of the total design size. Similar to its software counterpart, loop unrolling in hardware reduces the number of cycles necessary for the execution of a certain algorithm. It mainly consists of replications of the loop body corresponding to consecutive iterations.

Assuming that the core function F from Fig. 3.2 needs to be executed $n$ times, an equivalent algorithm, a so-called $k$-unrolled design can be implemented as depicted in Fig. 3.4. The core function F is simply replicated $k$ times and sequentially concatenated such that $\mathrm{F}^{(1)} = \mathrm{F}^{(2)} \ldots = \mathrm{F}^{(k)} = \mathrm{F}$. For the sake of simplicity, we assume that $k|n$. This condition only assures that the output register remains physically connected to $\mathrm{F}^{(k)}$. If $k = n$, we say that the algorithm is fully unrolled. As pointed out by Cardoso and Diniz [30], only deciding whether or how to apply the loop unrolling, either fully or partially, already leads to a large design space exploration problem.

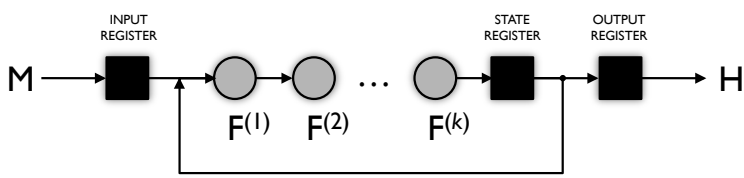


Figure 3.4: Unrolling transformation.

One of the advantages of loop unrolling in hardware lays in the fact that the performance of an unrolled structure can often be further improved, such that the critical path of the $k$-unrolled design is less than $k$ times the critical path of the initial design. Furthermore, this technique in combination with retiming is quite powerful and widely used in iteration bound analysis for obtaining a throughput optimal design.

We illustrate the symbiosis of these two techniques by providing Fig. 3.5, whilst Fig. 3.2 again serves as the start design. Assume that the core function is atomic, i.e. the function F cannot be further decomposed into functions of a lower complexity. Assume further, for the sake of simplicity, that the whole algorithm requires only 2 rounds. A simple $k = 2$-unrolling is first performed resulting in the design given in Fig. 3.5 a. Since the function F is atomic, we can assume that the performance of two concatenated core functions cannot be much improved. Therefore, the resulting design will roughly have the same throughput as the starting one. Now, by simply applying the retiming technique we can place the state register in between the two core functions as illustrated in Fig. 3.5 b and reduce the critical path. Hence,

the overall throughput increases around two times. This symbiosis is especially of interest for MD4-based hash functions where multiple loops are present in the system.

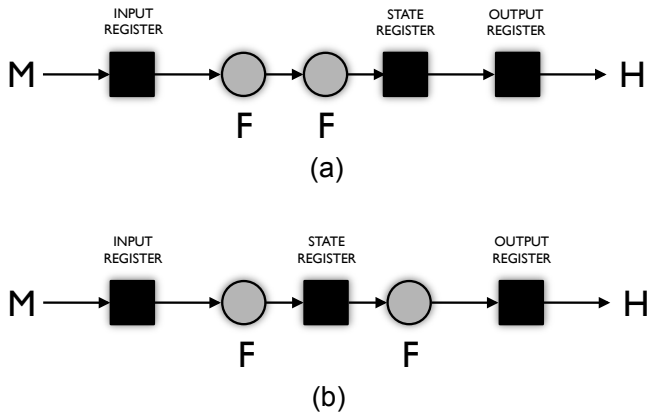


Figure 3.5: (a) $k = 2$-unrolled design. (b) retiming of the unrolled design.

### Pipelining

Before describing the well-known pipelining technique, we need to stress that the majority of cryptographic hash functions are recursive algorithms and therefore the use of pipelining is meaningful only for the scenarios where multiple messages are hashed.

Pipelining is a fundamental technique used in DSP systems for increasing the overall throughput. A basic idea is to split the processing of independent data into a series of steps such that the data is processed in parallel and the result is stored at the end of each step. To illustrate, we provide Fig. 3.6, which represents the pipelined version of the initial design (see Fig. 3.2). Similar to the unrolling, we call this $k$-pipelined design ($k$ represents the level of pipeline) and $\mathrm{F}^{(1)} = \mathrm{F}^{(2)} \ldots = \mathrm{F}^{(k)} = \mathrm{F}$. Obviously, if $k = n$ the design is fully pipelined. Again, for the sake of simplicity, we assume that $k|n$.

Since the core function is replicated $k$ times, the size of the design is proportional to the level of pipeline in this case. Another approach, if applicable, may often provide a better area-time trade-off. Assuming that the core function F can be decomposed into $k$ functions of a lower complexity, i.e. $\mathrm{F} = \mathrm{F}_k(\mathrm{F}_{k-1}(\ldots(\mathrm{F}_1)\ldots)$ we can introduce the pipeline as given in Fig. 3.7. This approach can be considered as a combination of retiming and a classical pipelining.

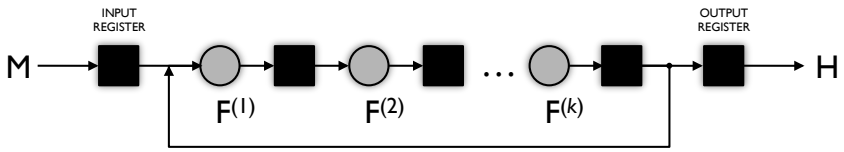


Figure 3.6: Pipelining technique.

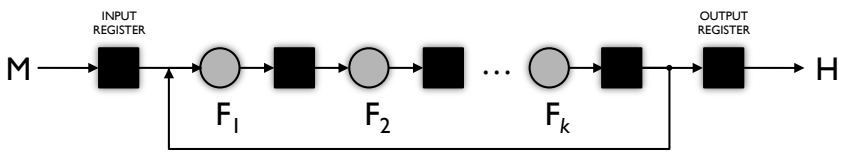


Figure 3.7: Decomposing core function into multiple pipeline stages.

## 3.4 On the High-Throughput Implementation of RIPEMD-160 Hash Algorithm

### Publication Data

M. Knežević, K. Sakiyama, Y. K. Lee, and I. Verbauwhede, "On the High- Throughput Implementation of RIPEMD-160 Hash Algorithm," in *IEEE International Conference on Application-Specific Systems, Architecture and Processors – ASAP 2008*, pp. 85–90, IEEE Computer Society, 2008.

#### Personal Contributions

- Principal author.

Our novel contribution consists of proposing two new high-throughput architectures for the FPGA implementation of the RIPEMD-160 hash algorithm. We apply some of the algorithm-specific transformations and, to the best of our knowledge, achieve the highest throughput of the RIPEMD-160 algorithm reported up to date. Using the Xilinx Virtex2Pro FPGA board we achieve a throughput of 624 Mb/s at the cost of 4410 LUTs. For the sake of completeness we provide ASIC synthesis results as well. Implemented in 0.13 $\mu$m CMOS technology, our design reaches 3.4 Gb/s at the cost of 28,210 GE.

### 3.4.1 RIPEMD-160 Algorithm

**Algorithm 14** RIPEMD-160 algorithm.

$T = rol_s(A + F_i(B, C, D) + X_i + K_i) + E$
$E = D$
$D = rol_{10}(C)$
$C = B$
$B = T$
$A = E$
$T' = rol_{s'}(A' + F'_i(B', C', D') + X'_i + K'_i) + E'$
$E' = D'$
$D' = rol_{10}(C')$
$C' = B'$
$B' = T'$
$A' = E'$

RIPEMD-160 shown in Alg. 14[1] is a hash algorithm that takes an input of arbitrary length (less than $2^{64}$ bits) and produces an output of 160-bit length after performing five independent rounds. Each round is composed of 16 iterations resulting in 80 iterations in total. RIPEMD-160 operates on 512-bit message blocks which are composed of sixteen 32-bit words. The compression function consists of two parallel datapaths as shown in Fig. 3.8.

$F_i$ and $F'_i$ are non-linear functions and $K_i$ and $K'_i$ are fixed constants. Temporary variables A, B, C, D and E for the left and A′, B′, C′, D′ and E′ for the right datapath, are initialized with the five 32-bit chaining variables, $h_0$, $h_1$, $h_2$, $h_3$ and $h_4$ respectively. Chaining variables are either initialized with fixed values to hash the first 512-bit message block or updated with the intermediate hash values for the following message blocks. Each step of the algorithm uses a different message word $X_i$ for the left and $X'_i$ for the right datapath. All the 16 message words are reused for each round but in a different order. For a detailed description of the algorithm please refer to work of Dobbertin et al. [48].

### 3.4.2 Optimization at Micro-Architecture Level

The MD family hash algorithms can be considered as an example of digital signal processing systems. Block diagrams are most frequently used to graphically represent a DSP system. A data flow graph (DFG) is an example of a block diagram where the nodes represent computations (or functions) and directed edges

[1]With + we denote addition modulo $2^{32}$.

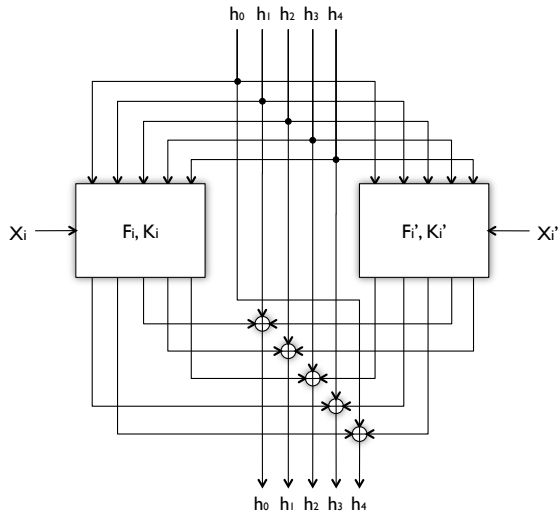


Figure 3.8: Compression function of the RIPEMD-160 algorithm.

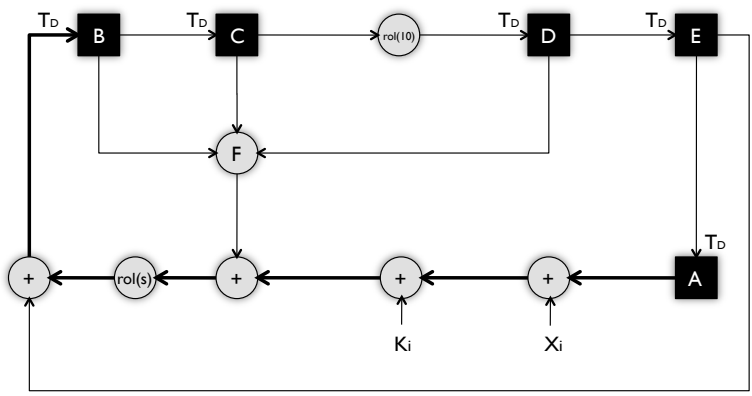


Figure 3.9: Data flow graph for compression function of the RIPEMD-160 algorithm.

represent datapaths. Each edge has a nonnegative number of delays associated with it. These unit-delay elements (often called algorithmic delays) can also be treated as functional blocks as they are implemented using registers. As an example of a DFG we can look at Fig. 3.9, where the functional nodes are represented with the light gray circles and registers are represented with the black squares. One unit-delay $T_D$ is associated to each edge, that is at the input of the register.

The *iteration bound* of the circuit is defined as

$$T_\infty = \max_{l \in L} \left\{ \frac{t_l}{w_l} \right\} \tag{3.1}$$

where $t_l$ is the loop calculation time, $w_l$ is the number of *algorithmic* delays (marked with $\mathrm{T_D}$ in Fig. 3.9) in the $l$-th loop, and $L$ is the set of the all possible loops. A DFG of RIPEMD-160, which is shown in Fig. 3.9, is derived from Alg. 14 and contains five different loops. The iteration bound is determined by the loop $\mathrm{B} \rightarrow \mathrm{F} \rightarrow + \rightarrow rol(s) \rightarrow + \rightarrow \mathrm{B}$ (dashed line) and is equal to

$$T_\infty = 2 \times Delay(+) + Delay(\mathrm{F}) + Delay(rol) \ . \tag{3.2}$$

The critical path of the DFG in Fig. 3.9 is the path marked with the bold lines $\big(4 \times Delay(+) + Delay(rol)\big)$ and it is larger than the iteration bound. Therefore, to achieve a throughput optimal design, we need to apply some transformations on the given DFG.

In [102], Lee et al. apply some DSP techniques on SHA-2 family hash algorithms, such as the iteration bound analysis and the retiming and unrolling transformations. By applying these techniques, an architecture whose critical path is equal to the iteration bound can be derived. In this optimization, the functional operations used in a hash algorithm, e.g. non-linear functions and additions, are assumed to be *atomic*, i.e. a functional operation cannot be split or merged into some other functional operations. In other words, the optimization is limited to the micro-architecture level.

By applying only the retiming transformation, we can obtain a DFG of RIPEMD-160 whose critical path delay is reduced to the iteration bound. Figure 3.10 shows the DFG after retiming transformation and the critical path is again marked with the bold lines. Now the critical path delay is equal to the iteration bound, which means that the DFG given in Fig. 3.10 represents a throughput optimal architecture that achieves a theoretical upper bound at the micro-architecture level. Note here that it is not always possible to achieve the throughput optimal design by using the retiming transformation only. It is sometimes necessary to apply additional techniques such as unrolling for this purpose. A complete methodology concerning this matter can be found in the work of Lee et al. [104, 105].

Due to the retiming transformations, two adders are placed between registers E and A1 now. This causes A1 to be initialized with $\mathrm{h_0} + \mathrm{X_0} + \mathrm{K_0}$ instead of only with $\mathrm{h_0}$. Making the values of $\mathrm{X}_i$ and $\mathrm{K}_i$ equal to zero at the last iteration, A1 becomes equal to A.

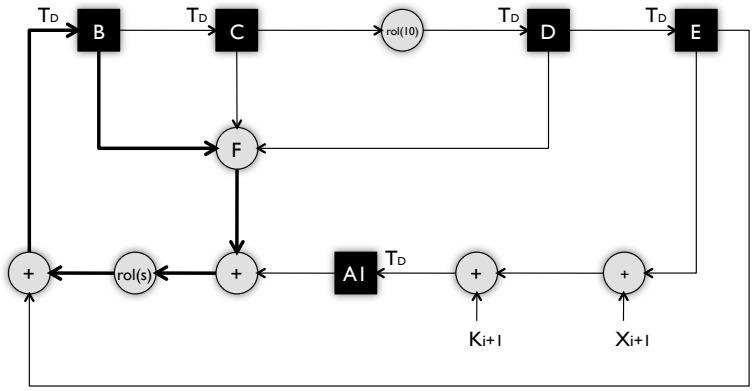


Figure 3.10: Throughput optimized DFG of the RIPEMD-160 algorithm.

In the next section we present gate level optimizations by merging a few functional nodes, which results in an architecture with an even higher throughput.

### 3.4.3 Optimization at Gate Level

Analyzing Fig. 3.10 we notice that a set of functional nodes consists of four modular adders, one non-linear function and two cyclic shifts. As the critical path is in the loop, the only way of optimizing the DFG is to optimize the loop. Unfortunately, the variable cyclic shift is placed between two modular adders and prevents us from using a carry save adder (CSA) instead. However, in this section we show how another approach can be used to further improve the overall performance of the loop.

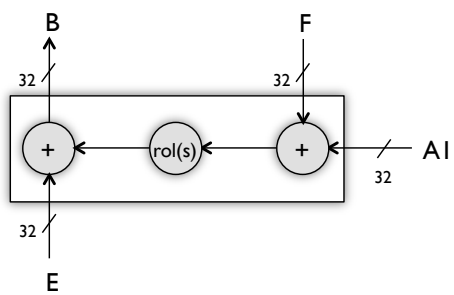


Figure 3.11: ADD+ROT part of the loop in the RIPEMD-160 algorithm.

Let us consider a simple example shown in Fig. 3.11 where the part of the loop with three 32-bit inputs, A1, E and F, and one output, B, is shown. To simplify the discussion we omit the input $s$ which represents the number of bits in the cyclic shift. The functionality of this block is given in Fig. 3.12. After adding two operands A1 and F, the cyclic shift is applied, and the operand E is finally added resulting in the output B. We define $carry\_1$ as the carry bit that may occur in the result of adding $(32 - s)$ least significant bits (LSBs) of A1 and F. This bit will be propagated to the $s$ most significant bits (MSBs) of the sum. Another carry bit ($carry\_2$) may occur in the result of the whole addition and will be discarded before the rotation starts (due to the modular addition).

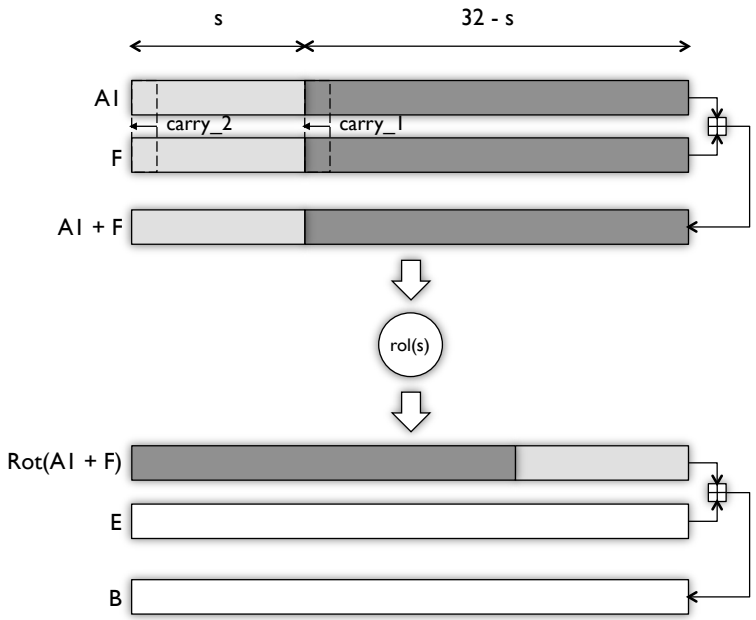


Figure 3.12: Functionality of the original ADD+ROT part in the RIPEMD-160 algorithm.

In order to optimize the given block, we rotate operands A1 and F before adding them together. Doing this we have to take care of the carry bits $carry\_1$ and $carry\_2$. The latter carry must not be propagated after the rotation of the two operands. To prevent this carry propagation we can subtract the vector $\Delta$ from the sum of $Rot$(A1) and $Rot$(F) as shown in Fig. 3.13. The bit value $\delta$ is equal to 1 if $carry\_2 = 1$, otherwise $\delta = 0$. Besides the $carry\_2$, we also need to take care of the $carry\_1$ bit. This carry must be added to the rotated operands $Rot$(A1) and $Rot$(F) (see Fig. 3.13). Depending on the $carry\_1$ and $carry\_2$ we have four

different possibilities. In order to reduce the critical path, we compute all four possibilities in parallel.

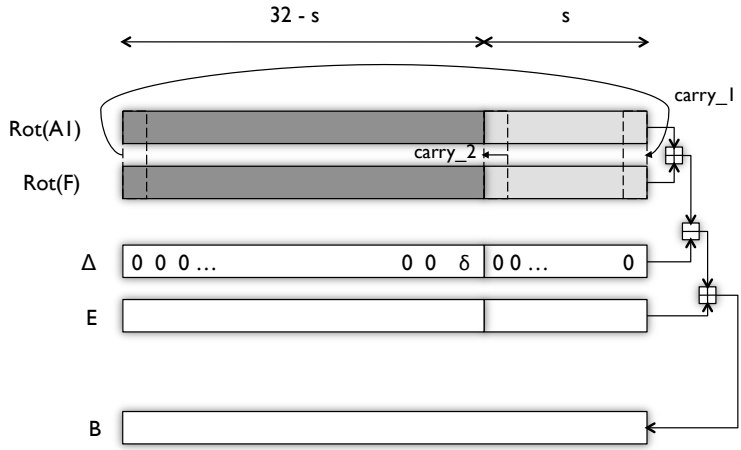


Figure 3.13: Functionality of the ADD+ROT part after the first transformation.

The only drawback of this architecture is that subtraction of $\Delta$ and addition of $carry\_1$ is still executed within the loop which does not decrease the critical path. Since addition is an associative operation we can subtract the vector $\Delta$ from the operand E before entering the loop, as illustrated in Fig. 3.14.

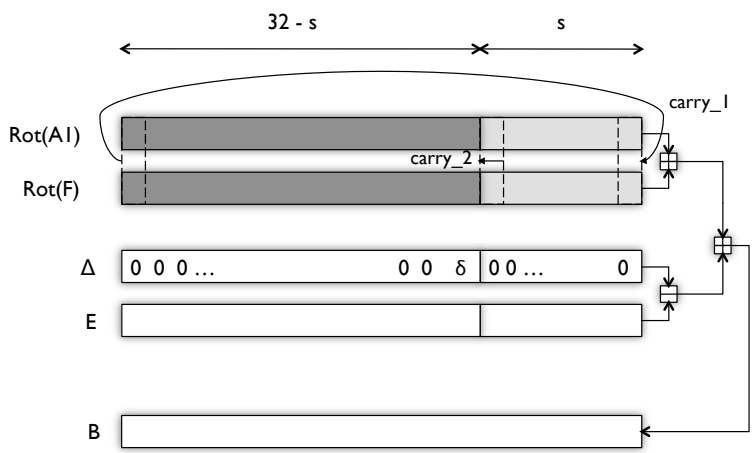


Figure 3.14: Functionality of the ADD+ROT part after optimization.

Using similar design criteria, we add *carry_1* after adding *Rot*(A1), *Rot*(F) and E. Instead of using one additional adder for this operation, we use a CSA and the fact that the *carry form* of CSA is always shifted to the left for one position before the final addition. In this way we just need to change the LSB bit of the shifted *carry form* depending on *carry_1*.

The architecture describing this whole logic is shown in Fig. 3.15. To achieve a high-throughput of the algorithm we prepare all four possibilities in parallel. The values of *carry_1* and *carry_2* determine which input of the MUX will be propagated to the output value B as it is shown in Table 3.1. Note here that input E1 is obtained as $E1 = D - \Delta$ where $\Delta$ is chosen such that $\delta = 1$. Input E represents the case where $\delta = 0$.

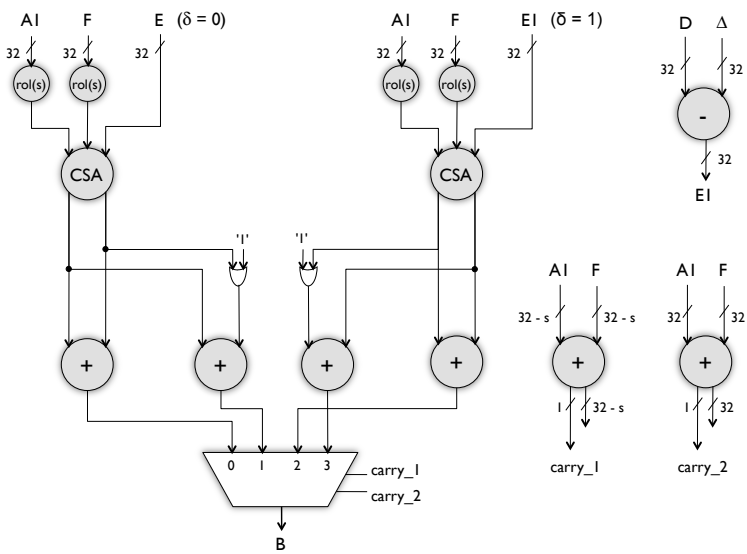


Figure 3.15: Throughput optimized ADD+ROT part of the loop in the RIPEMD-160 algorithm.

### 3.4.4 Final High-Throughput Architecture

Following the design principles described in the previous section and using an additional retiming transformation we can obtain the final high-throughput architecture for the RIPEMD-160 algorithm. The first step is to use the throughput optimized part of the loop instead of the original one (see Fig. 3.15). A DFG that shows this architecture is given in Fig. 3.16. The optimized part of the loop is denoted as a black box (ADD+ROT).

Table 3.1: Selecting the appropriate input of the MUX.

| MUX | $carry\_2$ | $carry\_1$ |
|---|---|---|
| 0 | 0 | 0 |
| 1 | 0 | 1 |
| 2 | 1 | 0 |
| 3 | 1 | 1 |

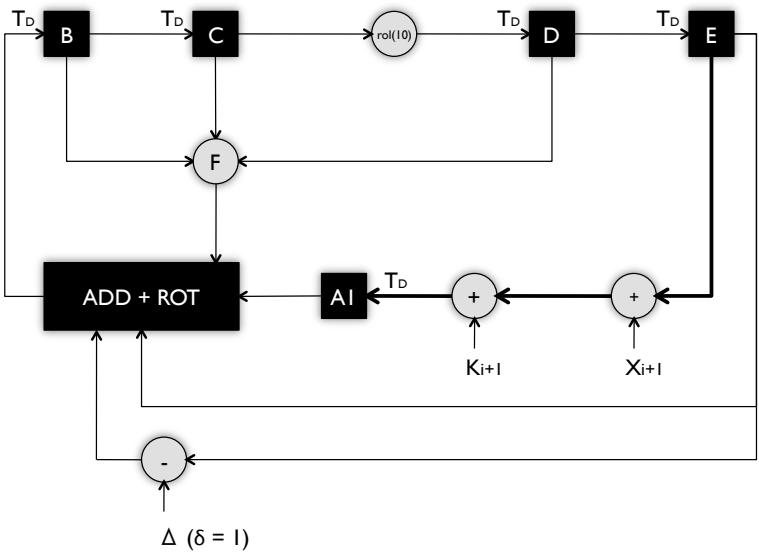


Figure 3.16: DFG of the RIPEMD-160 architecture with optimized ADD+ROT part of the loop.

The fact that we use the optimized part of the loop now moves the critical path between registers E and A1. This problem can be easily solved by using a CSA instead of two adders. Figure 3.17 shows this solution.

As the critical path occurs between the output of E and the input of B (bold line) now, we need to introduce one more register E1 and move the subtractor at its input as shown in Fig. 3.18. In this way the critical path is placed within the loop again and the high-throughput architecture of RIPEMD-160 is finally obtained.

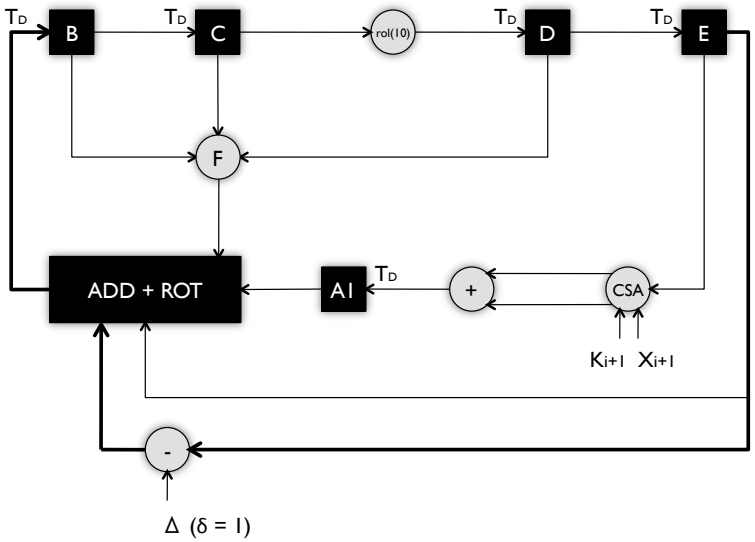


Figure 3.17: Using CSA instead of two adders changes the critical path.

### 3.4.5 Implementation Results and Comparison with Previous Work

In order to verify the speed of our architectures we have implemented the proposed solutions using GEZEL [154]. Both implementations were verified on a Xilinx Virtex2Pro FPGA board. Our results and comparison with previous work are given in Table 3.2.

Sklavos and Koufopavlou [156] propose a RIPEMD processor that performs both RIPEMD-128 and RIPEMD-160 hash algorithms and they separately implement a RIPEMD-160 processor for comparison purposes. In order to achieve the high throughput of 2.1 Gb/s the authors use a pipelining technique. However, due to the recursive mode of operation of the RIPEMD-160 algorithm, using pipelining is only possible for hashing independent messages. Our architecture with the optimized loop can easily be pipelined and for this special case we achieve a throughput of 3.122 Gb/s. Pipelining is done by replicating a DFG of the RIPEMD-160 algorithm five times, one for each of the five different nonlinear functions. However, as this comes at high price of occupied area and is only useful in limited numbers of applications, we do not provide more details about the pipelined implementation.

Another issue we would like to point out here is again related to the result of Sklavos and Koufopavlou [156]. The authors report a hardware evaluation of a

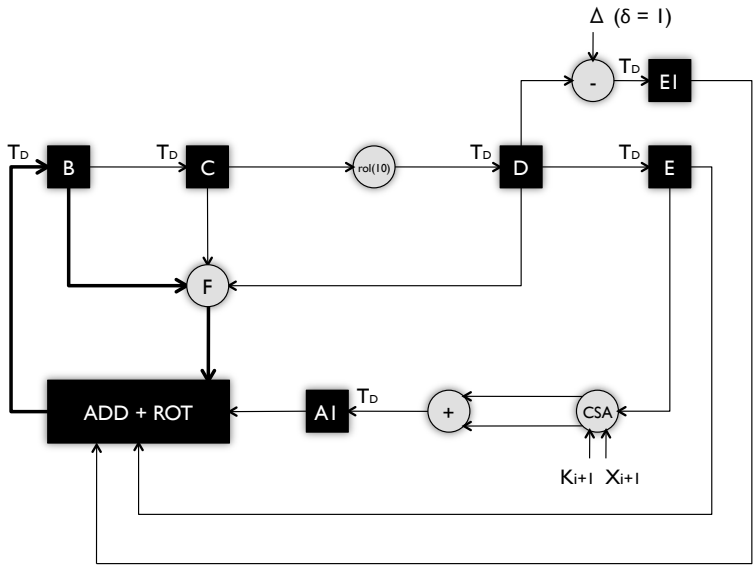


Figure 3.18: Throughput optimized DFG of the RIPEMD-160 algorithm with optimized ADD+ROT part.

RIPEMD processor on a Xilinx XC2V250 device and report an area consumption of 2014 CLBs. However, the mentioned device consists of only 1,536 slices, i.e. 384 CLBs (see Virtex-II Platform FPGA User Guide [178]). In the same paper they also report the use of a Xilinx v200bg256 device which is another series of Xilinx devices and comprises of 5,292 Logic Cells, i.e. 1176 CLBs (see Virtex 2.5V FPGA Complete Data Sheet [177]). Therefore, we are not able to provide a completely fair comparison, and we decide to use a Xilinx Virtex2Pro (XC2VP30) device for the purpose of our evaluation.

Here, we can also notice that the speed of the architecture with optimized ADD+ROT part is 10 % faster than the version without. On the other hand the size is 65 % larger due to parallel processing shown in Fig. 3.39 and the use of one additional register (see Fig. 3.18).

Finally, for the sake of completeness, we have also synthesized our design using UMC 0.13 $\mu$m CMOS process and Synopsys Design Compiler version C-2009.06-SP3. The fastest version reaches a throughput of 3.4 Gb/s, requiring 28,210 GE.

Table 3.2: Implementation results and comparison with previous work.

| Design | FPGA chip | Frequency [MHz] | Number of cycles | Throughput [Mb/s] | Size |
|---|---|---|---|---|---|
| [131](1) | EPF10K50 | 26.66 | 162 | 84 | 1964 LC |
| [49](2) | XCV300E | 42.90 | 337 | 65 | 2008 LUT |
| [86](3) | XC2V4000 | 43.47 | 162 | 137.4 | 14,911 LUT |
| [156](4) | XC2V250(5) | 73 | 82 | 455 | 2014 CLB(5) |
| Fig. 3.10 | XC2VP30 | 90.97 | 82 | 568 | 2721 LUT |
| Fig. 3.18 | XC2VP30 | 100.05 | 82 | 624 | 4410 LUT |

(1) This is a unified architecture of MD5 and RIPEMD-160 hash algorithms.

(2) This is a unified architecture of MD5, RIPEMD-160, SHA-1 and SHA-256 hash algorithms.

(3) This is a unified architecture of MD5, RIPEMD-160 and SHA-1 hash algorithms.

(4) In the original paper throughput of 2.1 Gb/s is shown for hashing 5 independent messages. To make a fair comparison we consider throughput for a single message only.

(5) In the original paper the use of 2014 CLBs, 4006 FGs and 1600 DFFs is reported. However, XC2V250 device contains only 384 CLBs [178] and therefore we assume that the reported results are obtained using another Xilinx device.

### 3.4.6 Summary

We showed how the iteration bound analysis can be used for the high-throughput implementation of the RIPEMD-160 hash algorithm. Since the iteration bound is a theoretical minimum of the critical path, there is no further throughput optimization at the micro-architecture level. Thus, we further improve the performance of our architecture at the gate level, achieving the final high-throughput implementation of the RIPEMD-160 algorithm. This approach can be a guideline for a high-throughput implementation of other popular hash algorithms.

## 3.5 Extensive Hardware Comparison of Fourteen Second-Round SHA-3 Candidates

### Publication Data

M. Knežević, K. Kobayashi, J. Ikegami, S. Matsuo, A. Satoh, Ü. Kocabaş, J. Fan, T. Katashita, T. Sugawara, K. Sakiyama, I. Verbauwhede, K. Ohta, N. Homma, and T. Aoki, "Fair and Consistent Hardware Evaluation of Fourteen Round Two SHA-3 Candidates," *IEEE Transactions on VLSI*, 13 pages, 2011. To appear.

### Personal Contributions

- Involved in: Planning; Implementation; Evaluation; Data analysis; Text writing.

Our novel contribution consists of proposing a platform, a design strategy and evaluation criteria for a fair and consistent hardware evaluation of the second-round SHA-3 candidates. Using a SASEBO-GII FPGA board as a common platform, combined with well defined hardware and software interfaces, we compare all 256-bit version candidates with respect to area, throughput, latency, power and energy consumption.

Our approach defines a standard testing harness for SHA-3 candidates, including the interface specification for the SHA-3 module on our testing platform. The second contribution is that we provide both FPGA and 90 nm CMOS ASIC synthesis results and thereby are able to compare the results. Our third contribution is that we release the source code of all the candidates and by using a common, fixed, publicly available platform, our claimed results become reproducible and open for public verification.

The performance evaluation of hardware, including the measurement of the power consumption, execution time, and hardware resources, is a rather complex problem. There are several reasons for this. Most importantly, the design space for hardware performance evaluation is larger than that of software. Additional design constraints (such as low-area, max-throughput, and min-energy) are required to define an optimal implementation. Second, accurate and generic performance evaluation metrics are hard to obtain. A throughput can be characterized provided that the hardware design can be accurately timed. The area metrics depend strongly on the target technology (ASIC/FPGA). A measurement of the power consumption is the most difficult, and is almost never mentioned in the literature.

In this section we address most of these issues and therefore, we summarize our contributions as follows:

- First, we propose a platform, a design strategy, and evaluation criteria for a fair and consistent hardware evaluation of the SHA-3 candidates.
- Second, we use a prototyping approach by mapping each of the 256-bit version hash candidates onto a SASEBO-GII FPGA board [125]. The hash candidates are then evaluated with respect to throughput, latency, hardware cost, and power and energy consumption.
- Third, we provide synthesis results in 90 nm CMOS technology with respect to throughput and circuit size. In addition, we provide power and energy consumption estimates.

- Finally, by releasing the source code of all the candidates and by using a common, fixed, publicly available platform, our claimed results become reproducible and open for public verification.

### 3.5.1 Related Work

Recently, several research groups have proposed comprehensive performance evaluation methods, which evaluate a set of hash algorithms on a common platform.

- Tillich et al. [164] developed RTL VHDL/Verilog code for all SHA-3 candidates. They present synthesis results in 180 nm CMOS technology. In order to reach the highest degree of accuracy, they further perform the place & route for the best versions of all fourteen candidates [165].
- Gaj et al. [54] developed a scripting system called ATHENa, targeted towards FPGA. A fair comparison is achieved by defining a standard interface and by automatic design space exploration. Furthermore, in [53] they report a comparison of all 512-bit version SHA-3 candidates using the same methodology.
- Baldwin et al. [16] propose a standard interface to achieve a fair comparison and illustrate their approach by providing the hardware figures for all fourteen SHA-3 candidates. They evaluate hardware designs and test for all message digest sizes (224, 256, 384, and 512 bits) and also include the padding as part of the hardware for the SHA-3 hash functions.
- Henzen et al. [69] evaluated all fourteen second-round SHA-3 candidates using 90 nm CMOS technology. All designs were placed & routed and the post-layout figures were reported.
- Guo et al. [60] presented post place and route figures for all fourteen candidates in 130 nm CMOS technology.

### 3.5.2 General Requirements for Hardware Evaluation

In this section, we reconsider the main requirements for conducting a fair and consistent hardware evaluation of the fourteen SHA-3 candidates.

First, we comment on the feasibility of compact implementations. Second, we discuss the speed performance metrics and power/energy consumption. Then, we open a question concerning fair comparison and consistent hardware evaluation of the remaining SHA-3 candidates. Finally, we present an attempt to classify the candidates with respect to their design properties. This classification will be useful, later on, for drawing some conclusions and comparing different candidates.

Table 3.3: Memory requirements for the SHA-3 candidates.

| Candidate | State Size [bit] | Total Memory [72] [bit] | Total Memory† [GE] | Total Area [GE] |
|---|---|---|---|---|
| Blake | 512 | 768 | 4,608 | 13,560 [68] |
| BMW | 512 | 1,536 | 9,216 | N/A‡ |
| CubeHash | 1,024 | 1,024 | 6,144 | 7,630 [19] |
| ECHO | 2,048 | 2,560 | 15,360 | 82,800 [110] |
| Fugue | 960 | 960 | 5,760 | 59,220 [61] |
| Grøstl | 512 | 1,024 | 6,144 | 14,620 [163] |
| Hamsi | 512 | 768 | 4,608 | N/A‡ |
| JH | 1,024 | 1,024 | 6,144 | N/A‡ |
| Keccak | 1,600 | 1,600 | 9,600 | N/A‡ |
| Luffa | 768 | 768 | 4,608 | 10,340 [119] |
| Shabal | 1,408 | 1,408 | 8,448 | 23,320 [19] |
| SHAvite-3 | 896 | 1,024 | 6,144 | N/A‡ |
| SIMD | 512 | 3,072 | 18,432 | N/A‡ |
| Skein | 512 | 768 | 4,608 | N/A |

Estimates for versions with 256-bit digest size are given.

† We estimate the size of a single flip-flop to be 6 GE.

‡ To the best of our knowledge, as of March 2011, these candidates had no published figures for low-cost hardware implementations.

### Area: Lower Bound on Compact Implementations

Depending on the application scenarios, one of the decision points, prior to starting with the hardware evaluation, is a choice of the actual architecture. Therefore, we provide a lower bound estimation on each of the fourteen candidates and argue that, given the required security margins, there are no candidates suitable for a lightweight implementation. Our estimation is simply based on the minimum amount of total memory needed for a certain algorithm. We define the state size to be the size of the chaining variable (see Table 3.3). We also refer to the work of Ideguchi et al. [72], that studies the RAM requirements of various SHA-3 candidates for the low-cost 8-bit CPUs. Furthermore, we estimate the size of the required memory with respect to the number of gate equivalences (GE), which represents the lower bound size. Finally, we provide figures for current, compact implementations of some of the second-round candidates.

Comparing the lower bound size of all fourteen candidates with the size of state of the art lightweight block ciphers, e.g. PRESENT [22] and KATAN &

KTANTAN [27], we conclude that all candidates are rather suited for a so-called *welterweight* category. Therefore, in this work, we focus only on the high-throughput variants of all second-round candidates.

### Speed: Latency versus Throughput

Regarding the speed of a hash candidate, we distinguish two performance figures. Depending whether the input message is a long (we consider very long messages in this case) or a short one (e.g. 256 bits or less), we evaluate the throughput and the latency, respectively. The throughput is defined as the amount of information processed per unit of time (bits/s), while the latency represents the time delay necessary for processing a certain amount of information from start to end (s).

This approach provides a fair comparison and an accurate evaluation for each of the candidates. In both cases, the speed performance is a function of several factors: maximum frequency, number of clock cycles necessary for a hash operation, number of cycles necessary for input and output, and the input block size. Furthermore, the latency also depends on the message size and the presence of the finalization function. Later, in Section 3.5.3, we provide formulae that support the previous discussion.

### Power versus Energy

The power consumption of a hash design is measured during a complete hash operation. The total power consumption can be seen as the sum of the static and the dynamic power dissipation. The energy cost is therefore the integral of the power consumption over the period of a hash operation. In order to obtain a standardized $nJ/bit$ metric, the energy cost is normalized to the input block size and to the message length for long and short messages, respectively.

### Fair Comparison

An important requirement for an open competition such as the SHA-3 competition is a fair comparison. To achieve this goal, we need to consider the following two aspects. First, the evaluation environment needs to be open and available to all designers and evaluators. It also needs to be unified and common for all the candidates. Second, the claimed results need to be reproducible and open for public verification. By using a common, fixed platform and making our code publicly available, we achieve the desired goal.

### Classification of Candidates

Another interesting issue to consider is the great diversity of all the second-round candidates. Therefore, we first classify all the algorithms with respect to their design properties. Figure 3.19 represents such a classification.

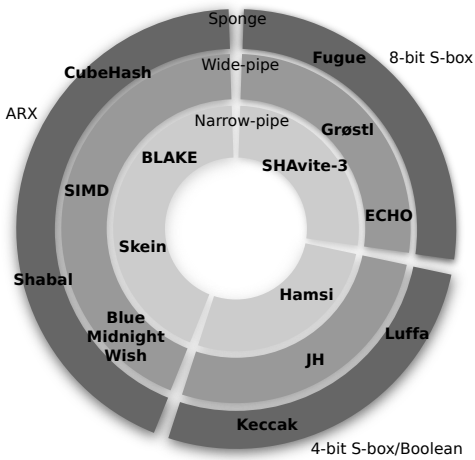


Figure 3.19: Second-round SHA-3 candidates classified with respect to their design properties (courtesy of Dai Watanabe from Hitachi Ltd, the designer of Luffa hash function).

With respect to the main source of non-linearity used in a design, all fourteen candidates can be classified into three main groups, as indicated by the three parts of the pie.

- 8-bit S-box based: ECHO, Fugue, Grøstl, SHAvite-3.
- 4-bit S-box/Boolean based: Hamsi, JH, Keccak, Luffa.
- Addition Rotation XOR (ARX) based: Blake, BMW, CubeHash, Shabal, SIMD, Skein.

Another classification by comparing the size of the compression function to the digest size and the input block size is possible, as indicated by the concentric circuits on the pie. If the output length of the intermediate compression function is equal to the digest size, the structure is called a *narrow-pipe*. The candidates with the output length of the compression function larger than the final hash length are classified as *wide-pipe*. Finally, the candidates whose compression function size

and digest size are fixed, and whose input block size is determined by considering a trade-off between security and efficiency are called the *sponge* constructions. Therefore, depending on the size of the compression function, the candidates can again be classified into three subgroups.

- Narrow-pipe: Blake, Hamsi, SHAvite-3, Skein.
- Wide-pipe: BMW, ECHO, Grøstl, JH, SIMD.
- Sponge: CubeHash, Fugue, Keccak, Luffa, Shabal.

Finally, we classify the candidates with respect to their input block size.

- 32-bit: Fugue, Hamsi.
- 256-bit: CubeHash, Luffa.
- 512-bit: Blake, BMW, Grøstl, JH, Shabal, SHAvite-3, SIMD, Skein.
- 1024-bit: Keccak.
- 1536-bit: ECHO.

Another classification, with respect to the number of cycles necessary for performing the hash operation, is also possible but would highly depend on the implementation strategy. Therefore we do not consider it at this time. However, this observation becomes interesting later, in Section 3.5.4, where the implementation results are discussed in detail. Next, we discuss our proposed evaluation scheme. We describe the evaluation environment, hardware/software interface, design strategy, evaluation metrics and finally, we provide the experimental results.

### 3.5.3 Hardware Evaluation Platform for SHA-3 Candidates

Figure 3.20 illustrates the target platform for our evaluation, which includes a SASEBO-GII board, a PC and an oscilloscope. The SASEBO board includes two FPGAs: a control FPGA and a cryptographic FPGA. On the PC, a test program enables a user to enter a sample message, which is transmitted to the control FPGA through a USB interface. The control FPGA controls the data flow to send this message to the cryptographic FPGA, where hash operations are performed. After the hash operation is done, the digest is returned to the PC through the control FPGA. As illustrated in Fig. 3.20, the interface between the control FPGA and the cryptographic FPGA is fixed and common among all SHA-3 candidates.

The control FPGA checks the latency of a single hash operation that is performed on the cryptographic FPGA and reports the number of clock cycles to the PC. The

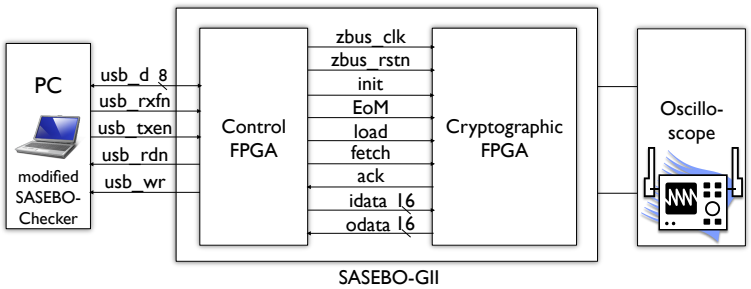


Figure 3.20: Evaluation environment using SASEBO-GII board.

PC then reports two different performance metrics. One is the number of clock cycles including the interface overhead while the other one is excluding the cycles for the data input and output.

During message hashing, we also measure the power consumption of the hashing operation. This trace, in combination with the performance data, enables a precise characterization of the power dissipation and energy consumption of the SHA-3 candidate on the cryptographic FPGA.

### Hardware and Software Interface

A key concept in our approach is the use of a standard interface to integrate the hash algorithms inside the cryptographic FPGA. In this section, we describe the major principles of this interface. We also compare our ideas with those of several other proposals, including the interfaces defined by Chen et al. [180], by Gaj et al. [32], and by Baldwin et al. [15].

In the following observations, it is useful to refer to the method used to interface SHA-3 candidates in software. For that purpose, the software implementations use an Application Program Interface (API) defined by NIST [128]. Three function calls are used:

- `void init(hashstate *d)` initializes the algorithm state of the hash, which is typically stored in a separate structure in order to make the hash implementation re-entrant.
- `void update(hashstate *d, message *s)` hashes a message of a given length and updates the hash state. The message is chopped into pieces of a standard length called a *block*. In case the message length is not an integral

number of blocks, the API will use a *padding* procedure which extends the message until it reaches an integral number of blocks in length.

- `void finalize(hashstate *d, digest *t)` extracts the actual digest from the hash state.

A hardware interface for a SHA-3 module emulates a similar functionality as the software API interface. The hardware interface therefore needs to address the following issues.

**Handshake protocol:** The hash interface needs to synchronize data transfer between the SHA-3 module and the environment. This is done by using a handshake protocol and one can distinguish a *master* and a *slave* protocol, depending on which party takes the initiative to establish the synchronization. The interface by Chen [180] uses a slave protocol for the input and output of the algorithm. The interfaces by Baldwin [15] and Gaj [32] define a slave protocol for the input and a master protocol for the output. The former type of interface is suited for a co-processor in an embedded platform, while the latter one is suited for high-throughput applications that would integrate the SHA-3 module using First Input First Output (FIFO) buffers. The interface in our proposal uses a slave protocol.

**Wordlength:** Typical block and digest lengths are wider (e.g. 512 bits) than the word length that can be provided by the standard platforms (e.g. 32 bits). Therefore, each hash operation will result in several data transfers. While this overhead is typically ignored by hardware designers, it is inherently part of the integration effort of the SHA-3 module. In our proposal, we use a 16-bit interface, which size is driven by the size of the data-bus shared among the control FPGA and the cryptographic FPGA.

**Control:** The functions of the software API need to be translated to the equivalent control signals in hardware. One approach, followed by Gaj [32], is to integrate this control as in-band data in the data stream. A second approach is to define additional control signals on the interface, for example to indicate the message start and end. This is the approach taken by Chen [180] and Baldwin [15]. We follow the same approach in our proposal as well.

**Padding:** Finally, *padding* may or may not be included in the SHA-3 hardware module. In the latter case, the hardware module implicitly assumes that an integer number of blocks will be provided for each digest. Common padding schemes are defined by in-band data formatting, and this makes it possible to implement the padding outside of the hardware module. The interface proposal by Baldwin [15] explicitly places the padding hardware into the interface. The other interface proposals leave the padding to be done outside of the hardware module. However, Chen [180] assumes that the hardware padding will only be implemented at the word-level, while Gaj [32] supports bit-level padding as well. We follow the approach of Chen.

Note that there are many solutions to the interface issue, and that we present only one approach. We also observe that the key issue for a fair comparison is to use a *common* interface for all the candidates. In addition, and that is very important, we show that our performance evaluation mechanism allows to factor out the overhead of the interface communication.

### Design Strategy

Besides a standard platform, our approach also defines a design strategy. As classified by Schaumont et al. [153] there are three types of cores that can be distinguished with respect to their implementation scope (register mapped, memory mapped and network mapped). Similar to this approach, Tillich [162] proposes the following classification:

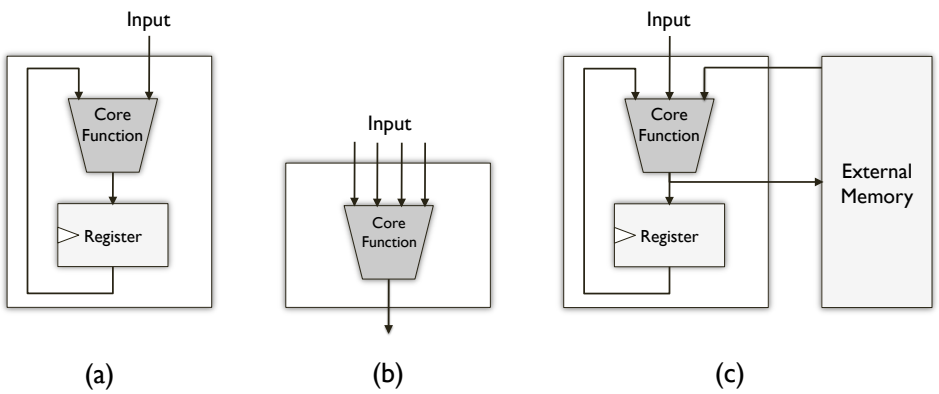


Figure 3.21: Three types of architectures: (a) fully autonomous. (b) core functionality. (c) with external memory.

- *Fully Autonomous Implementation* (Fig. 3.21a): Equivalent to a register mapped implementation proposed by Schaumont et al. [153]. In this architecture, one transfers the message data to a hash function over multiple clock cycles, until a complete message block is provided. The hash module buffers a complete message block locally, before initializing the hash operation. Therefore, this architecture can work autonomously, and the resulting hash module is well suited for the integration into other architectures (e.g. System-on-Chip).
- *Implementation of the Core Functionality* (Fig. 3.21b): This architecture has only the core part of a hash function, and ignores the storage of a full

message block. In other words, this architecture ignores the influence of a fixed interface on the total hardware performance.

- *Implementation with External Memory* (Fig. 3.21c): Equivalent to a memory mapped implementation proposed by Schaumont et al. [153]. In this architecture, only data necessary for executing the hashing calculation is stored in registers. Other data (e.g. intermediate values) is stored in the external memory. In general, the external memory is less expensive than the register based memory. Therefore, the architecture becomes a low-cost implementation. However, this architecture requires additional clock cycles for accessing the external memory, and therefore it is not suitable for high-throughput implementations.

In this work, we choose the Fully Autonomous architecture.

Previous work for evaluating hardware performance has been realized without using a standardized architecture, i.e. different architectures are used. For example, the design method by Namin et al. [124] and Baldwin et al. [14] are based on the core functionality type and they provide a rough estimate of the performance of hash function hardware. On the other hand, the design method by Tillich et al. [164] and Jungk et al. [13] is based on the fully autonomous type. They assume the presence of an *ideal* interface such that the input data for the hash function hardware is sent in a single cycle. Consequently, their evaluation results cannot be used directly for a performance evaluation of an accelerator of a CPU where only a limited amount of data is available in one clock cycle.

In this work, we estimate the influence of the standard hardware interface on each of the fourteen candidates. Our choice of a 16-bit data width is driven by the specification of the common evaluation platform, i.e. SASEBO-GII board. In addition, we provide evaluation metrics that allow us to estimate the hardware performance for an arbitrary data width as well. One can easily obtain the figures by taking into account the highest achievable frequency and the input block size of each of the candidates. Furthermore, we provide the hardware figures by factoring out the overhead introduced by the standard interface.

Figure 3.22 shows the detailed architecture of the cryptographic FPGA which we use for evaluating hardware performance. The cryptographic FPGA consists of an interface block which controls input and output, and a core function block which executes a hashing process. There are several SHA-3 candidates which need to keep an input message during the hashing process. In our environment, we use a message register file for that purpose.

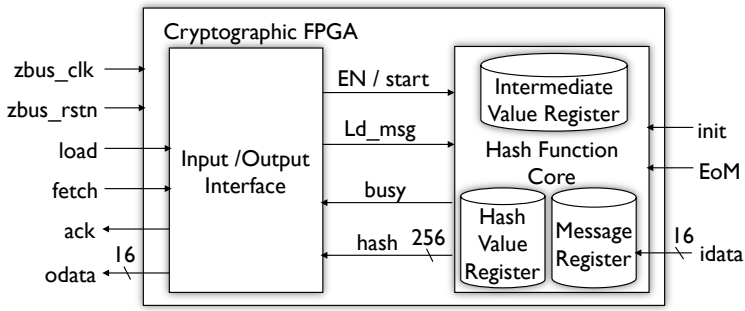


Figure 3.22: Architecture of cryptographic FPGA.

### Platform Specific Evaluation Topics

We implement fourteen SHA-3 candidates on the cryptographic FPGA, Xilinx Virtex-5 (xc5vlx30-3ff324) placed on the SASEBO-GII evaluation board. We check the hardware performance in terms of speed and hardware cost. The speed performance is evaluated by calculating latency or throughput, depending on the message length. It is calculated using the input block size, the maximum clock frequency, and the total number of clock cycles with or without the communication overhead. The cost performance is evaluated with the number of slices, registers, and LUTs for FPGA and the number of gate equivalences for ASIC. A design that has a high throughput with a low hardware cost is regarded as efficient. The power consumption of a hash design is measured during a complete hash operation. The energy cost is therefore the integral of the power consumption over the period of a hash operation. In order to obtain a standardized $nJ/bit$ metric, the energy cost is normalized with respect to the input block size and to the message length for long and short messages respectively.

In order to make the following discussion easier we introduce notations that are used further in this section.

A hash function executes a hashing process for each data block of input block size, and uses the result as a chaining value for the next input data block to perform the whole hashing process. The number of clock cycles needed for hashing $M$ bits of data can be expressed as

$$I = \frac{M_p}{B}(I_{in} + I_{core}) + I_{final} + I_{out} \ . \tag{3.3}$$

Here, $\frac{M_p}{B}$ is the number of hash core invocations where the hash core processes a $B$-bit data block per single invocation. Note that the coefficients of $I_{final}$ and $I_{out}$

$B$ : Input block size,
$w$ : Word size (interface data width),
$I$ : Total number of clock cycles,
$I_{in}$ : Number of clock cycles for loading one message block,
$I_{out}$ : Number of clock cycles for outputting the message digest,
$I_{core}$ : Number of clock cycles for completing the hash process,
$I_{final}$ : Number of clock cycles for the finalization,
$I_w$ : Number of clock cycles for transmitting one word of data,
$f_{max}$ : Maximum clock frequency,
$T$ : Throughput,
$L$ : Latency,
$M$ : Size of the message without padding,
$M_p$ : Size of the message with padding,
$H$ : Size of the message digest (hash output).

are both equal to one, since these processes are only executed when outputting the final message digest. The number of clock cycles needed for the input of the message block and the output of the hash result can be evaluated as

$$I_{in} = \frac{B}{w} I_w \ ,$$

$$I_{out} = \frac{H}{w} I_w \ . \tag{3.4}$$

In our specific protocol, we use $w = 16$ bits and $I_w = 3$ cycles. The former is driven by the evaluation platform specification, while the latter is a result of a simple acknowledgement-based protocol. As a result, the final throughput can be expressed as

$$T = \frac{M_p f_{max}}{\frac{M_p}{B}\left(I_{in} + I_{core}\right) + I_{final} + I_{out}} \ . \tag{3.5}$$

It is also useful to estimate the throughput of the core function only, by factoring out the interface part. Therefore, we write

$$T_{Core} = \frac{M_p f_{max}}{\frac{M_p}{B} I_{core} + I_{final}} \ . \tag{3.6}$$

When $M_p$ is sufficiently large, for example in the case of hashing a long message, $I_{final}$ and $I_{out}$ are negligible in Eq. 3.5 and Eq. 3.6. In this case, the throughput

is approximated as

$$T_{LongMessage} = \frac{Bf_{max}}{I_{in} + I_{core}} \ ,$$
$$T_{LongMessageCore} = \frac{Bf_{max}}{I_{core}} \ . \tag{3.7}$$

On the other hand, when $M_p$ is small, for example in the case of hashing a short message for authentication, we cannot ignore $I_{final}$ and $I_{out}$. Moreover, as the latency is an important metric for a short message (rather than the throughput), we use Eq. 3.8 to compare the speed performance of the SHA-3 candidates.

$$L = \frac{M_p}{T} \ ,$$
$$L_{Core} = \frac{M_p}{T_{Core}} \ . \tag{3.8}$$

Finally, we calculate power and normalized energy per bit consumption for both short and long messages. By $P_U$ and $P_F$ we denote the power consumption during the update and the final phase, respectively, and by $f$ we denote the operating frequency.

$$P_{ShortMessage} = \frac{\frac{M_p}{B} I_{core} P_U + I_{final} P_F}{\frac{M_p}{B} I_{core} + I_{final}} \ ,$$
$$E_{ShortMessage} = \frac{\frac{M_p}{B} I_{core} P_U + I_{final} P_F}{Mf} \ ,$$
$$P_{LongMessage} = P_U \ ,$$
$$E_{LongMessage} = \frac{P_U I_{core}}{Bf} \ . \tag{3.9}$$

### 3.5.4 FPGA Evaluation Results

We implement SHA-256 and all fourteen SHA-3 candidates aiming at high-throughput hardware implementations[2]. Although it is not possible to completely factor out the designer's influence in our comparison, all fifteen algorithms were

[2] We release the Verilog/VHDL source code for these 15 algorithms at `http://www.rcis.aist.go.jp/special/SASEBO/SHA3-en.html`.

prototyped and tested using the same evaluation platform. Each of them was evaluated according to the metrics indicated above, comparing speed performance, area, power consumption and energy consumption.

Table 3.4 and Table 3.5 show a comprehensive summary of the measurement results. **Bold** and gray data represent the best and the worst result in its class, respectively. As with all measurement data, it is important to understand the assumptions used when collecting these numbers. The tables include the following quantities for each candidate.

- The input message block size in bits;
- The highest clock frequency achievable on the Virtex-5 FPGA (xc5vlx30-3ff324) in MHz.
- The latency in terms of clock cycles. Several cases are shown: the cycle count of the input interface overhead ($I_{in}$); the cycle count of the output interface overhead ($I_{out}$); the cycle count of the core function ($I_{core}$); and the cycle count of the final processing ($I_{final}$). All mentioned measures are defined in Section 3.5.3.
- The throughput of the design in Mb/s. This value is calculated assuming that the FPGA is operating at the maximum achievable clock frequency for the given design. Both the throughput with ($T$) and without ($T_{Core}$) interface overhead is shown.
- The latency of the design for short messages in $\mu$s. This value is calculated assuming that the FPGA is operating at the maximum achievable clock frequency for the given design. Both the latency with ($L$) and without ($L_{Core}$) interface overhead is shown. We choose the size of a short message to be 256 bits prior to padding.
- The area cost of the design, in terms of occupied Virtex-5 slices, number of slice registers, and number of slice LUTs. The number of occupied slices provides the primary area measure in this case, while the numbers of slice registers and slice LUTs illustrate the actual utilization of the occupied slices.
- The power consumption of the design for long and short messages. For long messages, the average power consumption includes only the core functionality. For short messages, the average power consumption includes the core functionality and the finalization. The power consumption is measured directly on the core power supply of the FPGA. The power consumption is measured with the FPGA operating at 24 MHz which is the default operating frequency of the board.
- The energy consumption of the design for long and short messages. The energy consumption is normalized with the input block size and the message length

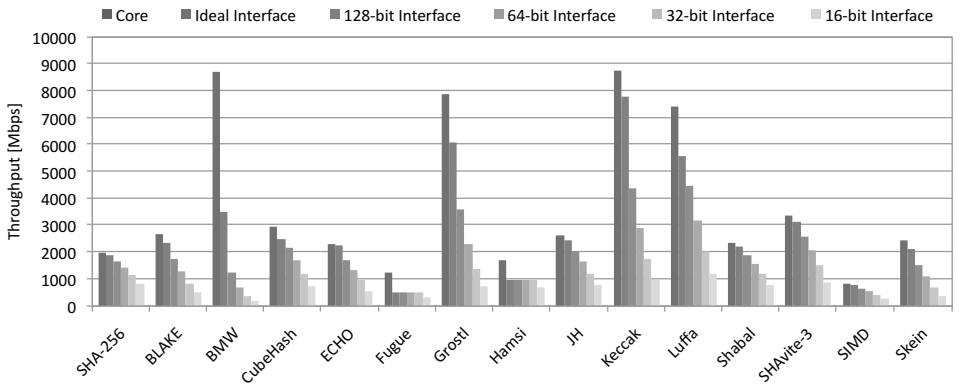


Figure 3.23: Maximum throughput for various types of interface with $I_w = 3$. Target platform: Virtex 5 (xc5vlx30-3ff324) FPGA board.

for long and short messages, respectively (expressed in $nJ/bit$). Also in this case, the difference between long-message energy and short-message energy relates to the inclusion of the finalization processing in the measurement.

As can be seen from the amount of reported data, there are many different dimensions where the comparison is possible. Since our main goal is a high-throughput implementation of all the candidates, we provide Fig. 3.23 where the candidates are compared with respect to the highest achievable throughput. We also offer the throughput estimates assuming different interfaces. The throughput is first estimated for the core function. Next, we provide the throughput figures assuming the ideal interface, meaning that we use only $I_w$ clock cycles for the input and another $I_w$ clock cycles for the output. Finally, we measure the throughput assuming a realistic interface width (from 16 bits to 128 bits).

Here, we draw an interesting, somewhat natural conclusion. The influence of the interface width is more noticeable for the candidates that have a small number of rounds and a larger size of the input block. Therefore, one may notice that the influence of the fixed interface is especially noticeable for BMW, Grøstl, Keccak, and Luffa.

In order to have a complete picture regarding the hardware cost that one needs to pay for implementing a high-throughput version of each candidate, we provide Fig. 3.24. The left-hand side of the figure represents a throughput versus area graph, ignoring the influence of the fixed interface, while the right-hand part shows the same graph by taking the interface into account. The candidates within the dashed ellipse are the ones with the largest Throughput/Area ratio.

Table 3.4: Results of the SHA-3 candidates on Virtex-5 (xc5vlx30-3ff324).

| SHA-3 Candidate | Input Block Size [bits] | Max. Clock Freq [MHz] | Total Number of Clock Cycles | | | | Long Message Throughput [Mb/s] | | Short Message Latency [$\mu$s] $M$=256 bits | | Number of Occupied Slices | Number of Slice Registers | Number of Slice LUTs |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | $I_{in}$ | $I_{out}$ | $I_{core}$ | $I_{final}$ | $T$ | $T_{Core}$ | $L$ | $L_{Core}$ | | | |
| SHA-256 | 512 | 260 | 96 | 48 | 68 | 0 | 812 | 1,958 | 0.815 | 0.262 | 609 | 1,224 | 2,045 |
| Blake | 512 | 115 | 99 | 48 | 22 | 0 | 487 | 2,676 | 1.443 | 0.191 | 1,660 | 1,393 | 5,154 |
| BMW | 512 | 34 | 96 | 48 | 2 | 2 | 178 | 8,704 | 4.353 | 0.118 | 4,350 | 1,317 | 15,012 |
| CubeHash | 256 | 185 | 48 | 48 | 16 | 160 | 740 | 2,960 | 1.816 | 1.038 | **590** | 1,316 | **2,182** |
| ECHO | 1,536 | 149 | 315 | 48 | 99 | 0 | 553 | 2,312 | 3.101 | 0.664 | 2,827 | 4,198 | 9,885 |
| Fugue | 32 | 78 | 6 | 48 | 2 | 37 | 312 | 1,248 | 2.013 | 0.705 | 4,013 | 1,043 | 13,255 |
| Grøstl | 512 | 154 | 96 | 48 | 10 | 10 | 744 | 7,885 | 1.065 | 0.130 | 2,616 | 1,570 | 10,088 |
| Hamsi | 32 | 210 | 6 | 48 | 4 | 5 | 672 | 1,680 | 0.681 | 0.195 | 718 | **841** | 2,499 |
| JH | 512 | 201 | 96 | 48 | 39 | 0 | 762 | 2,639 | 0.910 | 0.194 | 2,661 | 1,612 | 8,392 |
| Keccak | 1,024 | 205 | 192 | 48 | 24 | 0 | 972 | **8,747** | 1.288 | 0.117 | 1,433 | 2,666 | 4,806 |
| Luffa | 256 | **261** | 48 | 48 | 9 | 9 | **1,172** | 7,424 | **0.655** | **0.103** | 1,048 | 1,446 | 3,754 |
| Shabal | 512 | 228 | 96 | 48 | 50 | 150 | 800 | 2,335 | 1.509 | 0.877 | 1,251 | 2,061 | 4,219 |
| SHAvite-3 | 512 | 251 | 108 | 48 | 38 | 0 | 880 | 3,382 | 0.773 | 0.151 | 1,063 | 1,363 | 3,564 |
| SIMD | 512 | 75 | 96 | 48 | 46 | 0 | 270 | 835 | 2.533 | 0.613 | 3,987 | 6,693 | 13,908 |
| Skein | 512 | 91 | 102 | 48 | 19 | 19 | 385 | 2,452 | 2.066 | 0.418 | 1,370 | 1,956 | 4,979 |

Table 3.5: Power and energy consumption of the SHA-3 candidates on Virtex-5 (xc5vlx30-3ff324).

| SHA-3 Candidate | Power $[W]$ | | Energy $[nJ/bit]$ | |
|---|---|---|---|---|
| | Long Msg | Short Msg | Long Msg | Short Msg |
| SHA-256 | 0.21 | 0.21 | 0.65 | 1.30 |
| Blake | 0.27 | 0.27 | 0.49 | 0.98 |
| BMW | 0.41 | 0.41 | **0.07** | **0.27** |
| CubeHash | **0.23** | **0.23** | 0.61 | 7.27 |
| ECHO | 0.28 | 0.28 | 0.75 | 4.49 |
| Fugue | 0.36 | 0.37 | 0.95 | 3.28 |
| Grøstl | 0.31 | 0.31 | 0.25 | 1.00 |
| Hamsi | **0.23** | **0.23** | 1.19 | 1.52 |
| JH | 0.25 | 0.25 | 0.80 | 1.60 |
| Keccak | 0.29 | 0.29 | 0.29 | 1.16 |
| Luffa | 0.24 | 0.24 | 0.36 | 1.07 |
| Shabal | **0.23** | **0.23** | 0.94 | 7.62 |
| SHAvite-3 | 0.24 | 0.24 | 0.73 | 1.45 |
| SIMD | 0.29 | 0.29 | 1.09 | 2.17 |
| Skein | 0.30 | 0.30 | 0.47 | 1.86 |

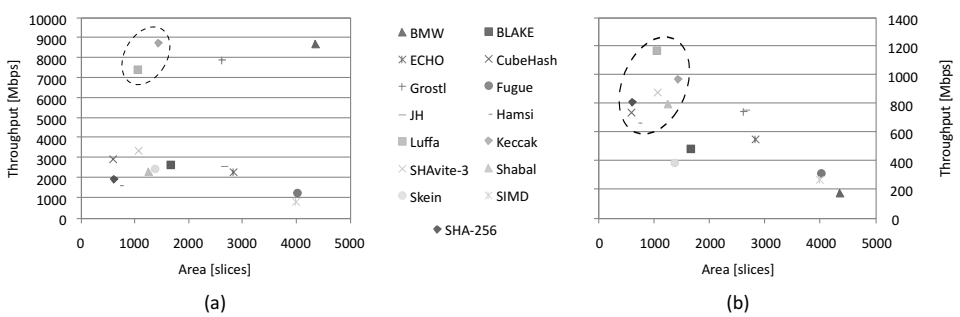


Figure 3.24: Throughput versus area graph: (a) core function only. (b) fixed interface with $w = 16$ bits and $I_w = 3$. Target platform: Virtex 5 (xc5vlx30-3ff324) FPGA board.

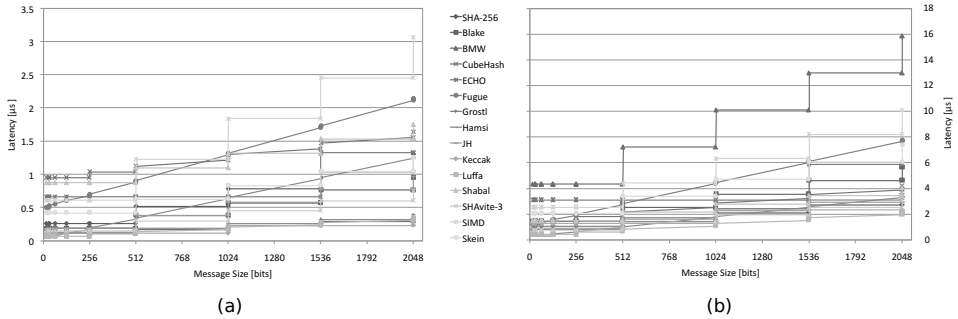


Figure 3.25: Latency versus message size graph: (a) core function only. (b) fixed interface with $w = 16$ bits and $I_w = 3$. Target platform: Virtex 5 (xc5vlx30-3ff324) FPGA board.

Due to the very small number of rounds of the core function, the hash candidate BMW provides the highest core throughput among all candidates. The hardware price, however, due to the heavy unrolled architecture, is large (BMW also consumes most of the hardware resources). Other candidates that have noticeably high core throughput are Keccak, Grøstl and Luffa. Furthermore, Luffa and Keccak achieve a high core throughput with a relatively small hardware cost.

Assuming a fixed interface with parameters $w = 16$ bits and $I_w = 3$, which indeed complies with our evaluation platform, Luffa achieves the highest throughput. Luffa also has the highest hardware efficiency since it achieves the highest throughput with a relatively small hardware cost. Other candidates that have noticeably high throughput in this case are Keccak and SHAvite-3.

To have a complete picture regarding the latency of all candidates with respect to different sizes of the unpadded message, we provide Fig. 3.25. The left-hand side represents the core latency of all candidates versus message size, while the right-hand side represents the latency by taking the 16-bit interface into account. It is interesting to observe that for short messages, with less than 512 bits, CubeHash, Shabal, and Fugue show rather high core latency. This is due to the fact that these candidates have a large number of rounds in the final stage of the hashing process. The stair-steps on the graph appear due to the fact that an additional message block for padding is needed whenever we hash an unpadded message with size equal to the input block size of the algorithm.

In order to explore the influence of a fixed interface on the minimum latency, we additionally provide Fig. 3.26. Here, we assume the length of the short unpadded message to be 256 bits. It can be noticed that Luffa has the shortest core latency among all candidates. Even when including the interface overhead, Luffa shows the

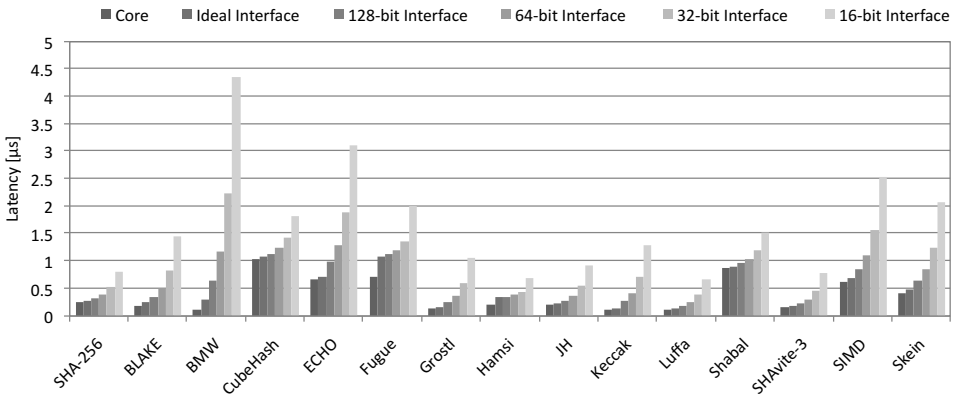


Figure 3.26: Minimum latency for various types of interface with $I_w = 3$. Target platform: Virtex 5 (xc5vlx30-3ff324) FPGA board.

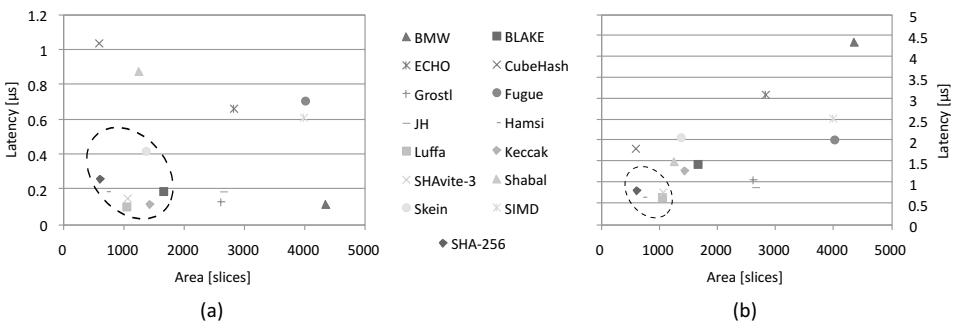


Figure 3.27: Latency versus area graph: (a) core function only. (b) fixed interface with $w = 16$ bits and $I_w = 3$. Target platform: Virtex 5 (xc5vlx30-3ff324) FPGA board.

best performance. The candidates with a larger number of cycles needed for the finalizing stage, such as CubeHash, Fugue, and Shabal, have noticeably high core latency. The biggest influence of a fixed standard interface is again demonstrated by BMW.

Finally, in Fig. 3.27 we show a latency versus area graph. Regarding the core latency versus area, we can select the set of candidates which show somewhat better performance compared to others, and those are: Luffa, Keccak, SHAvite-3, Hamsi, Blake, and Skein. With respect to the total latency (including the interface

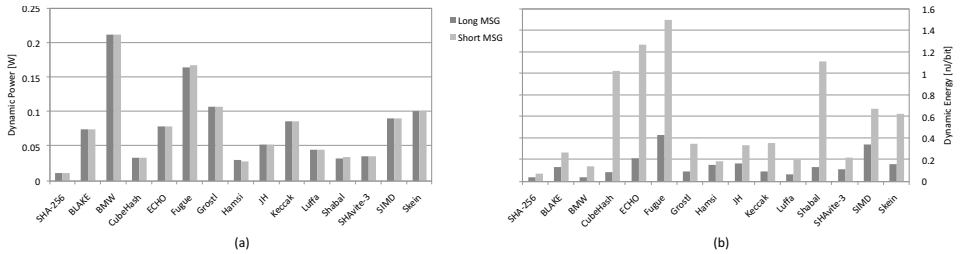


Figure 3.28: (a) Dynamic power consumption. (b) dynamic energy consumption. Target platform: Virtex 5 (xc5vlx30-3ff324) FPGA board.

overhead) versus area, the set containing Hamsi, Luffa, and SHAvite-3 shows the best performance. These candidates show the smallest Latency-Area product.

### Power and Energy Consumption

As mentioned in Section 3.5.2, we distinguish between a platform-dependent power (static power) and an algorithm-dependent power consumption (dynamic power). We measured the static power dissipation of the Virtex 5 FPGA on SASEBO-GII to be around 200 mW. Hence, the power numbers listed in Table 3.5 are dominated by the static power. To have an accurate comparison, we simply compare the candidates with respect to their algorithmic properties by measuring the dynamic power only, as depicted in Fig. 3.28a (the dynamic power is simply obtained by subtracting the static power from the total power consumption).

Due to the similar behavior during the update and the final phase, the difference between the power consumption for long and short messages is negligible. On the other hand, the dynamic energy consumption (see Fig. 3.28b) differs for long and short messages and is especially noticeable for candidates which require additional cycles for the finalizing stage (CubeHash, Fugue, Grøstl, Shabal, and Skein). ECHO and Keccak also have the same discrepancy, and this is due to the large input block while hashing a short message of only 256 bits. Since BMW is the largest design among all candidates, its power consumption is thereby the largest as well. However, due to the very small number of cycles needed for a hashing operation, BMW on the other hand consumes the least amount of energy.

#### Algorithmic Features versus Implementation Results

Recalling the classification from Fig. 3.19 we conclude that no obvious connection can be made between the hardware performance and the design properties of the fourteen candidates. As an illustration we provide the fact that the top 5 designs with respect to the core throughput are Keccak (4-bit S-box/Boolean, Sponge, 1024-bit), BMW (ARX, wide-pipe, 512-bit), Grøstl (8-bit S-box, wide-pipe, 512-bit), Luffa (4-bit S-box/Boolean, Sponge, 256-bit) and SHAvite-3 (8-bit S-box, narrow-pipe, 512-bit). They, all together, basically cover the complete design space as defined in Section 3.5.2.

However, several interesting conclusions can still be made by observing some of the algorithmic features versus the implementation results. Therefore, we observe that the narrow-pipe designs (Blake, Hamsi, SHAvite-3, and Skein) offer relatively low core throughput. Grøstl, Keccak, and Luffa, on the other hand, provide high throughput regardless of the interface type (none of them is a narrow-pipe design). Designs with very small input block size of only 32 bits (Fugue and Hamsi) offer a relatively small core throughput. ECHO, which is the candidate with the largest input block size also offers a small throughput, but this is more because ECHO has the largest number of rounds for hashing a block of the message.

As a conclusion of this section we argue that the Sponge based candidates with the light non-linear part (4-bit S-box/Boolean based) and large "input block size/number of rounds" ratio (Keccak and Luffa) show somewhat better overall performance in comparison to the other candidates. Due to the simplicity of the design, they have the shortest critical path, which in combination with the large "input block size/number of rounds" ratio results in high throughput and low latency.

### 3.5.5 ASIC Evaluation Results

In order to have a complete picture regarding the possible hardware platforms, we synthesized the code of SHA-256 and all fourteen SHA-3 candidates using the STM 90 nm CMOS technology. We used Synopsys Design Compiler version A-2007.12-SP3. The tool automatically estimated power consumptions by using its own signal switching model for the datapaths, and thus we did not control test vectors for the power estimation.

We synthesized several circuits from one design by changing speed constraints, and chose the three circuits, which showed the smallest size, the highest throughput, and the highest efficiency (throughput/gate). The result are presented in Table 3.6.

Our results are based on synthesis and we only provide the core throughput and the core latency as measures of speed. However, as we further plan to tape out the

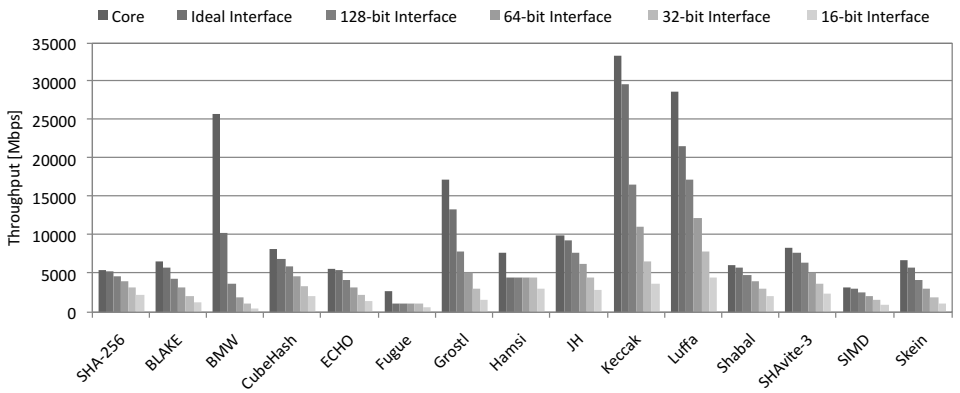


Figure 3.29: Maximum throughput for various types of interface with $I_w = 3$. Target platform: STM 90 nm CMOS technology, synthesis results.

candidates which advanced to the third, and final round of the competition, and to use a very similar evaluation platform (SASEBO-R), we provide estimates of the interface influence on the ASIC performance as well.

Similar to the previous section, we provide the following figures:

- Fig. 3.29 – Maximum throughput of all fourteen candidates assuming various types of interface.
- Fig. 3.30 – Throughput versus area graph.
- Fig. 3.31 – Latency versus message size graph.
- Fig. 3.32 – Minimum latency of all fourteen candidates assuming various types of interface.
- Fig. 3.33 – Latency versus area graph.
- Fig. 3.34 – Power and energy consumption.

Since the designs were implemented to achieve the highest throughput, only the first subrow in each row is relevant for comparison of maximum frequency, maximum core throughput, and minimum core latency. Therefore, we mark (in **bold** and gray) fastest and slowest designs by observing the first subrows only. For other columns, we mark the extreme results by observing every subrow in each row.

Table 3.6: Synthesis results of the SHA-3 candidates using 90 nm CMOS technology.

| SHA-3 Candidate | Max. Freq.† [MHz] | Max. Core Throughput† [Mb/s] | Min. Core Latency† [$\mu$s] | Total Area [GE] | Dynamic Power‡ [mW] | Dynamic Energy [pJ/bit] | | Hardware Efficiency [kb/sGE] |
|---|---|---|---|---|---|---|---|---|
| | | | | | | Long Msg | Short Msg | |
| SHA-256 | 735 | 5,536 | 0.09 | 18,677 | 3.11 | 2.31 | 4.62 | 290.6 |
| | 356 | 2,680 | 0.19 | 13,199 | 2.09 | 1.55 | 3.09 | 203.0 |
| | 117 | 878 | 0.58 | 11,332 | 1.77 | 1.32 | 2.63 | 77.4 |
| BLAKE-32 | 286 | 6,668 | 0.08 | 36,944 | 10.84 | 4.66 | 9.31 | 180.5 |
| | 260 | 6,061 | 0.08 | 30,292 | 4.94 | 2.12 | 4.25 | 200.1 |
| | 147 | 3,412 | 0.15 | 23,214 | 3.77 | 1.62 | 3.24 | 147.0 |
| BMW-256 | 101 | 25,937 | 0.04 | 128,655 | 9.25 | 0.36 | 1.44 | 201.6 |
| | 84 | 21,603 | 0.05 | 115,001 | 8.46 | 0.33 | 1.32 | 187.9 |
| | 67 | 17,262 | 0.06 | 105,566 | 7.47 | **0.29** | **1.16** | 163.5 |
| CubeHash16/32-256 | 515 | 8,247 | 0.37 | 35,548 | 7.07 | 4.42 | 53.00 | 232.0 |
| | 352 | 5,834 | 0.55 | 21,336 | 4.07 | 2.54 | 30.53 | 264.1 |
| | 172 | 2,749 | 1.12 | **16,320** | 3.60 | 2.25 | 26.98 | 168.5 |
| ECHO-256 | 362 | 5,621 | 0.27 | 101,068 | 17.24 | 11.11 | 11.11 | 55.6 |
| | 260 | 4,040 | 0.38 | 97,803 | 8.88 | 5.73 | 34.36 | 59.6 |
| | 147 | 2,278 | 0.67 | 57,834 | 8.32 | 5.36 | 32.16 | 39.4 |
| Fugue-256 | 170 | 2,721 | 0.32 | 56,734 | 3.57 | 2.23 | 7.66 | 48.0 |
| | 113 | 1,808 | 0.49 | 45,553 | 3.01 | 1.88 | 6.46 | 37.9 |
| | 78 | 1,245 | 0.71 | 46,683 | 2.92 | 1.82 | 6.27 | 26.7 |
| Grøstl-256 | 338 | 17,297 | 0.06 | 139,113 | 22.52 | 4.40 | 17.59 | 124.3 |
| | 258 | 13,196 | 0.08 | 86,191 | 12.74 | 2.49 | 9.95 | 153.1 |
| | 128 | 6,547 | 0.16 | 56,665 | 7.85 | 1.53 | 6.13 | 115.5 |
| Hamsi-256 | 971 | 7,767 | 0.04 | 67,582 | 6.94 | 8.67 | 11.11 | 114.9 |
| | 544 | 4,348 | 0.08 | 36,981 | 3.44 | 4.31 | 5.51 | 117.6 |
| | 352 | 2,817 | 0.12 | 32,116 | 2.80 | 3.50 | 4.48 | 87.7 |
| JH-256 | 763 | 10,022 | 0.05 | 54,594 | 2.94 | 2.24 | 4.48 | 183.6 |
| | 694 | 9,117 | 0.06 | 42,775 | **2.07** | 1.57 | 3.14 | 213.1 |
| | 353 | 4,639 | 0.11 | 31,864 | 2.13 | 1.63 | 3.25 | 145.6 |
| Keccak(-256) | 781 | **33,333** | **0.03** | 50,675 | 6.36 | 1.55 | 6.21 | 657.8 |
| | 541 | 23,063 | 0.04 | 33,664 | 3.62 | 0.88 | 3.54 | 685.1 |
| | 355 | 15,130 | 0.07 | 29,548 | 3.52 | 0.86 | 3.44 | 512.0 |
| Luffa-256 | **1010** | 28,732 | **0.03** | 39,642 | 5.14 | 1.81 | 5.42 | 724.8 |
| | 538 | 15,293 | 0.05 | 19,797 | 2.85 | 1.00 | 3.01 | **772.5** |
| | 263 | 7,466 | 0.10 | 19,359 | 2.91 | 1.02 | 3.07 | 385.6 |
| Shabal-256 | 592 | 6,059 | 0.34 | 34,642 | 5.80 | 5.66 | 45.30 | 174.9 |
| | 544 | 5,565 | 0.37 | 30,328 | 3.13 | 3.05 | 24.42 | 183.5 |
| | 351 | 3,593 | 0.57 | 27,752 | 3.16 | 3.08 | 24.65 | 129.5 |
| SHAvite-$3_{256}$ | 625 | 8,421 | 0.06 | 59,390 | 3.61 | 2.68 | 5.36 | 141.8 |
| | 493 | 6,637 | 0.08 | 42,036 | 2.46 | 1.83 | 3.66 | 157.9 |
| | 207 | 2,784 | 0.18 | 33,875 | 2.41 | 1.79 | 3.57 | 82.2 |
| SIMD-256 | 285 | 3,171 | 0.16 | 138,980 | 13.56 | 12.18 | 24.37 | 22.8 |
| | 261 | 2,906 | 0.18 | 122,118 | 10.77 | 9.67 | 19.35 | 23.8 |
| | 113 | 1,259 | 0.41 | 88,947 | 10.74 | 9.64 | 19.29 | 14.2 |
| Skein-512-256 | 251 | 6,734 | 0.15 | 43,132 | 17.17 | 6.37 | 25.48 | 76.4 |
| | 206 | 5,551 | 0.18 | 28,782 | 4.42 | 4.68 | 18.73 | 87.7 |
| | 50 | 1,347 | 0.76 | 22,562 | 3.25 | 3.25 | 13.01 | 79.0 |

† Only the first subrow in each row is relevant for comparison of Max. Frequency, Max. Core Throughput, and Min. Core Latency.
‡ The power consumption is estimated for the frequency of 100 MHz.

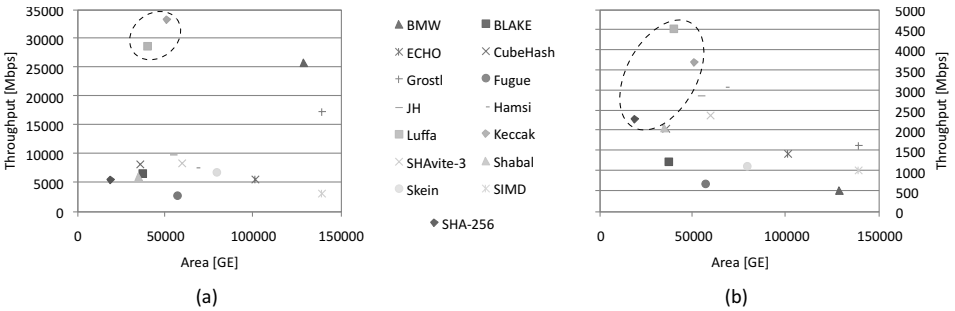


Figure 3.30: Throughput versus area graph: (a) core function only. (b) fixed interface with $w = 16$ bits and $I_w = 3$. Target platform: STM 90 nm CMOS technology, synthesis results.

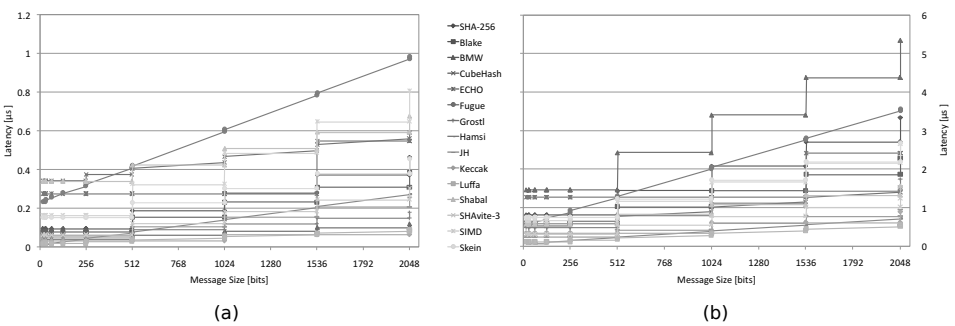


Figure 3.31: Latency versus message size graph: (a) core function only. (b) fixed interface with $w = 16$ bits and $I_w = 3$. Target platform: STM 90 nm CMOS technology, synthesis results.

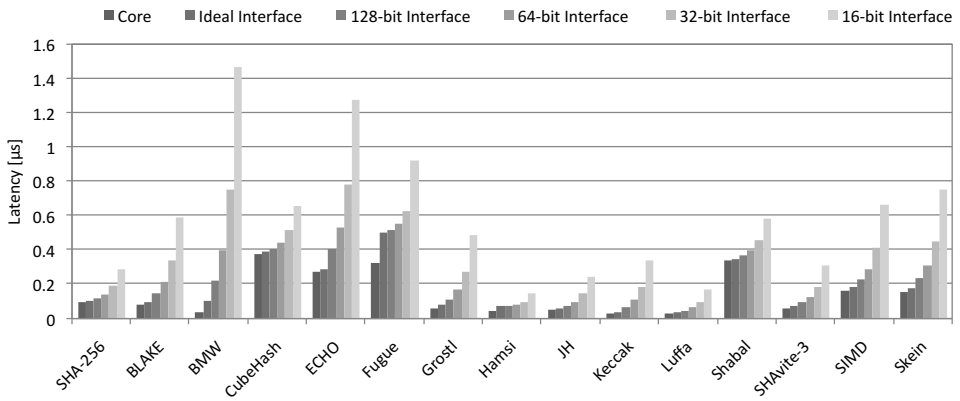


Figure 3.32: Minimum latency of all 14 candidates assuming various types of interface with $I_w = 3$. Target platform: STM 90 nm CMOS technology, synthesis results.

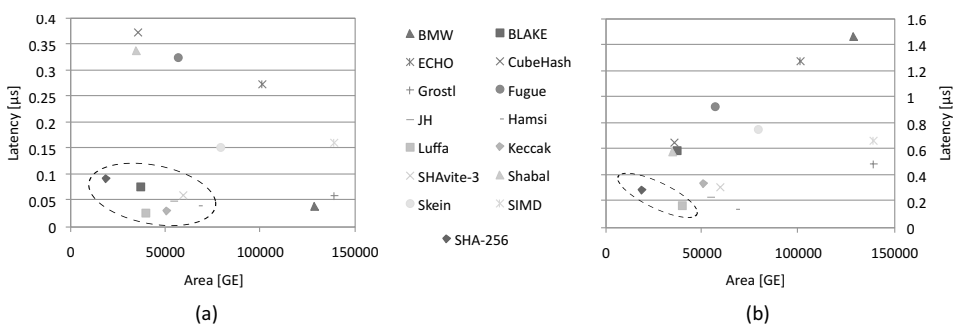


Figure 3.33: Latency versus area graph: (a) core function only. (b) Fixed interface with $w = 16$ bits and $I_w = 3$. Target platform: STM 90 nm CMOS technology, synthesis results.

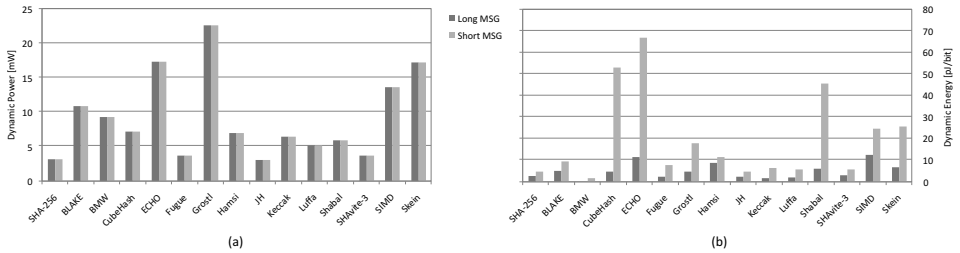


Figure 3.34: (a) Dynamic power consumption. (b) Dynamic energy consumption. Target platform: STM 90 nm CMOS technology, synthesis results.

### 3.5.6 Correlation between ASIC and FPGA Results

By observing the provided graphs we argue that there is a good level of correlation between the ASIC and the FPGA results, with a few considerable differences. For example, when observing Fig. 3.24a and Fig. 3.30a, we notice that Fugue and Grøstl differ considerably, while Blake and Hamsi differ noticeably. Further comparing Fig. 3.24b versus Fig. 3.30b, Fig. 3.27a versus Fig. 3.27a, and Fig. 3.27b versus Fig. 3.33b we notice that Fugue, Grøstl, and JH differ considerably. Another considerable difference is in the power/energy consumption for BMW, ECHO, and Grøstl. These three candidates are the largest in area among all, and since the power is estimated and measured using different platforms (ASIC and FPGA), this difference is acceptable. Therefore, we conclude that the obtained FPGA results represent a rather reliable way of estimating the ASIC performance, especially with respect to speed and area.

### 3.5.7 The SHA-3 Finalists

On December 9, 2010, NIST has selected five SHA-3 finalists to advance to the final round:

- Blake
- Grøstl
- JH
- Keccak
- Skein

In 2012, NIST will choose a single algorithm to become the SHA-3 standard. An extensive cryptanalysis as well as a thorough performance examination on a multitude of platforms of these five algorithms is therefore expected before the winner is finally announced.

### 3.5.8 Summary

For a complete hardware evaluation, there are plenty of evaluation platforms to be considered. Therefore, fixing one is crucial for conducting a fair and consistent comparison. In this section, we proposed an evaluation platform and a consistent evaluation method to conduct a fair hardware evaluation of the remaining SHA-3 candidates. This proposal meets the requirements analyzed from actual hash applications and conditions of standard selection. The platform includes a SASEBO-GII evaluation board, evaluation software, and appropriate interface definition. Using this method, we implement all the second-round SHA-3 candidates and obtain the resulting cost and performance factors. This technical study provides a fair and consistent evaluation scheme. At the end, we hope that by sharing our experience we contribute to the SHA-3 competition and by providing the proposed methodology we influence other similar future selections of standard cryptographic algorithms.

## 3.6 Hardware Evaluation of the Luffa Hash Family

### Publication Data

M. Knežević and I. Verbauwhede, "Hardware Evaluation of the Luffa Hash Family," in *Workshop on Embedded Systems Security – WESS 2009*, 6 pages, 2009.

#### Personal Contributions

- Principal author.

Our novel contribution consists of providing efficient hardware architectures for the Luffa hash algorithm. We explore different trade-offs and propose several architectures, targeting both compact and high-throughput designs.

In the previous section, we provided a thorough hardware comparison of all fourteen second-round SHA-3 candidates. Here, we focus on the Luffa hash algorithm. We explore some of the possible trade-offs and propose several architectures, targeting both compact and high-throughput designs. The most compact architecture of

10,157 gate equivalences (GE) was achieved for the 224/256-bit version of Luffa. The same version, optimized for speed, achieves a throughput of 32.7 Gb/s, while the pipelined design approaches the throughput of 291.7 Gb/s. Techniques such as retiming, pipelining and simple multiplexing were used for the high-throughput, pipelined and compact implementations, respectively.

Figure 3.35 illustrates a generic construction of the Luffa hash algorithm. It consists of the intermediate mixing $C'$ (called a *round function*) and the *finalization* $C''$. The round function is a composition of a *message injection function* $MI$ and a *permutation* $P$ of $w$ 256-bit inputs as shown in Fig. 3.36 (Luffa-224/256 variant). The permutation is divided into multiple sub-permutations $Q_j$ of 256-bit inputs.

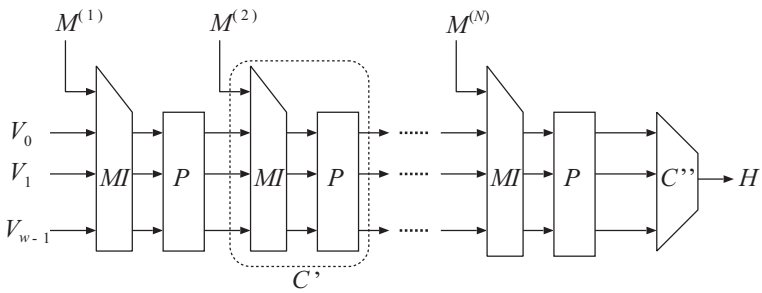


Figure 3.35: A generic construction of the Luffa hash algorithm.

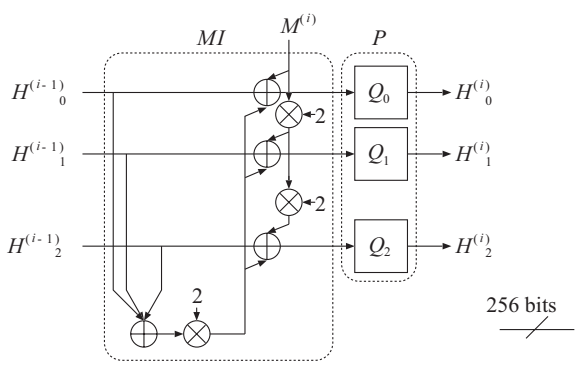


Figure 3.36: The round function $C'$ ($w = 3$).

The family of hash functions Luffa consists of four variants specified by the output hash length (224, 256, 384 and 512 bits). The main difference is the number of sub-permutations $w$ and the message injection function $MI$ used in each of them.

The number of sub-permutations equals 3, 4 and 5 for the version of 224/256-bit, 384-bit and 512-bit Luffa, respectively.

The finalization $C''$ consists of iterating an *output function* $OF$ and a round function with a fixed message 0x00 . . . 0. If the number of (padded) message blocks is greater than one, a blank round with a fixed message block 0x00 . . . 0 is applied at the beginning of the finalization. The output function XORs all the block values and outputs the result of 256-bits. Figure 3.37 illustrates the finalization function. The output of the hash function is defined as $Z_0$ for the 224/256-bit version, $Z_0$ concatenated with the most significant half of $Z_1$ for the 384-bit version, and $Z_0$ concatenated with $Z_1$ for the 512-bit version of Luffa.

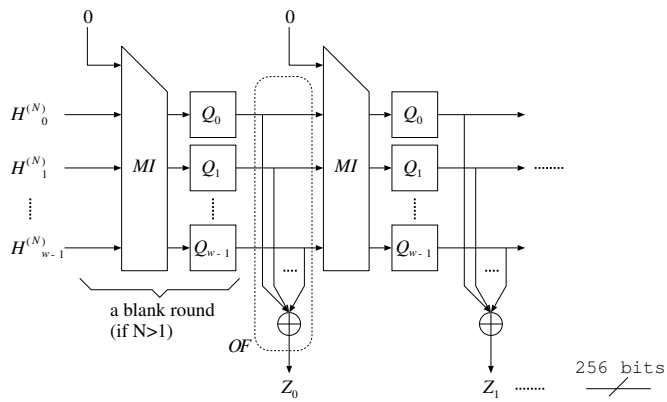


Figure 3.37: The finalization function $C''$.

The round function of Luffa uses a non-linear permutation $Q_j$ with input and output size of 256 bits. The main component of the permutation $Q_j$ is the *step function* which consists of `SubCrumb` and `MixWord` blocks, as illustrated in Fig. 3.38. A `SubCrumb` block contains 4-bit input S-boxes (see Fig. 3.39) while `MixWord` represents linear permutations of two 32-bit words (see Fig. 3.40). The parameters $\sigma_i$ are fixed and given as $\sigma_1 = 2$, $\sigma_2 = 14$, $\sigma_3 = 10$, $\sigma_4 = 1$. Finally, `AddConstant` is performed before the output of the step function is ready. It is a simple XOR with precalculated constant values. To perform the complete round, one needs to execute the message injection function once and the step function 8 times. For a detailed description of the Luffa hash family please refer to the work of De Cannière et al. [28].

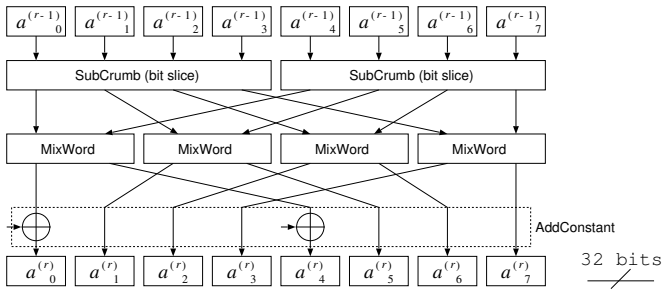


Figure 3.38: The step function.

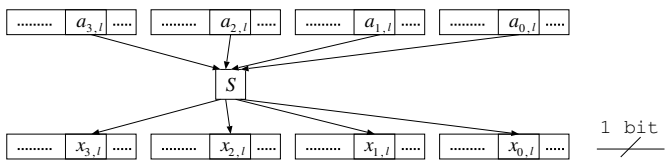


Figure 3.39: `SubCrumb` block.

## 3.6.1 Hardware Implementation

In this section we propose four different architectures for the Luffa hash family. First, we target a high-throughput architecture. Second, we focus on a compact design. Then, we show some of the possible area-throughput trade-offs, and finally we propose a fully pipelined architecture that reaches a throughput of 291.7 Gb/s.

A hardware performance evaluation of the Luffa hash family was done by synthesizing the proposed designs using a 0.13 $\mu$m CMOS High-Speed standard cell library. The code was first written in GEZEL [154] and tested for its functionality using the test vectors provided by the software implementations. The GEZEL code was then translated to VHDL and synthesized using the Synopsys Design Compiler version Y-2006.06.

### High-Throughput Implementation

For the high-throughput implementation, the goal was to minimize the critical path. To implement the round function, we have used $w$ permutation blocks in parallel, each of them containing 64 S-boxes and 4 `MixWord` blocks. The straightforward implementation, outlined in Fig. 3.41, resulted in a critical path of 1.18 ns and a cycle count of 8. The critical path was placed from the input of the message

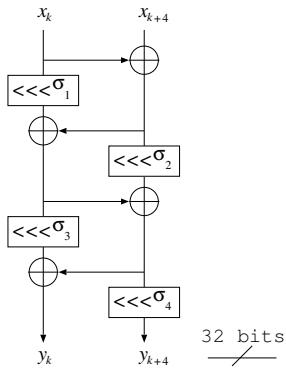


Figure 3.40: `MixWord` block.

injection function to the output of the permutation block (dashed arrow). Inputs and outputs of the step function (permutation block) are denoted as $A_j^r$ where $1 \leq r \leq 8$ and $0 \leq j \leq w-1$.

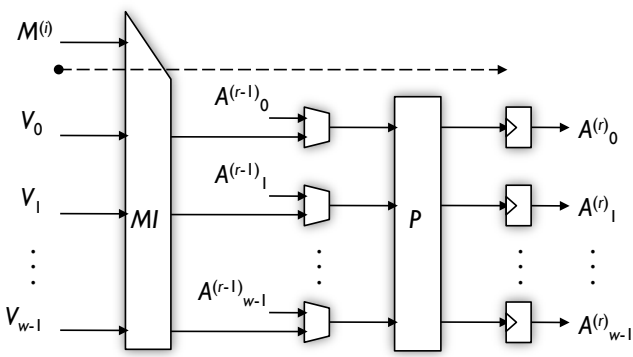


Figure 3.41: Straightforward implementation of the round function.

As the message injection function is performed only once at the beginning of every round, we moved the state registers at the input of the permutation blocks (see Fig. 3.42). It resulted in a faster design, shortening the critical path to only 0.87 ns. One more clock cycle had to be spent in order to perform the complete round, but the final throughput got increased by about 20 %.

The synthesis results are given in Table 3.7. As we can see, all variants of the Luffa hash algorithm achieve a throughput of more than 30 Gb/s. The throughput for

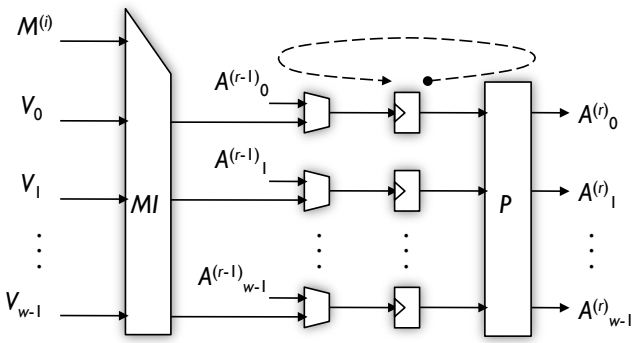


Figure 3.42: High-Throughput implementation of the round function.

"one-block" message as well as for the very long message was calculated according to the following equation:[3]

$$\text{Throughput} = \frac{\text{Frequency}}{\text{Number of Cycles}} \times 256 \text{ bit} \ .$$

Table 3.7: High-Throughput implementations of the Luffa hash family.

| Luffa Variant | Area [GE] | Freq. [MHz] | Number of cycles per round | Throughput [Mb/s] |
|---|---|---|---|---|
| 224/256 | 25,833 | 1,149 | 9 | 32,683 |
| 384 | 34,401 | 1,149 | 9 | 32,683 |
| 512 | 40,715 | 1,111 | 9 | 31,602 |

To show some of the possible trade-offs regarding the high-throughput implementation, we have synthesized a number of different designs only by changing the clock frequency and hence, changing the total performance of the design (see Fig. 3.43).

### Compact Implementation

A compact implementation was made using only one non-linear permutation block. Inside the permutation block we have used only two S-boxes for implementing the

[3] "One-block" message is a message of exactly 256 bits after the padding is performed, so its hashing does not require the blank round in the finalization phase. For a very long message, the number of cycles spent for the finalization is negligible.

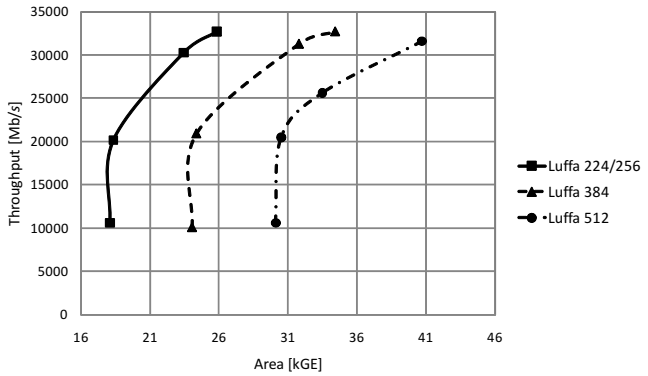


Figure 3.43: High-Throughput implementations.

`SubCrumb` block (see Fig. 3.44). To perform the whole `SubCrumb` operation we have used 32 cycles, regularly using the most significant bits of the registers $a_0 \dots a_7$ to be the inputs of the S-boxes. The registers were then shifted to the left and the least significant bits were updated using the outputs of the S-boxes.

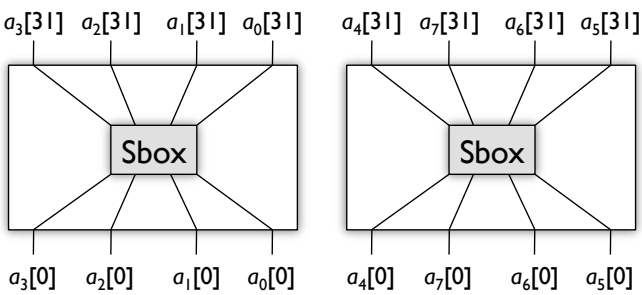


Figure 3.44: Compact `SubCrumb` block.

A single `MixWord` block was used for performing the `MixWord` operation (see Fig. 3.45). This approach resulted in a large number of cycles, while on the other hand it efficiently reduced the final gate count. We used $w$ 256-bit registers to maintain the internal state.

As can be seen from Table 3.8, the most compact implementation is obtained for Luffa-224/256 and consumes approximately 10 kGE. Note that our only goal for the compact implementation was to have a small circuit size, regardless of the final throughput. Hence, we fixed the frequency to 100 MHz and synthesized our design.

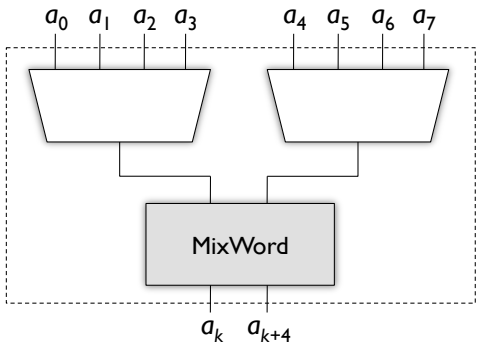


Figure 3.45: Compact `MixWord` block.

To show some of the possible trade-offs regarding the compact implementations we provide Fig. 3.46.

Table 3.8: Compact implementations of the Luffa hash family.

| Luffa Variant | Area [GE] | Freq. [MHz] | Number of cycles per round | Throughput [Mb/s] |
|---|---|---|---|---|
| 224/256 | 10,157 | 100 | 891 | 29 |
| 384 | 13,168 | 100 | 1188 | 21 |
| 512 | 16,720 | 100 | 1485 | 17 |

### Throughput-Area Trade-Offs

Finally, to show all the possible trade-offs, we have also implemented a version with only one non-linear permutation block but this time, inside the permutation block, we have used 64 S-boxes and 4 `MixWord` blocks. This approach efficiently reduced the number of clock cycles, while introducing a small overhead in area. Thus, a design of, e.g. Luffa-224/256 containing only 12.9 kGE achieved a throughput of more than 15 Gb/s. For each of the Luffa hash functions we have synthesized a few versions, some of them with the constraints on area and some with the constraints on speed. The most compact and the fastest designs are given in Table 3.9, whereas Fig. 3.47 further shows the throughput-area trade-offs for the given implementations.

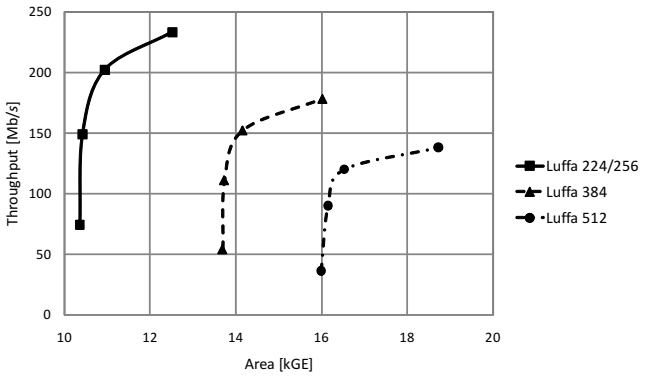


Figure 3.46: Compact implementations.

Table 3.9: Throughput-Area trade-offs of the Luffa hash family.

| Luffa Variant | Area [GE] | Freq. [MHz] | Number of cycles per round | Throughput [Mb/s] |
|---|---|---|---|---|
| 224/256 | 12,057 | 221 | 25 | 6,279 |
| 224/256 | 17,315 | 781 | 25 | 22,222 |
| 384 | 15,535 | 193 | 33 | 5,491 |
| 384 | 21,853 | 699 | 33 | 19,891 |
| 512 | 18,643 | 195 | 41 | 5,556 |
| 512 | 25,264 | 662 | 41 | 18,837 |

### Pipelined Implementation

When hashing independent message blocks, one can benefit from using the pipelined architecture as illustrated in Fig. 3.48. Multiple non-linear permutation blocks need to be added ($8w$ blocks) as well as $8w$ pipelined 256-bit registers (one for each permutation block). This approach effectively increases the throughput more than 8 times at the cost of additional area overhead. As can be seen from Table 3.10, a throughput of 291.7 Gb/s is achieved for the Luffa-224/256 version at a cost of 151.3 kGE. This is a fully pipelined implementation and achieves the highest throughput in case of hashing 8 independent messages in parallel.

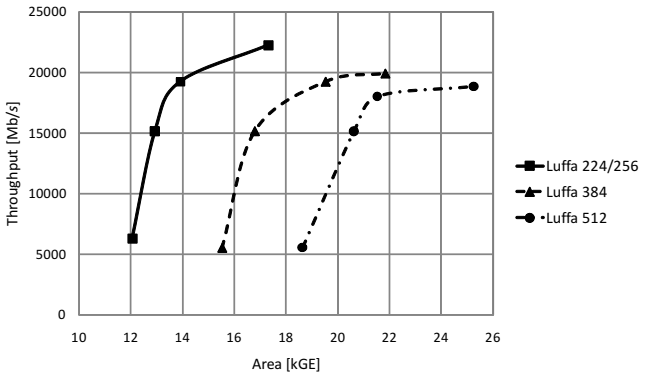


Figure 3.47: Throughput-Area trade-offs.

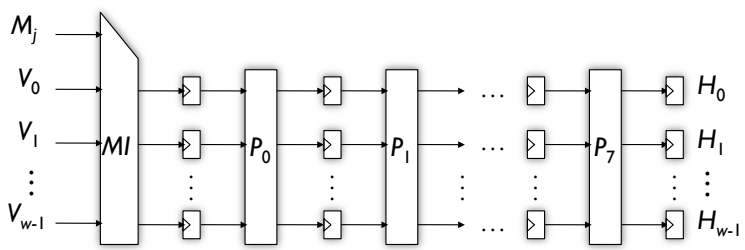


Figure 3.48: Pipelined architecture for the Luffa hash family.

### Comparison with Previous Standards

To compare our implementations with the implementations of the previous standards (SHA-1 and SHA-2), we provide Table 3.11 with the state of art results concerning the ASIC technology. Observing the results we can conclude that concerning throughput, the Luffa hash family outperforms all the previous standards. The only implementation that is more compact than Luffa is the implementation of SHA-1 proposed by Kim and Ryou [88]. This is however to be expected as the digest size produced by SHA-1 is 160 bits long in comparison to 256-bit digest of Luffa-224/256.

Please note that the compact implementation of SHA-256 by Kim and Ryou [89] has appeared after our results were published. Their implementation consumes only 8,588 GE using 0.25 $\mu$m CMOS process which is less than the most compact Luffa design.

Table 3.10: Pipelined implementations of the Luffa hash family.

| Luffa Variant | Area [GE] | Freq. [MHz] | Number of cycles per round | Throughput [Mb/s] |
|---|---|---|---|---|
| 224/256 | 151,347 | 1,282 | 9 | 291,738 |
| 384 | 222,293 | 1,219 | 9 | 277,507 |
| 512 | 273,211 | 1,219 | 9 | 277,507 |

Table 3.11: Comparison results with the previous standards.

| Design | Technology [$\mu$m] | Area [GE] | Throughput [Mb/s] |
|---|---|---|---|
| SHA-1 [101]† | 0.18 | 54,133 | 3,103 |
| SHA-224/256 [102]† | 0.13 | 22,025 | 5,975 |
| SHA-224/256 [38]† | 0.13 | N/A | $> 7,420$ |
| SHA-384/512 [102]† | 0.13 | 43,330 | 9,096 |
| Luffa-224/256† | 0.13 | 25,833 | 32,683 |
| Luffa-384† | 0.13 | 34,401 | 32,683 |
| Luffa-512† | 0.13 | 40,715 | 31,602 |
| SHA-1 [88]‡ | 0.25 | 6,812 | 130 |
| SHA-224/256 [152]‡ | 0.13 | 11,484 | 1,096 |
| SHA-384/512 [152]‡ | 0.13 | 23,146 | 1,455 |
| Luffa-224/256‡ | 0.13 | 10,157 | 29 |
| Luffa-384‡ | 0.13 | 13,168 | 21 |
| Luffa-512‡ | 0.13 | 16,720 | 17 |

† High-Throughput designs.
‡ Compact designs.

### 3.6.2 Summary

The hardware implementations of the Luffa hash family have been evaluated in this section. We conclude that the design is very well suited for both compact and high-throughput implementations. The most compact architecture of 10,157 GE was achieved for the 224/256-bit version of Luffa. The same version achieves a maximum throughput of 32.7 Gb/s, while the pipelined design reaches a throughput of 291.7 Gb/s. Due to ample parallelism provided by the Luffa hash family, it is possible to make plenty of trade-offs and choose the most appropriate design for a specific application. For example, a design of, e.g. Luffa-224/256 achieving 20.7 Gb/s consumes 16.7 kGE, while the design that consumes only 12.9 kGE achieves a throughput of more than 15 Gb/s.

Regarding hardware implementations, one can further explore the different levels of parallelism and make trade-offs by trading the throughput for the circuit size and vice versa. An especially challenging part remains the compact implementation of hash functions in general and hence, we expect more research effort in that direction.

## 3.7 Conclusion

Some of the well-known DSP techniques, used in the context of efficient hardware implementations of hash functions, have been discussed in this chapter. Combining them with several algorithm-specific techniques we achieved the highest throughput of the RIPEMD-160 hash function reported in literature. Due to the parallel computation used in this approach we tested our implementation on the Xilinx Virtex2Pro FPGA board and achieved a throughput of 624 Mb/s.

An extensive hardware evaluation of the fourteen second-round SHA-3 candidates was also presented in this chapter. By using a widely available testing platform, fixing a design strategy and evaluation criteria, and by publishing our code online, we established an environment for a comprehensive and fair evaluation that can be reproduced and publicly verified. The final decision of NIST, assuring that Blake, Grøstl, JH, Keccak, and Skein advance to the third, final round, is to some extent driven by the candidates' performance both in hardware and software. It is not surprising to see Keccak in the final round, as its hardware performance turned out to be best or second best, or at least in the top five according to all our evaluation criteria. Grøstl, for example, has one of the highest core throughputs both in ASIC and on an FPGA. JH has the smallest dynamic power consumption in ASIC among all candidates and it has one of the shortest latencies on both platforms. Blake and Skein on the other hand are both narrow-pipe and ARX based designs. Although one of the conclusions of Section 3.5.4 (where we compare algorithmic features against the implementation results) is that the narrow-pipe designs provide relatively low core throughput, these designs have the most in common with the biggest class of MD4-based hash algorithms. Therefore, we expect these two candidates to achieve further improvement in hardware performance by applying some of the algorithm-specific techniques, as discussed in this chapter.

Finally, we have further evaluated the hardware performance of the Luffa hash family, one of the fourteen SHA-3 candidates, by providing a wide range of its ASIC implementations. Luffa did not advance to the final round of the competition although, together with Keccak, provides the best performance in hardware.

# Chapter 4

# Lightweight Cryptography – A Battle for a Single Gate

## 4.1 Introduction

Technological developments in the field of low-end devices, such as Radio Frequency Identification (RFID) tags, are proceeding at a rapid pace preserving, however, never ending implementation challenges. Driven by the very fierce constraints, two of those are of utmost importance: silicon area and power. Based on economical and technical limitations respectively, these two constraints remain the key factors in today's evolution of low-cost devices. Due to the linear relationship between silicon area and chip manufacturing costs from one side, and the billions of tags produced every year from the other, the total production cost is naturally a limiting factor. A very constrained chip area, where an additional gate might lead to the solution not being used, is a way of addressing this challenge. Furthermore, the passive RFID tags are supplied with energy of the electromagnetic field provided by a reader, the strength of which decreases with operating distance. Therefore, the less power a passive RFID tag consumes, the longer the operating distance. Finally, assuming small footprint and reduced power consumption are achieved, one must not ignore the importance of speed since the implemented protocols need to be transparent to the end-users.

Low-end devices are used in many applications and environments, leading to an ever increasing need to provide security (and privacy). In order to satisfy these needs, several suitable building blocks, such as secure and low-cost block ciphers, have been developed in the past years. The growing importance as well as the lack of secure and suitable candidates, has initiated a line of research aiming to satisfy

these requirements. Some stream ciphers, such as Grain [67] and Trivium [29] may also be considered fit for these constrained environments, with 1,293 GE and 749 GE[1] implementations, respectively. However, some protocols cannot be realized using stream ciphers, thus, leaving the issue of finding a more compact and secure block cipher open.

It becomes clear that, once entering the world of lightweight cryptography, one has to be ready to sacrifice a certain level of security for efficient implementations. Specifically, targeting cryptographic primitives such as block ciphers we try to reduce the key length and the block size to an absolute acceptable minimum, therefore reducing the overall security of the primitive. Hence, a typical trade-off that is considered throughout this chapter is illustrated in Fig. 4.1.

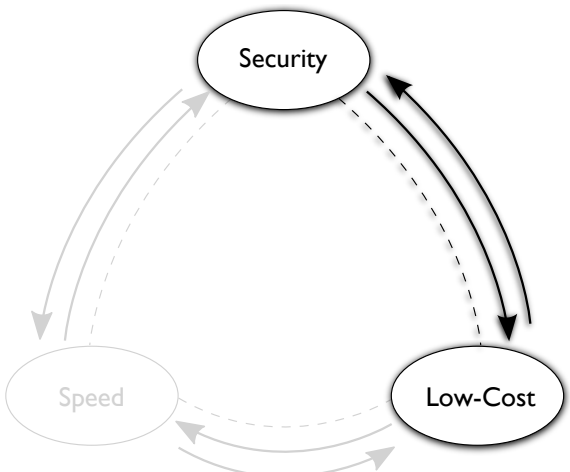


Figure 4.1: Security versus low-cost trade-off.

The chapter deals with one of the solutions to this open problem. We propose KATAN & KTANTAN, a family of small and efficient hardware-oriented block ciphers. The first set of ciphers consists of the KATAN ciphers, KATAN32, KATAN48, and KATAN64. All three ciphers accept 80-bit keys and have a different block size ($n$-bit for KATAN$n$). These three block ciphers are highly compact and achieve the minimal size (while offering adequate security). The second set, composed of KTANTAN32, KTANTAN48, and KTANTAN64, realize even smaller block ciphers in exchange for agility. KTANTAN$n$ is more compact than KATAN$n$, but at the same time, is suitable only for cases where the device is initialized with one key that can never be altered, i.e. for the KTANTAN families,

[1]This work is a full-custom design implemented with C$^2$MOS dynamic logic, by Mentens et al. [117]. The die size is equivalent to 749 standard CMOS logic NAND gates. The clock frequency required for this solution is far from being suitable for constrained environments.

the key of the device is burnt into the device. Thus, the only algorithmic difference between KATAN$n$ and KTANTAN$n$ is the key schedule (which is considered slightly more secure in the KATAN$n$ case).

Before going into details and providing a description of the KATAN & KTANTAN family, we first outline some related work.

### 4.1.1 Related Work

After the work of Feldhofer et al. [52], who provided the smallest implementation of the Advanced Encryption Standard (AES) [39] at the time, requiring only 3,400 GE, it became clear that AES is the candidate block cipher for low-cost devices. In 2006, Hämäläinen et al. [62] further improved the previous work by proposing a low area implementation of AES that requires only 3,100 GE and 160 clock cycles.

One of the requirements of the initial AES development as stated by the National Institute of Standards and Technology (NIST) in 1997, was to have an algorithm capable of protecting sensitive government information well into the next century [126]. Therefore, the block size and the key size of the smallest flavor of AES, both 128 bits, offer security margins that are not always needed for securing low-cost devices. In fact, a solution to this matter has already been proposed back in 1994 when Wheeler and Needham designed the Tiny Encryption Algorithm (TEA) [175]. TEA operates on a 64-bit data state with a key of 128 bits and consists of 64 rounds. As a response to a related key attack on TEA by Kelsey et al. [85], the tweaked version of TEA, called XTEA was published in 1998 [174]. The simplicity of the basic round function of TEA, led to the implementation by Yu et al. [179] of only 2,335 GE.

The years 2005 and 2006 bring several lightweight block ciphers, starting with mCrypton, a design proposed by Lim and Korkishko [109]. It has a 64-bit state and is specified for three different key lengths of 64, 96, and 128 bits, each requiring the equivalent size of 2,420 GE, 2,681 GE, and 2,949 GE, respectively. The Scalable Encryption Algorithm (SEA), proposed by Standaert et al. [157] is mainly targeted at embedded software applications, having a key and a block size of 96 bits. Instead of being a low-cost solution only, special emphasis is put on scalability in this case. This additional feature comes at a price of higher area requirements. Implemented by Macé et al. [111], the design results in a total area of 3,758 GE. Published by Hong et al. [71], HIGHT is a generalized Feistel-like structure with 64-bit block size and a 128-bit key. The authors claim a hardware requirement of 3,048 GE for the realization of this block cipher. Finally, as introduced by Poschmann et al. [100], a design based on the Data Encryption Standard (DES) with a new lightweight S-box, called DESL, resulted in 1,848 GE, requiring 144 cycles to encrypt a single block. The same paper proposes a stronger variant, DESXL, requiring 2,168 GE. Both variants have a key size of 56 bits and 64-bit data block.

In 2007, inspired by the AES finalist SERPENT [20], Bogdanov et al. [22] introduce the PRESENT block cipher. PRESENT has an SP-Network structure with a key of 80 bits and a block size of 64 bits, and can be implemented using the equivalent of 1,570 GE. A more dedicated implementation of PRESENT in 0.35 $\mu$m CMOS technology reaches 1,000 GE. The same design in 0.25 $\mu$m and 0.18 $\mu$m CMOS technology consumes 1,169 GE and 1,075 GE, respectively [145].

In 2009, Izadi et al. [79], introduce MIBS, a new lightweight block cipher. Its Feistel structure with a lightweight round function achieves the smallest implementation of 1,396 GE, having a key and a block size of 64 bits. The bigger version with key size of 80 bits is implemented with minimum 1,530 GE.

Recently, a new block cipher, taking into account the cryptographic implications of integrated circuit (IC) printing [55], is proposed by Knudsen et al. [98]. The PRINT cipher comes in two flavors, with 48 and 96-bit block size while the effective key length equals five thirds of the actual block size. Due to the properties of IC printing, a promising technology that is still in the early stages of its development, the hardware size of the key schedule is proportional to the Hamming weight of the key. Therefore, the smallest implementation of the PRINT cipher requires only 402 GE, while the biggest one consumes 967 GE. Although providing a remarkable result assuming IC printing technology is fully mature, the PRINT cipher does not seem practical for current CMOS technology. The final layout of the PRINT cipher circuit is key-dependent and, unless the same key is embedded into millions of tags, mass production of such circuits is economically infeasible. Concerning the cryptanalysis of the PRINT cipher, Abdelraheem, Leander, and Zenner [8] show that about half of the rounds of the cipher can be successfully broken using differential cryptanalysis.

Another notable result is recent work of Poschmann et al. [135] where the GOST block cipher [130] is implemented in only 651 GE while having a key-length of 256 bits. The architecture benefits from GOST's simple key schedule and assumes that the key is fixed and cannot be changed once the circuit has been manufactured. The latest cryptanalysis result by Isobe [78] showed that a 256-bit key of the full GOST cipher can be recovered with $2^{225}$ computations and $2^{32}$ known plaintexts.

For an in-depth discussion of lightweight cryptography from the engineering perspective, we refer to Poschmann's thesis entitled 'Lightweight Cryptography – Cryptographic Engineering for a Pervasive World' [134], as well as Feldhofer's 'Low-Power Hardware Design of Cryptographic Algorithms for RFID Tags' dissertation [51].

# 4.2 KATAN & KTANTAN – A Family of Small and Efficient Hardware-Oriented Block Ciphers

## Publication Data

C. D. Cannière, O. Dunkelman, and M. Knežević, "KATAN & KTANTAN – A Family of Small and Efficient Hardware-Oriented Block Ciphers," in *Cryptographic Hardware and Embedded Systems – CHES 2009*, vol. 5747 of *Lecture Notes in Computer Science*, pp. 272–288, Springer, 2009.

### Personal Contributions

- Involved in: Feedback on hardware requirements; Hardware optimizations; Implementation; Text writing.

Our novel contribution consists of proposing a new family of very efficient hardware oriented block ciphers. The family contains six block ciphers divided into two flavors. All block ciphers share the 80-bit key size. The smallest cipher of the entire family, KTANTAN32, can be implemented in only 462 GE, while the biggest one, KATAN64, uses 1,054 GE.

While analyzing the previous solutions to the problem of having a lightweight cryptographic primitive, we have noticed that the more compact the cipher is, the more area is dedicated for storing the intermediate values and key bits. For example, in Grain [67], almost all of the 1,294 GE which are required, are used for maintaining the internal state. This phenomena also exist in DESL [100] and PRESENT [22], but to a lesser degree. This follows from a two-fold reasoning. First, stream ciphers need an internal state of at least twice the security level while block ciphers are exempt from this requirement. Second, while in stream ciphers it is possible to use relatively compact highly nonlinear combining functions, in block ciphers the use of an S-box puts a burden on the hardware requirements.

Figure 4.2 represents a view of a block cipher by looking through the eyes of a hardware designer. As already mentioned, the biggest portion of the cipher is dedicated to the memory necessary for performing the key schedule as well as for maintaining the state of the cipher. By having a key size of at least 80 bits and a block size of no less than 32 bits we can achieve a moderate security level against brute force attacks. The way security is actually achieved is determined by the smallest portion of the cipher – the datapath. In order to control the whole design, the control logic needs to be integrated as well. Having a minimized datapath and incorporating the control logic within the datapath itself seems to be a good method towards achieving area minimization of block ciphers. This is indeed the way we address the problem in our contribution to the field of lightweight cryptography.

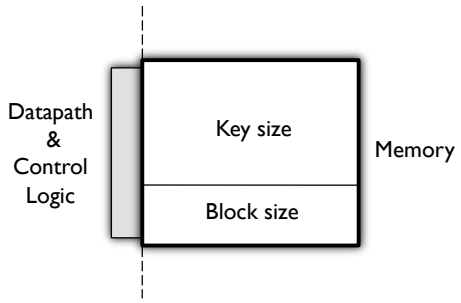


Figure 4.2: Block cipher – hardware perspective.

Another interesting issue that we have encountered during the analysis of previous results is the fact that various implementations not only differ in the basic gate technology, but also in the number of gate equivalents required for storing a bit. In the standard library we have used in this work, a simple flip-flop implementation can take between 5 and 12 GE. This, of course, depends on the type of the flip-flop that is used (scan or standard D flip-flop, with or without set/reset signals, input and output capacitance, etc). Typical flip-flops that are used to replace a combination of a multiplexer and a flip-flop are, so-called, scan flip-flops of which the most compact version, in our library, has a size equivalent to 6.25 GE. These flip-flops basically act as a combination of a simple D flip-flop and a MUX2to1. Using this type of flip-flops is beneficial both for area and power consumption.

In order to avoid any ambiguity we provide Table 4.1 with the area requirements of the selected standard cells available in our UMC 0.13 $\mu$m library. The library contains many other cells and we only outline ones that are of interest to us. The size of the cells varies depending mainly on the driving strength of the cell. A synthesis tool will therefore choose an appropriate cell depending on many factors, e.g. speed constraints, physical area constraints, fan-in, fan-out, length of the wires, etc.

Furthermore, we notice that in PRESENT [22], the 80-bit key is stored in an area of about 480 GE, i.e. about 6 GE for one bit of memory, while in DESL, the 64-bit state is stored in 780 GE (about 12 GE for a single bit). As we have already discussed, this is related to many different factors such as the type of flip-flops, technology, library, etc. Finally, we note that in some cases (which do not necessarily fit an RFID tag due to practical reasons) it is possible to reduce the area required for storing one memory bit to only 8 transistors (i.e. about 2 GE). This approach was presented by Mentens et al. [117].

An additional issue which we observed is that in many low-end applications, the key is loaded once to the device and is never changed. In such instances, it should

Table 4.1: Area requirements of selected standard cells in our UMC 0.13 $\mu$m library (FSC0L_D).

| Standard cell | Number of inputs | Area [$\mu m^2$] | Area [GE] |
|---|---|---|---|
| NOT | 1 | 3 – 28 | 0.75 – 7 |
| NAND | 2 | 4 – 23 | 1 – 5.75 |
| | 3 | 6 – 14 | 1.5 – 3.5 |
| | 4 | 12 – 18 | 3 – 4.5 |
| NOR | 2 | 4 – 40 | 1 – 10 |
| | 3 | 6 – 13 | 1.5 – 3.25 |
| | 4 | 11 – 19 | 2.75 – 4.75 |
| AND | 2 | 5 – 19 | 1.25 – 4.75 |
| | 3 | 7 – 16 | 1.75 – 4 |
| | 4 | 10 – 33 | 2.5 – 8.25 |
| OR | 2 | 5 – 25 | 1.25 – 6.25 |
| | 3 | 7 – 26 | 1.75 – 6.5 |
| XOR | 2 | 11 – 16 | 2.75 – 4 |
| | 3 | 22 – 26 | 5.5 – 6.5 |
| | 4 | 30 – 31 | 7.5 – 7.75 |
| MUX | 2 | 9 – 28 | 2.25 – 7 |
| | 3 | 16 – 27 | 4 – 6.75 |
| | 4 | 25 – 35 | 6.25 – 8.75 |
| D Flip Flop | 1 | 20 – 40 | 5 – 10 |
| Scan Flip Flop | 1 | 25 – 47 | 6.25 – 11.75 |

be possible to provide an encryption solution which can handle a key which is not stored in volatile memory, in a more efficient manner.

A final issue related to reducing the area requirements of the cipher is the block size. By decreasing the block size, it is possible to further reduce the memory complexity of the cipher. On the other hand, reducing the plaintext size to less than 32 bits has strong implications on the security of the systems using this cipher. For example, due to the birthday bound, a cipher with block size smaller than 32 bits is distinguishable from a family of random permutations after $2^{16}$ blocks.

The life span of a simple RFID tag indeed fits this restriction, but some RFID tags and several devices in sensor networks may need to encrypt larger amounts of data (especially if the used protocols require the encryption of several values in each execution). Thus, we decided to offer 3 block sizes to implementers — 32 bits, 48 bits, and 64 bits.

Our specific design goals are as follows:

- For an $n$-bit block size, no differential characteristic with probability greater than $2^{-n}$ exists for 128 rounds (about half the number of rounds of the cipher).
- For an $n$-bit block size, no linear approximation with bias greater than $2^{-n/2}$ exists for 128 rounds.
- No related-key key-recovery or slide attack with time complexity smaller than $2^{80}$ exists on the entire cipher.
- High enough algebraic degree for the equation describing half the cipher to thwart any algebraic attack.

We note that the first two conditions ensure that no differential-linear attack (or a boomerang attack) exist for the entire cipher as well. We also had to rank the possible design targets as follows:

- Minimize the size of the implementation.
- Keeping the critical path as short as possible.
- Increase the throughput of the implementation (as long as the increase in the foot print is small).
- Decrease the power consumption of the implementation.

### 4.2.1 General Construction and Building Blocks

Following the design of KeeLoq [118], we decided to adopt a cipher whose structure resembles a stream cipher. To this extent we have chosen a structure which resembles Trivium [29], or more precisely, its two register variant Bivium as the base for the block cipher. While the internal state of Trivium consists of 288 bits to overcome the fact that in each round, one bit of internal state is revealed, for the block cipher this extra security measure is unnecessary. Hence, we select the block size and the internal state of the cipher to be equal.

The structure of the KATAN & KTANTAN ciphers is very simple — the plaintext is loaded into two registers (whose lengths depend on the block size). Each round, several bits are taken from the registers and enter two nonlinear Boolean functions. The outputs of the Boolean functions are loaded in the least significant bits of the registers (after they were shifted). Of course, this is done in an invertible manner. To ensure sufficient mixing, 254 rounds of the cipher are executed.

We have devised several mechanisms used to ensure the security of the cipher, while maintaining a small foot print. The first one is the use of an LFSR instead of a

counter for counting the rounds and to stop the encryption after 254 rounds. As there are 254 rounds, an 8-bit LFSR with as sparse polynomial feedback can be used. The LFSR is initialized with some state, and the cipher has to stop running the moment the LFSR arrives to some predetermined state.

We have implemented the 8-bit LFSR counter, and the result fits in a gate equivalent of 60 GE, whilst using an 8-bit counter (the standard alternative) took 80 GE. Moreover, the expected speed of the LFSR (i.e. the critical path) is shorter than the one for the 8-bit counter.

Another advantage of using an LFSR is the fact that when considering one of the bits taken from it, we expect a sequence which keeps on alternating between 0's and 1's in a more irregular manner than in a counter (of course the change is linear). We use this feature to enhance the security of our block ciphers as described later.

One of the problems that may arise in such a simple construction is related to self-similarity attacks such as the slide attacks. For example, in KeeLoq [118] the key is used again and again. This made KeeLoq susceptible to several slide attacks (see for example the work of Curtois et al. [37] and Indesteege et al. [73]). A simple solution to this problem is to have the key loaded into an LFSR with a primitive feedback polynomial (thus, altering the subkeys used in the cipher). This solution helps the KATAN family to achieve security against the slide attack.

While the above building block is suitable when the key is loaded into memory, in the KTANTAN family, it is less favorable (as the key is hardcoded in the device). Thus, the only means to prevent a slide attack is by generating a simple, non-repetitive sequence of bits from the key. To do so, we use the "round counter" LFSR, which produces easily computed bits, that at the same time follow a non-repetitive sequence.

The third building block which we use prevents the self-similarity attacks and increases the diffusion of the cipher. The cipher actually has two (very similar but distinct) round functions. The choice of the round function is made according to the most significant bit of the round-counting LFSR. This irregular update also increases the diffusion of the cipher, as the nonlinear update affects both the differential and the linear properties of the cipher.

Finally, both KATAN and KTANTAN were constructed such that an implementation of the 64-bit variants can support the 32-bit and the 48-bit variants at the cost of small extra controlling hardware. Moreover, given the fact that the way the key is stored and the subkeys are derived is the only difference between a KATAN$n$ and KTANTAN$n$ cipher, it is possible to design a very compact circuit that supports all six ciphers.

### 4.2.2 The KATAN Set of Block Ciphers

The KATAN ciphers come in three variants: KATAN32, KATAN48 and KATAN64. All the ciphers in the KATAN family share the key schedule which accepts an 80-bit key and 254 rounds as well as the use of the same nonlinear functions.

We start by describing KATAN32, and describe the differences of KATAN48 and KATAN64 later. KATAN32, the smallest of this family has a plaintext and ciphertext size of 32 bits. The plaintext is loaded into two registers $L_1$, and $L_2$ (of respective lengths 13 and 19 bits) where the least significant bit of the plaintext is loaded in bit 0 of $L_2$, whilst the most significant bit of the plaintext is loaded in bit 12 of $L_1$. In each round, $L_1$ and $L_2$ are shifted to the left (bit $i$ is shifted to position $i+1$), where the new computed bits are loaded in the least significant bits of $L_1$ and $L_2$. After 254 rounds of the cipher, the contents of the registers are then exported as the ciphertext (where bit 0 of $L_2$ is the least significant of the ciphertext).

KATAN32 uses two nonlinear functions $f_a(\cdot)$ and $f_b(\cdot)$ in each round. The nonlinear functions $f_a$ and $f_b$ are defined as follows:

$$f_a(L_1) = L_1[x_1] \oplus L_1[x_2] \oplus (L_1[x_3] \cdot L_1[x_4]) \oplus (L_1[x_5] \cdot IR) \oplus k_a$$

$$f_b(L_2) = L_2[y_1] \oplus L_2[y_2] \oplus (L_2[y_3] \cdot L_2[y_4]) \oplus (L_2[y_5] \cdot L_2[y_6]) \oplus k_b$$

where $IR$ is an irregular update rule (i.e. $L_1[x_5]$ is XORed in the rounds where the irregular update is used), and $k_a$ and $k_b$ are the two subkey bits. For round $i$, $k_a$ is defined to be $k_{2i}$, whereas $k_b$ is $k_{2i+1}$. The selection of the bits $\{x_i\}$ and $\{y_j\}$ are defined for each variant independently, and listed in Table 4.2.

After the computation of the nonlinear functions, the registers $L_1$ and $L_2$ are shifted, where the MSB falls off (into the corresponding nonlinear function), and the LSB is loaded with the output of the second nonlinear function, i.e. after the round the LSB of $L_1$ is the output of $f_b$, and the LSB of $L_2$ is the output of $f_a$.

The key schedule of the KATAN32 cipher (and the other two variants KATAN48 and KATAN64) loads the 80-bit key into an LFSR (the least significant bit of the key is loaded in position 0 of the LFSR). Each round, positions 0 and 1 of the LFSR are generated as the round's subkey $k_{2i}$ and $k_{2i+1}$, and the LFSR is clocked twice. The feedback polynomial that was chosen is a primitive polynomial with minimal hamming weight of 5 (there are no primitive polynomials of degree 80 with only 3 monomials):

$$x^{80} + x^{61} + x^{50} + x^{13} + 1 \ .$$

We note that these locations compose a full difference set, and thus, are less likely to lead to guess and determine attacks faster than exhaustive key search.

In other words, let the key be $K$, then the subkey of round $i$ is $k_a||k_b = k_{2\cdot i}||k_{2\cdot i+1}$ where

$$k_i = \begin{cases} K_i & \text{for } i = 0 \dots 79 \ , \\ k_{i-80} \oplus k_{i-61} \oplus k_{i-50} \oplus k_{i-13} & \text{otherwise} \ . \end{cases}$$

The differences between the various KATAN ciphers are:

- The plaintext/ciphertext size,
- The lengths of $L_1$ and $L_2$,
- The position of the bits which enter the nonlinear functions,
- The number of times the nonlinear functions are used in each round.

While the first difference is obvious, we define in Table 4.2 the lengths of the registers and the positions of the bits which enter the nonlinear functions used in the ciphers. The selection of the bits $\{x_i\}$ and $\{y_j\}$ are defined for each variant independently, and are listed in Table 4.2.

For KATAN48, in one round of the cipher the functions $f_a$ and $f_b$ are applied twice. The first pair of $f_a$ and $f_b$ is applied, and then after the update of the registers, they are applied again, using the same subkeys. Of course, an efficient implementation can implement these two steps in parallel. In KATAN64, each round applies $f_a$ and $f_b$ three times (again, with the same key bits).

Table 4.2: Parameters defined for the KATAN/KTANTAN family of ciphers.

| Cipher | $\lvert L_1\rvert$ | $\lvert L_2\rvert$ | $x_1$ | $x_2$ | $x_3$ | $x_4$ | $x_5$ |
|---|---|---|---|---|---|---|---|
| KATAN32/KTANTAN32 | 13 | 19 | 12 | 7 | 8 | 5 | 3 |
| KATAN48/KTANTAN48 | 19 | 29 | 18 | 12 | 15 | 7 | 6 |
| KATAN64/KTANTAN64 | 25 | 39 | 24 | 15 | 20 | 11 | 9 |
| Cipher | $y_1$ | $y_2$ | $y_3$ | $y_4$ | $y_5$ | $y_6$ | |
| KATAN32/KTANTAN32 | 18 | 7 | 12 | 10 | 8 | 3 | |
| KATAN48/KTANTAN48 | 28 | 19 | 21 | 13 | 15 | 6 | |
| KATAN64/KTANTAN64 | 38 | 25 | 33 | 21 | 14 | 9 | |

We outline the structure of KATAN32 (which is similar to the round function of any of the KATAN variants or the KTANTAN variants) in Fig. 4.3.

Finally, specification-wise, we define the counter which counts the number of rounds. The round-counter LFSR is initialized to the all 1's state, and clocked once using

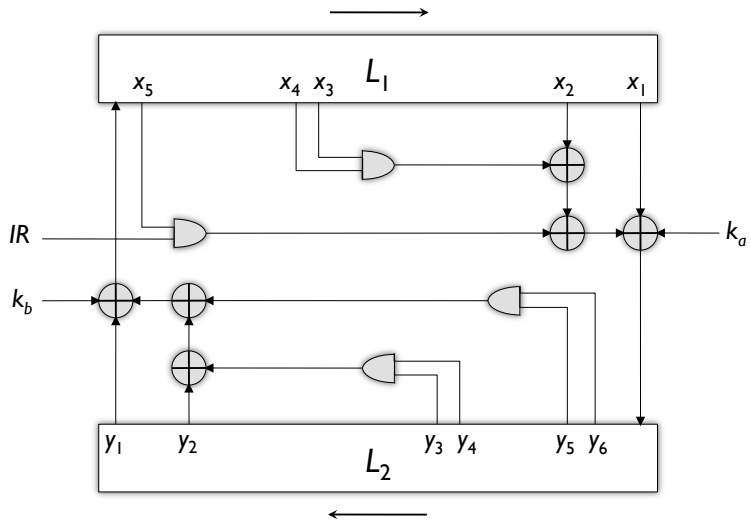


Figure 4.3: The outline of a round of the KATAN/KTANTAN ciphers.

the feedback polynomial $x^8 + x^7 + x^5 + x^3 + 1$. Then, the encryption process starts, and ends after 254 additional clocks when the LFSR returns to the all 1's state. As mentioned earlier, we use the most significant bit of the LFSR to control the irregular update (i.e. as the $IR$ signal). For sake of completeness, in Table 4.3 we provide the sequence of irregular rounds.

Table 4.3: The sequence of the irregular updates; $IR = 1$ means that the irregular update rule is used in this round, while $IR = 0$ means that this is not the case.

| | | | | |
|---|---|---|---|---|
| $IR_{0-39}$ | 1111111000 | 1101010101 | 1110110011 | 0010100100 |
| $IR_{40-79}$ | 0100011000 | 1111000010 | 0001010000 | 0111110011 |
| $IR_{80-119}$ | 1111010100 | 0101010011 | 0000110011 | 1011111011 |
| $IR_{120-159}$ | 1010010101 | 1010011100 | 1101100010 | 1110110111 |
| $IR_{160-199}$ | 1001011011 | 0101110010 | 0100110100 | 0111000100 |
| $IR_{200-239}$ | 1111010000 | 1110101100 | 0001011001 | 0000001101 |
| $IR_{240-253}$ | 1100000001 | 0010 | – | – |

We note that due to the way the irregular update rule is chosen, there are no sequences of more than 7 rounds that share the pattern of the regular/irregular updates. This ensures that any self-similarity attack cannot utilize more than 7 rounds of the same function (even if the attacker chooses keys that suggest the

same subkeys). Thus, it is easy to see that such attacks are expected to fail when applied to the KATAN family.

### 4.2.3 The KTANTAN Family

The KTANTAN family is very similar to the KATAN family except of the key schedule. While in the KATAN family, the 80-bit key is loaded into a register which is then repeatedly clocked, in the KTANTAN family of ciphers, the key is burnt (i.e. fixed) and the only possible "flexibility" is the choice of subkey bits. Thus, the design problem in the KTANTAN ciphers is choosing a sequence of subkeys in a secure, yet efficient manner.

In order to minimize the hardware size, while maintaining the throughput, we treat the key as 5 words of 16 bits each. From each 16-bit word we pick the same bit (using a MUX16to1) according to the four most significant bits of the round controlling LFSR. Then, out of the five bits, we choose one using the four least significant bits of the round-counting LFSR.

Formally, let $K = w_4||w_3||w_2||w_1||w_0$, where the least significant bit of $w_0$ is the least significant bit of $K$, and the most significant bit of $w_4$ is the most significant bit of $K$. We denote by $T$ the round-counting LFSR (where $T_7$ is the most significant bit), then, let $a_i = \text{MUX16to1}(w_i, T_7T_6T_5T_4)$, where $\text{MUX16to1}(x, y)$ gives the $y$-th bit of $x$. Then, the key bits which are used are

$$k_a = \overline{T_3} \cdot \overline{T_2} \cdot (a_0) \oplus (T_3 \vee T_2) \cdot \text{MUX4to1}(a_4a_3a_2a_1, T_1T_0)$$

$$k_b = \overline{T_3} \cdot T_2 \cdot (a_4) \oplus (T_3 \vee \overline{T_2}) \cdot \text{MUX4to1}(a_3a_2a_1a_0, \overline{T_1T_0})$$

(where $\text{MUX4to1}(x, y)$ is a MUX with 4 input bits and 1 output bit).

When considering $k_a$ or $k_b$, of the 80-bit key, only one bit is used only twice, 15 are used four times, and the remaining 64 bits are used 3 times (but in total each key bit is used at least 5 times). Moreover, even if an attacker tries to pick two keys which realize the same subkey sequence for either $k_a$ or $k_b$, the maximal length of such a sequence for either $k_a$ or $k_b$ is 35 rounds (i.e. necessarily after 35 rounds the sequences differ). We also note that due to the irregular update, during these 35 rounds, the round function is going to be different in any case.

The last issue concerning the KTANTAN key schedule is finding the most efficient way to implement it. One possible solution is to have the entire selection logic in one round. This approach requires 5 parallel MUX16to1 and our hardware implementations show that the total area consumed by the MUXes is about 180 GE. A second approach is to use one MUX16to1 and re-use it over 5 clock cycles. At first glance, this approach may lead to a smaller circuit (while the implementation is slower). However, due to the cost of the extra control logic, this approach is not only slower, but leads to larger circuits.

### 4.2.4 Hardware Implementation

We implement KATAN32 using Synopsys Design Compiler version Y-2006.06 and the FSC0L_D 0.13 $\mu$m standard cell CMOS library tailored for UMC's 0.13 $\mu$m Low Leakage process. Our implementation requires 802 GE, of which 742 GE are used for the sequential logic, and 60 GE are used for the combinational logic. In other words, the memory occupies 92.5 % of the total design. As already discussed, this amount of memory is the minimum requirement for designing a moderately secure, low-cost block cipher. All of the other logic, which in this case represents 7.5 % of the cipher, serves to control the design and to provide security against attacks other than brute force. At 100 kHz the cipher consumes only 381 nW, running at the speed of 12.5 kb/s. This is the gate level power estimation obtained using Synopsys Design Compiler version Y-2006.06[2].

For KATAN48 the implementation size is 927 GE (of which 842 are for the sequential logic) and the total power consumption is estimated to be 439 nW. For the 64-bit variant, KATAN64, the total area is 1,054 GE (of which 935 are for the sequential logic) and the power consumption 555 nW.

Here we would like to note that further area reduction for KATAN48 and KATAN64 is possible by utilizing a clock gating technique. As explained above, the only difference between KATAN32 on one hand and KATAN48 and KATAN64 on the other, is the number of times nonlinear functions $f_a$ and $f_b$ are applied with the same subkeys per single round. Therefore, we can clock the key register and the counter such that they are updated once in every two (three) cycles for KATAN48 (KATAN64). However, this approach reduces the throughput two (three) times respectively, and is useful only when a compact implementation is an ultimate goal. An area of 916 GE with a throughput of 9.4 kb/s (at 100 kHz) is obtained for KATAN48 and 1,027 GE with a throughput of 8.4 kb/s (at 100 kHz) for KATAN64.

At the cost of a little hardware overhead, the throughput of the KATAN family of block ciphers can be doubled or even tripled. To increase the speed of the cipher, we double (triple) the logic for the nonlinear functions $f_a$ and $f_b$ as well as the logic for the feedback coefficients of the counter and the key register. The implementation results are given in Table 4.4.

KTANTAN32 is implemented using the same FSC0L_D 0.13 $\mu$m CMOS library. Our implementation requires 462 GE, of which 244 GE are used for the sequential logic, and 218 GE are used for the combinational logic. The simulated power consumption at 100 kHz, and throughput of 12.5 kb/s is only 146 nW. For the synthesis and the power estimation we have again used the same version of Synopsys Design Compiler.

[2]Although the gate level power estimation gives a rough estimate, it is useful for comparison with related work reported in the literature.

Table 4.4: Area-Throughput trade-offs (UMC 0.13$\mu$m CMOS, Synopsys Design Compiler version Y-2006.06, synthesis results).

| Cipher | Block [bit] | Key [bit] | Size [GE] | GE per Bit of Memory | Throughput$^\star$ [kb/s] |
|---|---|---|---|---|---|
| KATAN32 | 32 | 80 | 802 | 6.25 | 12.5 |
| KATAN32 | 32 | 80 | 846 | 6.25 | 25 |
| KATAN32 | 32 | 80 | 898 | 6.25 | 37.5 |
| KATAN48$^\dagger$ | 48 | 80 | 916 | 6.25 | 9.4 |
| KATAN48 | 48 | 80 | 927 | 6.25 | 18.8 |
| KATAN48 | 48 | 80 | 1,002 | 6.25 | 37.6 |
| KATAN48 | 48 | 80 | 1,080 | 6.25 | 56.4 |
| KATAN64$^\dagger$ | 64 | 80 | 1,027 | 6.25 | 8.4 |
| KATAN64 | 64 | 80 | 1,054 | 6.25 | 25.1 |
| KATAN64 | 64 | 80 | 1,189 | 6.25 | 50.2 |
| KATAN64 | 64 | 80 | 1,269 | 6.25 | 75.3 |
| KTANTAN32 | 32 | 80 | 462 | 6.25 | 12.5 |
| KTANTAN32 | 32 | 80 | 673 | 6.25 | 25 |
| KTANTAN32 | 32 | 80 | 890 | 6.25 | 37.5 |
| KTANTAN48$^\dagger$ | 48 | 80 | 571 | 6.25 | 9.4 |
| KTANTAN48 | 48 | 80 | 588 | 6.25 | 18.8 |
| KTANTAN48 | 48 | 80 | 827 | 6.25 | 37.6 |
| KTANTAN48 | 48 | 80 | 1,070 | 6.25 | 56.4 |
| KTANTAN64$^\dagger$ | 64 | 80 | 684 | 6.25 | 8.4 |
| KTANTAN64 | 64 | 80 | 688 | 6.25 | 25.1 |
| KTANTAN64 | 64 | 80 | 927 | 6.25 | 50.2 |
| KTANTAN64 | 64 | 80 | 1,168 | 6.25 | 75.3 |

$^\star$ — A throughput is estimated for frequency of 100 kHz.
$^\dagger$ — Using clock gating.

For KTANTAN48 the implementation size of 588 GE (of which 344 are used for the sequential logic) is obtained together with an estimated power consumption of 234 nW. For the 64-bit variant, KTANTAN64, the total area is 688 GE (of which 444 are for the sequential logic) and the power consumption 292 nW. By using the clock gating as explained above, an area of 571 GE (684 GE) and a throughput of 9.4 kb/s (8.4 kb/s) for KATAN48 (KATAN64) is achieved.

Similar to the KATAN family, we can also double (triple) the throughput for all the versions of KTANTAN family. To do that, we double (triple) the number of MUX16to1, MUX4to1, round functions $f_a$ and $f_b$, and all the logic used for the feedback coefficients of the counter. Additionally, a few more gates are necessary to perform the key schedule efficiently.

While here we put emphasis on the smallest possible variants, it can be easily

seen that increasing the speed of the implementation is feasible with only a small hardware overhead. Therefore, we provide more implementation results in Table 4.4. We compare our results with previous constructions in Table 4.5. We note here that some of the implementations achieve an amazingly low gate count due to the number of GE per bit of memory used. As already discussed, this is an issue inherent not only to the encryption algorithm, but also a matter of the technology that is used.

Table 4.5: Comparison of ciphers designed for low-end environments (optimized for size).

| Cipher | Block [bit] | Key [bit] | Size [GE] | GE per Bit of Memory | Throughput$^\star$ [kb/s] | Technology [$\mu$m] |
|---|---|---|---|---|---|---|
| AES-128 [52] | 128 | 128 | 3,400 | 7.97 | 12.4 | 0.35 |
| AES-128 [62] | 128 | 128 | 3,100 | 5.8 | 0.08 | 0.13 |
| HIGHT [71] | 64 | 128 | 3,048 | N/A | 188.25 | 0.25 |
| mCrypton [109] | 64 | 64 | 2,420 | 5 | 492.3 | 0.13 |
| DES [100] | 64 | 56 | 2,309 | 12.19 | 44.4 | 0.18 |
| DESL [100] | 64 | 56 | 1,848 | 12.19 | 44.4 | 0.18 |
| PRESENT-80 [22] | 64 | 80 | 1,570 | 6 | 200 | 0.18 |
| PRESENT-80 [145] | 64 | 80 | 1,000 | N/A | 11.4 | 0.35 |
| MIBS-64 [79] | 64 | 64 | 1,396 | 6 | 200 | 0.18 |
| MIBS-80 [79] | 64 | 80 | 1,530 | 6 | 200 | 0.18 |
| TEA [179] | 64 | 128 | 2,335 | N/A | N/A | N/A |
| SEA [111] | 96 | 96 | 3,758 | N/A | 103 | 0.13 |
| Grain [57] | 1 | 80 | 1,294 | 7.25 | 100 | 0.13 |
| Trivium [117] | 1 | 80 | 749 | $2^\Diamond$ | $100^\dagger$ | 0.35 |
| KATAN32 | 32 | 80 | 802 | 6.25 | 12.5 | 0.13 |
| KATAN48 | 48 | 80 | 927 | 6.25 | 18.8 | 0.13 |
| KATAN64 | 64 | 80 | 1,054 | 6.25 | 25.1 | 0.13 |
| KTANTAN32 | 32 | 80 | 462 | 6.25 | 12.5 | 0.13 |
| KTANTAN48 | 48 | 80 | 588 | 6.25 | 18.8 | 0.13 |
| KTANTAN64 | 64 | 80 | 688 | 6.25 | 25.1 | 0.13 |

$^\star$ — A throughput is estimated for frequency of 100 kHz.
$^\dagger$ — This throughput is projected, as the chip requires higher frequencies.
$^\Diamond$ — This is a full-custom design using $C^2$MOS dynamic logic.

One may argue that further area reduction comes by serializing input/output of the plaintext (and the key for KATAN)/ciphertext. While this seems to be a good approach at first glance, one must not ignore the throughput degradation in this case. Furthermore, one needs to introduce the clock gating and be able to control such an implementation in an efficient way, not introducing more overhead than savings by using this method.

### 4.2.5 Security Analysis

Our design philosophy was based on providing a very high level of security. To do so, we designed the ciphers with very large security margins. For example, as a design target we have set an upper bound for the differential probability of any 128-round differential characteristic at $2^{-n}$ for an $n$-bit block size.

#### Differential and Linear Cryptanalysis

We have analyzed all ciphers under the assumption that the intermediate encryption values are independent. While this assumption does not necessarily hold, it simplifies the analysis and is not expected to change the results too much. Moreover, in our analysis we always take a "worst case" approach, i.e. we consider the best scenario for the attacker, which most of the times does not happen. Hence, along with the large security margins, even if the assumption does not hold locally, it is expected that our bounds are far from being tight.

To simplify the task of identifying high probability differentials, we used computer-aided search. Our results show that depending on the used rounds, the best 42-round differential characteristic for KATAN32 has probability of $2^{-11}$ (it may even be lower for different set of rounds). Hence, any 126-round differential characteristic must have probability no more than $(2^{-11})^3 = 2^{-33}$. Similar results hold for linear cryptanalysis (the best 42-round linear approximation has a bias of $2^{-6}$, i.e. a bias of $2^{-16}$ for 126-round approximation).

For KATAN48, the best 43-round differential characteristic has probability of at most $2^{-18}$. Hence, any 129-round differential characteristic has probability of at most $(2^{-18})^3 = 2^{-54}$. As the probability of an active round is at least $2^{-4}$ this actually proves that our design criteria for 128-round differential characteristics is satisfied. The corresponding linear bias is $2^{-10}$ (for 43 rounds) or $2^{-28}$ (for 129 rounds).

Finally, repeating the analysis for KATAN64, our computer-aided search found that the best 37-round differential characteristic has probability $2^{-20}$. Hence, any 111-round differential characteristic has probability of at most $2^{-60}$, along with the fact that the best 18-round differential characteristic has probability of at most $2^{-5}$, then the best 129-round differential characteristic has probability of no more than $2^{-65}$. The linear bounds are $2^{-11}$ for 37 rounds and $2^{-31}$ for 111 rounds.

Hence, we conclude that the KATAN family is secure against differential and linear attacks. As there is no difference between the KATAN and the KTANTAN families with respect to their differential and linear behaviors, then the above is also true for the KTANTAN family.

### 4.2.6 Combined Attacks

As shown in the previous section, the probability of any differential characteristic of 128 rounds can be bounded by $2^{-n}$ for KATAN$n$. Moreover, even for 64 rounds, there are no "good" characteristics. Hence, when trying to combine these together, we do not expect to obtain good combined attacks.

For example, consider a differential-linear approximation. As noted before, the differential characteristic of 42-round KATAN32 has probability at most $2^{-11}$. The bias of a 42-round KATAN32 is at most $2^{-6}$. Hence, the best differential-linear property for 120 rounds is expected to have bias of at most $2 \cdot 2^{-11} \cdot (2^{-6})^2 = 2^{-22}$ (we assume a worst case assumption that allows the attacker to gain some free rounds in which the differential is truncated). Of course, an attacker may try to construct the differential-linear approximation using a different division of rounds. However, as both the probability and bias drop at least exponentially with the number of rounds, a different division is not expected to lead to better differential-linear approximations.

The same holds for the (amplified) boomerang attack. The attack (just like the differential-linear attack) treats the cipher as composed of two sub-ciphers. The probability of constructing a boomerang quartet is $\hat{p}^2\hat{q}^2$, where $\hat{p} = \sqrt{\sum_\beta \Pr^2[\alpha \to \beta]}$ where $\alpha$ is the input difference for the quartet, and $\beta$ is the output difference of the characteristic in the first sub-cipher. Again, as $\hat{p}^2 \leq \max_\beta \Pr[\alpha \to \beta]$ which is bounded at $2^{-22}$ for 84-round KATAN32. The same goes with respect to $\hat{q}$, and thus, the probability of a boomerang quartet in 128-round KATAN32 is at most $2^{-44}$.

The same rationale can be applied to KATAN48 and KATAN64, obtaining similar bounds. Specifically, the bounds for differential-linear bias is $2^{-37}$ (for 140 rounds) and $2^{-50}$ (for 160 rounds), respectively. The bounds for constructing a boomerang quartet for 128 rounds are $2^{-54}$ and $2^{-65}$, respectively.

Another combined attack which may be considered is the impossible differential attack. This attack is based on finding a differential which has probability zero of as many rounds as possible. The most common way to construct such a differential is in a miss-in-the-middle manner, which is based on finding two (truncated) differentials with probability 1 which cannot co-exist. Due to the quick diffusion, changing even one bit would necessarily affect all bits after at most 42 rounds (37 for KATAN48 and 38 for KATAN64), and thus, there is no impossible differential of more than 168 rounds (after 42 rounds, a change of any bit may affect all bits, and thus, after 84 rounds, each differential may have any output difference).

Hence, we conclude that the KATAN family (as well as the KTANTAN family) of block ciphers is secure against combined attacks.

### Slide and Related-Key Attacks

As mentioned before, the KATAN & KTANTAN family was designed to foil self-similarity attacks by using two types of rounds which are interleaved in a non-repeating manner. First, consider the slide attack, which is based on finding two messages such that they share most of the encryption process (which are some rounds apart). Given the fact that there is a difference between the deployed round functions, this is possible only for a very small number of rounds, even if we allow these relations to be probabilistic in nature (i.e. assume that the bit of the intermediate value is set to 0 thus preventing the change in the function to change the similarity between the states). For example, when considering KATAN32, there is no slide property with probability $2^{-32}$ starting from the first round of the cipher. The first round from which such a property can be constructed is round 19. If an attacker achieves the same intermediate encryption value after round 19 and round 118, he may find a "slid" pair which maintains the equality with probability $2^{-31}$ until the end of the cipher (i.e. the output of the second encryption process will be the same as the intermediate encryption value of the first encryption at round 155). This proves that there are no good slid properties in the cipher family (we note that this probability is based on the assumption that the subkeys are the same, which is not the case, unless the key is the all zeros key). When it comes to KATAN48 or KATAN64, this probability is even lower (as there are more bits which need to be equal to zero), i.e. $2^{-62}$ and $2^{-93}$, respectively, rendering slide attacks futile against the KATAN & KTANTAN family (these values are actually an upper bound as they assume that all the subkeys are the same).

Now consider a related-key attack. In the related-key setting, the attacker searches for two intermediate encryption values as well as keys which develop in the same manner for as many rounds as possible. As noted before, there are no "good" relations over different rounds, which means that the two intermediate encryption values have to be in the same round. However, by changing even one single bit of the key causes a difference after at most 80 rounds of the similar encryption process. Hence, no related-key plaintext pairs (or intermediate encryption values) exist for more than 80 rounds (similarity in 80 rounds would force the key and the intermediate encryption value to be the same). As this is independent of the actual key schedule algorithm, it is easy to see that both KATAN and KTANTAN are secure against this attack.

The only attack in this category which remains is a related-key differential attack. This is the only attack where there is a difference between the two families of ciphers according to their key schedule algorithm. We first consider the case of the KATAN family. The key schedule of the KATAN family expands linearly the 80-bit key into 508 subkey bits (each is used once in KATAN32, twice in KATAN48, and thrice in KATAN64). We note that the probability of the differential is reduced any time a difference enters one of the nonlinear functions (i.e. the AND operation).

Thus, it is evident that good related-key differentials have as little active bits as possible. Moreover, we can relate the number of active bits throughout the encryption process to the issue of active bits in the key schedule. Each active bit of the subkey (i.e. a subkey bit with a difference) either causes a difference in the internal state (which in turn incurs probability and activation of more bits), or is being canceled by previous differences. We note that each active subkey bit which is not canceled, necessarily induces a probability "penalty" of $2^{-2}$ in KATAN32, $2^{-4}$ in KATAN48, and $2^{-6}$ in KATAN64. Moreover, due to the way the cipher works, each active bit can "cancel" at most four other active subkey bits.[3] Hence, if the weight of the expanded subkey difference is more than 80, then it is assured that the probability of any related-key differential of KATAN32 is at most $2^{-32}$ (this follows from the fact that each active bit in the intermediate encryption value may cancel up to four subkey bit differences injected, and we shall assume a worst case assumption that the positions align "correctly"). For KATAN48, due to the increased penalty, it is sufficient that the minimal weight of the expanded subkey difference is more than 60, and for KATAN64 the minimal weight needs to be at least 54. We have analyzed the minimal weight using the MAGMA software package, and the current bounds are between 72 and 84. Hence, we conclude that the KATAN family of block ciphers is expected to be resistant to related-key differential attacks.

For the KTANTAN family, due to the fixed key, the concept of related-key attacks is of theoretical interest. Still, we can follow a more detailed analysis using the same ideas as we used for regular differential searches. While the search space is huge, our current results show that there is no related-key differential characteristic for more than 150 rounds of KTANTAN32 with probability greater than $2^{-32}$. Similar results are expected to hold for KTANTAN48 and KTANTAN64.

### Cube Attacks and Algebraic Attacks

Given the low algebraic degree of the combining function, it may look as if KATAN and KTANTAN are susceptible to algebraic attacks or the cube attack [45]. However, when considering the degree of the expressions involving the plaintext, one can see that after 32 rounds (for KATAN32) the degree of each internal state bit is at least 2, which means that after 160 rounds, the degree of each internal state bit can reach 32. For KATAN48, the degree is at least 2 after 24 rounds, (or about 48 after 144 rounds), and for KATAN64 it is 2 after 22 rounds and can reach 64 after 132 rounds). Hence, as the degree can reach the maximal possible value (and there are some more rounds to spare), it is expected that the KATAN & KTANTAN family is secure against algebraic attacks.

[3] We note that five or six can be canceled, but in this case, the probability penalty of an active bit is increased by more than the "gain" offered by using this active bit more times

Another possible attack is the cube attack, which was successful against reduced-round variants of Trivium (with less initialization rounds than in Trivium). We note that in Trivium the internal state is clocked four full cycles (i.e. each bit traverses all the registers exactly four times). In KATAN32, most bits traverse the registers eight times (where a few do so only seven times), going through more nonlinear combiners (each Trivium round uses only one AND operation per updated bit), and thus is expected to be more secure against this attack than Trivium. The same is also true for KATAN48 (where about half of the bits traverse the registers 10 times, and the other bits do so 11 times) and KATAN64 (where most of the bits traverse the registers 12 times, and a few do only 11 times).

### 4.2.7 Cryptanalysis of KATAN & KTANTAN Family of Block Ciphers

Here, we outline the cryptanalytical results of the KATAN & KTANTAN family of block ciphers that have been published recently.

The most notable result comes from Bogdanov and Rechberger [23] who exploit the weaknesses in the KTANTAN key schedule. By using a variant of the meet-in-the-middle attack on block ciphers, the authors report an attack of time complexity $2^{75.170}$ encryptions on the full KTANTAN32 cipher with only 3 plaintext/ciphertext pairs. Furthermore, they apply the same attack to KTANTAN48 and KTANTAN64, resulting in time complexity $2^{75.044}$ and $2^{75.584}$ encryptions with 2 plaintext/ciphertext pairs, respectively. Due to the weaknesses of the KTANTAN key schedule, the same attack is not applicable to the KATAN family.

Knellwolf et al. [90] apply a conditional differential cryptanalysis of non-linear feedback shift register (NLFSR) based cryptosystems. As a result, they report an attack on KATAN32 reduced to 78 out of 254 rounds, recovering at least two key bits with probability almost one and complexity $2^{22}$. For KATAN48 the results are recovering one key bit and the sum of two key bits after 70 out of 254 rounds with a complexity of $2^{34}$. The attack on KATAN64 results in recovering one key bit and the sum of two key bits after 68 out of 254 rounds with a complexity of $2^{35}$. Finally, the analysis of KATAN directly translates to the KTANTAN family and results in a similar outcome.

Bard et al. [17] perform algebraic, AIDA/cube, and side channel analysis of the KATAN & KTANTAN family of block ciphers. Their cube attacks reach 60, 40, and 30 rounds, while with the algebraic attacks they break 79, 64, and 60 rounds of KATAN32, KATAN48, and KATAN64, respectively. Furthermore, by performing side channel attacks with one-bit information leakage from the internal state, they show how to break the full 254-round unprotected KATAN32.

### 4.2.8 New Key Schedule for KTANTAN Family of Block Ciphers

## Publication Data

A. Bogdanov, O. Dunkelman, M. Knežević, C. Rechberger, and I. Verbauwhede, "On the Key Schedules of Lightweight Block Ciphers for RFID Applications," submitted to *IEEE Communications Letters*, 3 pages, 2011.

#### Personal Contributions

- Involved in: Space exploration for the new key schedule; Hardware optimizations; Implementation; Text writing.

In order to thwart the variant of the meet-in-the-middle attack proposed by Bogdanov and Rechberger [23], we propose a tweaked key schedule of KTANTAN which results in a small hardware overhead, yet providing enhanced security of the cipher. Therefore, we first provide a brief overview of the attack.

#### Overview of the Attack

The basic idea of the attack is as follows. The starting observation is that some key bits of $K$ are not used in the first $S_1$ rounds (i.e. they are neutral with respect to those rounds). Similarly, another set of key bits is not used in the last $S_2$ rounds. The attack proceeds by looping through those identified key bits independently in a forward and backward computation. As both chunks $S_1$ and $S_2$ do not cover the full cipher, in addition, the matching is done in a separate partial matching phase of the middle $R - S_1 - S_2$ rounds transformation from two fully determined states.

#### Necessary Condition for Bitwise Key Schedules

For the proposed attack on KTANTAN, it suffices to consider single bits at a time. However, a generalization is possible that applies to our new design and would consider all used sums of bits in a number of consecutive rounds. Similarly to neutral bits, in this context, a *sum-neutral* key bit with respect to these rounds is defined as a key bit that appears in a sum only if at least one other fixed key bit occurs in this sum.

Here, we derive several useful security metrics with respect to MITM attacks, concerning both neutral and sum-neutral bits, and show how they can be used to define a necessary condition for the security of a cipher. The security metrics are as follows:

- Upper bound on the maximal number $\pi_1$ of consecutive rounds that *can* be covered by any partial-matching phase (under reasonable assumptions).
- Lower bound (over all possible neutral bits) on the number of middle rounds $\pi_2$ that at least *need* to be covered by a partial-matching phase given a particular key schedule.
- A generalization of the above point. Lower bound (over all possible sum-neutral bits) on the number of middle rounds $\pi_2'$ that at least *need* to be covered by a partial-matching phase given a particular key schedule.

Now we are ready to formulate the necessary condition for a key schedule protecting against our attack and some of its generalizations:

**Condition 1.** *The number of consecutive rounds that the attacker* can *cover by any partial-matching phase must be smaller than the number of middle rounds that the attacker would at least* need *to cover with neutral or sum-neutral bits,* $\pi_1 < \pi_2'$ *and* $\pi_1 < \pi_2$.

### New Key Schedule for KTANTAN

An 8-bit round counting LFSR is used to control the key schedule. It is defined by the feedback polynomial

$$\zeta^8 + \zeta^7 + \zeta^5 + \zeta^3 + 1$$

and its initial state is all ones. Let $l_{r,7}l_{r,6}...l_{r,1}l_{r,0}$ denote the 8-bit state of the LFSR in round $r$. Each round involves two key bits. The key schedule of KTANTAN chooses two bits of $K$ in each round. This is done by applying two layers of MUX logic. First, $K$ is divided into 5 chunks $W_i$ of 16 bits each: $K = W_4||W_3||W_2||W_1||W_0$. One bit out of each chunk is selected:

$$\omega_{r,i} = \text{MUX16to1}(W_i, l_{r,7}l_{r,6}l_{r,5}l_{r,4}), i = 0, \ldots, 4 \ ,$$

where the LFSR bits define the position in $W_i$ to choose. Second, two out of these five bits are chosen controlled by the other half of the LFSR state:

$$\begin{aligned} \kappa_{1,r} &= \overline{l_{r,3}} \cdot \overline{l_{r,2}} \cdot \omega_{r,0} \oplus (l_{r,3} \vee l_{r,2}) \cdot \text{M}_1 \\ \kappa_{2,r} &= \overline{l_{r,3}} \cdot l_{r,2} \cdot \omega_{r,4} \oplus (l_{r,3} \vee \overline{l_{r,2}}) \cdot \text{M}_2 \ , \end{aligned}$$

where $\text{M}_1 = \text{MUX4to1}(\omega_4\omega_3\omega_2\omega_1, l_{r,1}l_{r,0})$ and $\text{M}_2 = \text{MUX4to1}(\omega_3\omega_2\omega_1\omega_0, \overline{l_{r,1}l_{r,0}})$.

To construct the new key schedule we also consider two layers of the MUX logic. Similar to the original key schedule of KTANTAN, we first divide $K$ into 20 chunks $W_i$ of 4 bits each: $K = W_{19}||W_{18}||\ldots||W_1||W_0$. One bit out of each chunk is now selected as:

$$\omega_{r,i} = \text{MUX4to1}(W_i, l_{r,7}l_{r,6}), i = 0, \ldots, 19 \ ,$$

where the LFSR bits again define the position in $W_i$ to choose. In the second layer of MUX logic we use 10 MUX2to1 for $\kappa_{1,r}$ and another 10 for $\kappa_{2,r}$. The outputs of the MUX logic are now simply XORed and the key bits are obtained as:

$$\kappa_{1,r} = \bigoplus_{i=0}^{9} \text{MUX2to1}(\{\omega_{r,2i}, \omega_{r,2i+1}\}, s_{r,i})$$
$$\kappa_{2,r} = \bigoplus_{i=10}^{19} \text{MUX2to1}(\{\omega_{r,2i-20}, \omega_{r,2i-19}\}, s_{r,i}) \ ,$$

where the value $s_{r,i}$ controlling the position to be selected is given in Table 4.6.

Table 4.6: Parameter $s_{r,i}$ defines the selection of the key bits.

| $s_{r,i}$ | $\overline{l_{r,4}}$ | $\overline{l_{r,4}}$ | $l_{r,2}$ | $\overline{l_{r,6}}$ | $\overline{l_{r,5}}$ | $\overline{l_{r,3}}$ | $l_{r,6}$ | $\overline{l_{r,3}}$ | $\overline{l_{r,4}}$ | $l_{r,2}$ |
|---|---|---|---|---|---|---|---|---|---|---|
| $i$ | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| $s_{r,i}$ | $\overline{l_{r,4}}$ | $l_{r,3}$ | $l_{r,3}$ | $l_{r,3}$ | $\overline{l_{r,3}}$ | $l_{r,2}$ | $\overline{l_{r,5}}$ | $l_{r,6}$ | $\overline{l_{r,2}}$ | $l_{r,7}$ |
| $i$ | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 |

As can be seen from Table 4.6, the controlling bits of the second layer MUX logic are chosen from the 6 most significant bits of the LFSR. This approach allows for an elegant implementation of the faster versions of the cipher. On the other hand, this specific choice is obtained as the result of computer-assisted random search over many possibilities. The resulting choice is a balance between high security margin against the new attacks, and at the same time allows for good implementation characteristics such as gate count, speed, and power consumption. Figure 4.4 illustrates the new key schedule.

### Security arguments

To back up our high security margin statement, and using previously derived criteria, we can state the following. For our choice of key schedule we have $\pi_2 = 219$, and $\pi_2' = 164$. We have obtained $\pi_1 \in \{52, 46, 37\}$ for $t \in \{32, 48, 64\}$. Hence we support the resistance of this choice of key schedule against our attack and potential generalizations of it in the following way: We would need to cover at least 164 rounds in a partial-matching phase, even though we can at most cover $\{52, 46, 37\}$ rounds. Hence, even for significantly improved matching techniques compared to the known methods, the security margin remains high.

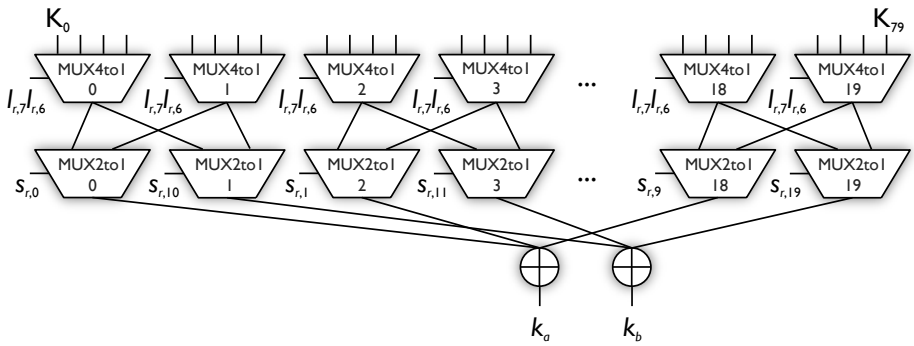


Figure 4.4: Two layers of MUXes assemble the new key schedule for KTANTAN.

### Hardware Overhead

In order to estimate the overhead of the new key schedule, we implement all three versions of the new cipher and compare them with the figures we obtained for the original designs of KTANTAN. Comparing different designs in a decent manner is only possible once the fixed benchmarking platform is established. Therefore, we implement all designs in Verilog and synthesize them using UMC 0.13 $\mu$m Low Leakage standard cell library. For the purpose of synthesis and power estimates we use Synopsys Design Compiler version C-2009.06-SP3.[4] The results are given in Table 4.7. The throughput and the power consumption are estimated for a frequency of 100 kHz.

The results show that the new key schedule introduces a small hardware overhead compared to the original design of KTANTAN. On the other hand, the new proposed design still maintains good hardware characteristics and significantly outperforms the larger members of the KATAN family.

Here, we would like to provide another, interesting observation. Please note that the area consumption of the KTANTAN family with the old key schedule slightly differs from the one reported in Section 4.2.4. The only difference during the synthesis flow in these two cases was the version of the Synopsys synthesis tool. This indeed illustrates an importance of having a fixed design flow, where factors such as synthesis tools, standard-cell library, synthesis scripts, etc. need to be fixed.

[4] We were not able to use the previous version of the synthesis tool due to the licensing issue.

Table 4.7: Hardware overhead for the new key schedule of the KTANTAN family (UMC 0.13$\mu$m CMOS, Synopsys Design Compiler version C-2009.06-SP3, synthesis results).

| Cipher | | Block [bits] | Key [bits] | Size [GE] | Throughput [kb/s] | Power [nW] |
|---|---|---|---|---|---|---|
| New Key Schedule | KTANTAN32 | 32 | 80 | 476 | 12.5 | 135 |
| | KTANTAN32$^\dagger$ | 32 | 80 | 676 | 25.0 | 181 |
| | KTANTAN32$^\ddagger$ | 32 | 80 | 875 | 37.5 | 225 |
| | KTANTAN48 | 48 | 80 | 620 | 18.8 | 189 |
| | KTANTAN48$^\dagger$ | 48 | 80 | 841 | 37.6 | 241 |
| | KTANTAN48$^\ddagger$ | 48 | 80 | 1050 | 56.4 | 295 |
| | KTANTAN64 | 64 | 80 | 759 | 25.1 | 236 |
| | KTANTAN64$^\dagger$ | 64 | 80 | 1,004 | 50.2 | 299 |
| | KTANTAN64$^\ddagger$ | 64 | 80 | 1,251 | 75.3 | 360 |
| Old Key Schedule | KTANTAN32 | 32 | 80 | 441 | 12.5 | 128 |
| | KTANTAN32$^\dagger$ | 32 | 80 | 599 | 25.0 | 155 |
| | KTANTAN32$^\ddagger$ | 32 | 80 | 748 | 37.5 | 179 |
| | KTANTAN48 | 48 | 80 | 583 | 18.8 | 175 |
| | KTANTAN48$^\dagger$ | 48 | 80 | 763 | 37.6 | 210 |
| | KTANTAN48$^\ddagger$ | 48 | 80 | 935 | 56.4 | 237 |
| | KTANTAN64 | 64 | 80 | 721 | 25.1 | 221 |
| | KTANTAN64$^\dagger$ | 64 | 80 | 927 | 50.2 | 266 |
| | KTANTAN64$^\ddagger$ | 64 | 80 | 1,121 | 75.3 | 310 |

$^\dagger$ — Doubled throughput.
$^\ddagger$ — Tripled throughput.

### 4.2.9 Summary

In this section we have presented two families of hardware-efficient encryption algorithms. To our best knowledge, the design of KATAN32 provides the best memory-datapath ratio of the block ciphers published to date. Having only 7.5 % of combinational logic, the design truly follows the design criteria of minimizing the hardware overhead. KATAN48 and KATAN64 achieve an overhead of 9.2 % and 11.3 %, respectively.

Although being smaller than its sibling, the design of the KTANTAN family provides considerably worse results in this respect. Having a hardcoded key, the KTANTAN ciphers have a relatively heavy key schedule, which occupies the biggest part of the combinational logic of the circuit. More specifically, KTANTAN32, KTANTAN48, and KTANTAN64 consist of 47.2 %, 41.5 %, and 35.5 % combinational logic, respectively. Additionally, this heavy key schedule, that was not expected to be as

strong as for KATAN, was indeed a weakness exploited in the development of an attack on the KTANTAN family as described by Bogdanov and Rechberger [23].

Once considering the possibility of hardcoding the key on a device itself, a natural solution to prevent having a weak key schedule, yet maintaining a lightweight block cipher, is to hardcode the whole, expanded key of a stronger key schedule. In the case of the KATAN & KTANTAN family, this requires hardcoding 508 bits of the KATAN expanded key instead of the initial 80. Where applicable, this might be an efficient solution. Only a round function of KATAN is to be implemented in this case, while the expanded key is to be stored in non-volatile memory. For the sake of illustration, we provide data in Table 4.8.

This approach, of course, does not only apply to the KATAN & KTANTAN family of block ciphers, but rather to any existing design. A device needs to have enough non-volatile storage to maintain the expanded key, and that is indeed one of the main constraints for low-end devices. For the sake of illustration, the length of the expanded key of AES-128 is 1280 bits, while PRESENT, for example, has an expanded key of 2048 bits.

Table 4.8: Area and memory requirements for round function and expanded key of the KATAN family (UMC 0.13$\mu$m CMOS, Synopsys Design Compiler version C-2009.06-SP3, synthesis results).

| Cipher | Round function [GE] | Memory for expanded key [bit] |
|---|---|---|
| KATAN32 | 315 | 508 |
| KATAN48 | 455 | 508 |
| KATAN64 | 595 | 508 |

## 4.3 Conclusion

By describing the design flow of the KATAN & KTANTAN family of block ciphers in this chapter, we have shown a typical trade-off one needs to consider when designing a lightweight cryptographic primitive. Reducing the key length and the block size to a minimum, such that the cipher is secure against brute force attacks, and by introducing a lightweight round function as well as a minimized key schedule, we accomplished the desired goal. The novel idea of sharing the datapath with the control logic of the cipher, as proposed in this chapter, has been adopted and used in the literature ever since.

Once entering the world of lightweight cryptography, one must not ignore the importance of physical security of the implementation. Although coming at a higher price, protection against side channel attacks is often necessary in order to protect the secrets. Following this demand, our future research will address issues of physical security of low-cost cryptographic primitives. Since both KATAN and KTANTAN have a simple structure and use very basic components, it appears that common techniques to protect the implementation should be easy to adopt.

# Chapter 5

# Conclusions and Future Work

## Conclusion

In 1965, Moore [122] predicted that the number of transistors on a chip would double about every two years. Remarkably, the industry has kept pace with this exponential growth since the invention of the first integrated circuit in 1958 [87]. However, the physical limits of atomic structures in the current CMOS technology are approaching a hard bound that cannot be overcome. As Moore pointed out in 2003, "No exponential change of a physical quantity can [...] continue forever" [123]. The International Technology Roadmap for Semiconductors (ITRS) reminds that the major challenges of the next-generation lithography will come with the appearance of 16 nm CMOS processes [77].

Predictions about the end of Moore's law have been wrong several times already. The next landmark, as stated by Intel, is 2020 when the limits of the current CMOS technology could be reached [75]. No matter whether the current CMOS technology will continue to shrink or new technologies will appear, the continuous need for speed and low power will remain the major challenge in the domain of VLSI digital design. Cryptography and its hardware implementations will continue to follow demands driven by the integrated circuits development. Efficient hardware implementations of cryptographic primitives will therefore remain an exciting research topic.

This thesis has covered techniques for efficient hardware implementations of cryptographic primitives. The proposed novelties are introduced mainly at the algorithmic level and that makes them appropriate to operate on any hardware platform. The platform-dependent optimizations at lower levels of abstraction are still possible for further improvements of the proposed techniques.

The first contribution of the thesis is a finding of several sets of moduli for faster modular multiplication, directly applicable to state of the art public-key cryptosystems. The sets are suited for both Montgomery and classical modular multiplications. Not only is the modular multiplication faster for the proposed sets, but also the precomputational phase of the algorithms is omitted. A similar approach is later extended to the bipartite modular multiplication. Moreover, a novel multiplication method called tripartite modular multiplication is also proposed in the second chapter. The algorithm represents a symbiosis of three existing algorithms, namely Barrett, Montgomery, and Karatsuba algorithms.

The second contribution of the thesis explores techniques for high-throughput hardware implementations of cryptographic hash functions. The iteration bound analysis, as well as some of the algorithm-specific techniques, is applied for designing a high-throughput architecture of the RIPEMD-160 algorithm. The throughput of 3.4 Gb/s is reached in 0.13 $\mu$m CMOS technology and, to the best of our knowledge, is the highest throughput of RIPEMD-160 reported in the literature. A methodology for a fair and consistent evaluation of fourteen second-round SHA-3 candidates is also presented in the third chapter. We proposed a platform, a design strategy and evaluation criteria in order to assure fair comparison. Using a SASEBO-GII FPGA board as a common platform, combined with a well defined hardware interface, we compared all 256-bit version candidates with respect to area, throughput, latency, power and energy consumption. Our approach defines a standard testing harness for SHA-3 candidates, including the interface specification for the SHA-3 module on our testing platform. Additionally, we have provided 90 nm CMOS ASIC synthesis results. We released online the source code of all the candidates and by using a common, fixed, publicly available platform, we made our claimed results reproducible and open for public verification. As the final contribution of Chapter 3, we proposed efficient hardware architectures for the Luffa hash algorithm, one of the fourteen second-round SHA-3 candidates.

The third contribution of the thesis is a new family of small, very efficient, hardware-oriented block ciphers – KATAN & KTANTAN family. The smallest design of the entire family, KTANTAN32, can be implemented in 462 GE while achieving encryption speed of 12.5 kb/s at 100 kHz. KATAN64, the largest and most flexible candidate of the family uses 1,054 GE and has a throughput of 25.1 kb/s. Another curiosity of the KATAN & KTANTAN family is that 92.5 % of KATAN32 consists of pure sequential logic. During the design phase of the proposed family, several novelties have been introduced. It is the first time that the actual control logic of the cipher itself is integrated into the round function of the cipher. Also, the various block sizes ranging from 32 to 64 bits have been considered. Finally, the concept of hardwiring a key on the chip is also novel with this respect and significantly helps to further reduce the area of the circuit.

## Future Work

In Chapter 2, we introduced several hardware architectures to prove the superiority of our proposed algorithms. We believe that further improvements of these architectures are possible and worth to explore. The proposed algorithms are also suitable for software implementations and it would be interesting to examine their advantages over classical algorithms on common software platforms. The tripartite algorithm is introduced in its basic form and an interesting question that remains is to determine the level of parallelism that results in optimal performance. Moreover, the tripartite algorithm is not only suited for high-speed implementations, but can also result in more compact designs. Due to the use of Karatsuba's algorithm, one can further benefit from using smaller multipliers while still preserving rather good speed performance of the circuit.

Chapter 3 dealt with high-throughput hardware implementations of hash functions. Now, that the five finalists of the SHA-3 competition are known, a natural extension of our work is to further consider algorithm-specific techniques and to compare remaining candidates using the methodology we proposed. The diversity of the candidates which advanced into the final round is still rather large. Blake and Skein are narrow-pipe and ARX based, Grøstl is wide-pipe and based on an 8-bit S-box, JH is wide-pipe and 4-bit S-box/Boolean based, and finally Keccak is sponge and 4-bit S-box/Boolean based. After algorithm-specific techniques are applied, it would certainly be interesting to find a stronger correlation between algorithmic features and the implementation results. Moreover, the power and energy estimates need to be done with more accuracy especially regarding the ASIC implementations. As already pointed out, the current candidates seem not to be suitable for very constrained devices, and therefore it would be interesting to further explore possibilities of implementing compact versions of the remaining candidates.

Finally, in Chapter 4, we introduced a new family of lightweight block ciphers and we expect the proposed novelties to be applied for the future development of compact cryptographic primitives. Some of the unveiled ideas can certainly be used for a design of lightweight hash functions and possibly further, meaning compact public-key cryptosystems. As the lightweight cryptography has become a very popular research topic, we expect to see more research following that direction. Finally, since the lightweight cryptography is meant to be used mainly in embedded devices, another big challenge that remains is to address the side channel issues of the proposed designs as well as lightweight cryptographic primitives in general.

# Bibliography


[1] ECRYPT II Yearly Report on Algorithms and Keysizes. European Network of Excellence in Cryptology II, 2010.

[2] EPC Tag Data Standard (TDS). Available at `http://www.gs1.org/gsmp/kc/epcglobal/tds`.

[3] Helion Technology. Available at `http://www.heliontech.com/index.htm`.

[4] IEEE P802.3ba 40Gb/s and 100Gb/s Ethernet Task Force. Available at `http://www.ieee802.org/3/ba/`.

[5] Secure Hash Standard. Federal Information Processing Standards Publication 180. National Institute of Standards and Technology (NIST), 1993.

[6] Secure Hash Standard. Federal Information Processing Standards Publication 180-1. National Institute of Standards and Technology (NIST), 1995.

[7] Secure Hash Standard. Federal Information Processing Standards Publication 180-2. National Institute of Standards and Technology (NIST), 2003.

[8] M. A. Abdelraheem, G. Leander, and E. Zenner. Differential Cryptanalysis of Round-Reduced PRINTcipher: Computing Roots of Permutations. In *Fast Software Encryption, 18th International Workshop — FSE 2011*, Lecture Notes in Computer Science, pages 5–14. Springer, 2011. Pre-proceedings.

[9] R. Anderson and E. Biham. Two Practical and Provably Secure Block Ciphers: BEAR and LION. In *Fast Software Encryption, Third International Workshop — FSE '96*, volume 1039 of *Lecture Notes in Computer Science*, pages 113–120. Springer, 1996.

[10] ANSI. ANSI X9.62 The Elliptic Curve Digital Signature Algorithm (ECDSA). Available at `http://www.ansi.org/`.

[11] K. Aoki, J. Guo, K. Matusiewicz, Y. Sasaki, and L. Wang. Preimages for Step-Reduced SHA-2. In *Advances in Cryptology — ASIACRYPT 2009*,

volume 5912 of *Lecture Notes in Computer Science*, pages 578–597. Springer, 2009.

[12] R. M. Avanzi, H. Cohen, C. Doche, G. Frey, T. Lange, K. Nguyen, and F. Vercauteren. *Handbook of Elliptic and Hyperelliptic Curve Cryptography*. CRC Press, 2005.

[13] J. A. B. Jungk, S. Reith. On Optimized FPGA Implementations of the SHA-3 Candidate Groestl. Cryptology ePrint Archive, Report 2009/206, 2009. Available at `http://eprint.iacr.org/2009/206`.

[14] B. Baldwin, A. Byrne, M. Hamilton, N. Hanley, R. P. McEvoy, W. Pan, and W. P. Marnane. FPGA Implementations of SHA-3 Candidates: CubeHash, Grøstl, Lane, Shabal and Spectral Hash. Cryptology ePrint Archive, Report 2009/342, 2009. Available at `http://eprint.iacr.org/2009/342`.

[15] B. Baldwin, A. Byrne, L. Lu, M. Hamilton, N. Hanley, M. O'Neill, and W. P. Marnane. A Hardware Wrapper for the SHA-3 Hash Algorithms. Cryptology ePrint Archive, Report 2010/124, 2010. Available at `http://eprint.iacr.org/2010/124`.

[16] B. Baldwin, N. Hanley, M. Hamilton, L. Lu, A. Byrne, M. O'Neill, and W. P. Marnane. FPGA Implementations of the Round Two SHA-3 Candidates. In *20th International Conference on Field Programmable Logic and Applications — FPL 2010*, pages 400–407, 2010.

[17] G. V. Bard, N. Courtois, J. Nakahara, P. Sepehrdad, and B. Zhang. Algebraic, AIDA/Cube and Side Channel Analysis of KATAN Family of Block Ciphers. In *Progress in Cryptology — INDOCRYPT 2010*, volume 6498 of *Lecture Notes in Computer Science*, pages 176–196. Springer, 2010.

[18] P. Barrett. Implementing the Rivest Shamir and Adleman Public Key Encryption Algorithm on a Standard Digital Signal Processor. In *Advances in Cryptology — CRYPTO '86*, volume 263 of *Lecture Notes in Computer Science*, pages 311–323. Springer, 1987.

[19] M. Bernet, L. Henzen, H. Kaeslin, N. Felber, and W. Fichtner. Hardware Implementations of the SHA-3 Candidates Shabal and CubeHash. In *Midwest Symposium on Circuits and Systems*, pages 515–518. IEEE Computer Society, 2009.

[20] E. Biham, R. Anderson, and L. Knudsen. Serpent: A New Block Cipher Proposal. In *Fast Software Encryption, 5th International Workshop — FSE '98*, volume 1372 of *Lecture Notes in Computer Science*, pages 222–238. Springer, 1998.

[21] A. Bogdanov, O. Dunkelman, M. Knežević, C. Rechberger, and I. Verbauwhede. On the Key Schedules of Lightweight Block Ciphers for RFID Applications. submitted to *IEEE Communications Letters*, 2011.

[22] A. Bogdanov, L. R. Knudsen, G. Le, 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 2007*, volume 4727 of *Lecture Notes in Computer Science*, pages 450–466. Springer, 2007.

[23] A. Bogdanov and C. Rechberger. Generalizing Meet-in-the-Middle Attacks: Cryptanalysis of the Lightweight Block Cipher KTANTAN. In *Selected Areas in Cryptography — SAC 2010*, 2010. To appear.

[24] D. Boneh, G. Durfee, and N. Howgrave-Graham. Factoring $N = p^r q$ for Large r. In *Advances in Cryptology — CRYPTO '99*, volume 1666 of *Lecture Notes in Computer Science*, pages 326–337. Springer, 1999.

[25] A. Bosselaers, R. Govaerts, and J. Vandewalle. SHA: A Design for Parallel Architectures? In *Advances in Cryptology — EUROCRYPT '97*, volume 1233 of *Lecture Notes in Computer Science*, pages 348–362. Springer, 1997.

[26] A. Bosselaers and B. Preneel, editors. *RIPE, Integrity Primitives for Secure Information Systems, Final Report of RACE Integrity Primitives Evaluation (RIPE-RACE 1040).* Lecture Notes in Computer Science 1007, 1995.

[27] C. D. Cannière, O. Dunkelman, and M. Knežević. KATAN & KTANTAN — A Family of Small and Efficient Hardware-Oriented Block Ciphers. In *Cryptographic Hardware and Embedded Systems — CHES 2009*, volume 5747 of *Lecture Notes in Computer Science*, pages 272–288. Springer, 2009.

[28] C. D. Cannière, H.Sato, and D. Watanabe. Hash Function *Luffa*. In *The First SHA-3 Candidate Conference*, 2009.

[29] C. D. Cannière and B. Preneel. Trivium. In *The eSTREAM Finalists*, volume 4986 of *Lecture Notes in Computer Science*, pages 244–266. Springer, 2008.

[30] J. M. P. Cardoso and P. C. Diniz. Modeling Loop Unrolling: Approaches and Open Issues. In *Computer Systems: Architectures, Modeling, and Simulation*, volume 3133 of *Lecture Notes in Computer Science*, pages 224–233. Springer, 2004.

[31] Ç.K. Koç and T. Acar. Montgomery Multiplication in GF($2^k$). *Designs, Codes and Cryptography*, 14(1):57–69, 1998.

[32] CERG at George Mason University. Hardware Interface of a Secure Hash Algorithm (SHA). Functional Specification, October 2009. Available at `http://cryptography.gmu.edu/athena/`.

[33] F. Chabaud and A. Joux. Differential Collisions in SHA-0. In *Advances in Cryptology — CRYPTO '98*, volume 1462 of *Lecture Notes in Computer Science*. Springer, 1998.

[34] D. Coppersmith. Factoring with a Hint. IBM Research Report RC 19905, 1995.

[35] D. Coppersmith. Finding a Small Root of a Bivariate Integer Equation; Factoring with High Bits Known. In *Advances in Cryptology — EUROCRYPT '96*, volume 1070 of *Lecture Notes in Computer Science*, pages 178–189. Springer, 1996.

[36] D. Coppersmith. Small Solutions to Polynomial Equations, and Low Exponent Vulnerabilities. *Journal of Cryptology*, 10(4):233–260, 1996.

[37] N. T. Courtois, G. V. Bard, and D. Wagner. Algebraic and Slide Attacks on KeeLoq. In *Fast Software Encryption, 15th International Workshop — FSE 2008*, volume 5086 of *Lecture Notes in Computer Science*, pages 97–115. Springer, 2008.

[38] L. Dadda, M. Macchetti, and J. Owen. An ASIC Design for a High Speed Implementation of the Hash Function SHA-256 (384, 512). In *ACM Great Lakes Symposium on VLSI*, pages 421–425. ACM, 2004.

[39] J. Daemen and V. Rijmen. *The design of Rijndael: AES–The Advanced Encryption Standard*. Springer, 2002.

[40] B. den Boer and A. Bosselaers. An Attack on the Last Two Rounds of MD4. In *Advances in Cryptology — CRYPTO '91*, volume 576 of *Lecture Notes in Computer Science*, pages 194–203. Springer, 1992.

[41] B. den Boer and A. Bosselaers. Collisions for the Compression Function of MD5. In *Advances in Cryptology — CRYPTO '93*, volume 773 of *Lecture Notes in Computer Science*, pages 293–304. Springer, 1994.

[42] J.-F. Dhem. *Design of an Efficient Public-Key Cryptographic Library for RISC-based Smart Cards*. PhD Thesis, Université Catholique de Louvain, Belgium, 1998.

[43] J.-F. Dhem. Efficient Modular Reduction Algorithm in $\mathbb{F}_q[x]$ and Its Application to Left to Right Modular Multiplication in $\mathbb{F}_2[x]$. In *Cryptographic Hardware and Embedded Systems — CHES '2003*, volume 2779 of *Lecture Notes in Computer Science*, pages 203–213. Springer, 2003.

[44] W. Diffie and M. E. Hellman. New Directions in Cryptography. *IEEE Transactions on Information Theory*, 22:644–654, 1976.

[45] I. Dinur and A. Shamir. Cube Attacks on Tweakable Black Box Polynomials. In *Advances in Cryptology — EUROCRYPT 2009*, volume 5479 of *Lecture Notes in Computer Science*, pages 278–299. Springer, 2009.

[46] H. Dobbertin. The Status of MD5 After a Recent Attack. *CryptoBytes Technical Newsletter*, 2(2), 1996.

[47] H. Dobbertin. Cryptanalysis of MD4. *Journal of Cryptology*, 11(4):253–271, 1998.

[48] H. Dobbertin, A. Bosselaers, and B. Preneel. RIPEMD-160: A Strengthened Version of RIPEMD. In *Fast Software Encryption, Third International Workshop — FSE '96*, volume 1039 of *Lecture Notes in Computer Science*, pages 71–82. Springer, 1996.

[49] S. Dominikus. A Hardware Implementation of MD-4 Family Hash Algorithms. In *9th IEEE International Conference on Electronics, Circuits, and Systems – ICECS '02*, volume 3, pages 1143–1146, 2002.

[50] T. ElGamal. A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms. In *Advances in Cryptology — CRYPTO '85*, volume 218 of *Lecture Notes in Computer Science*, pages 10–18. Springer, 1986.

[51] M. Feldhofer. *Low-Power Hardware Design of Cryptographic Algorithms for RFID Tags*. PhD Thesis, Graz University of Technology, Austria, 2008.

[52] M. Feldhofer, J. Wolkerstorfer, and V. Rijmen. AES Implementation on a Grain of Sand. *IEE Proceedings Information Security*, 152(1):13–20, 2005.

[53] K. Gaj, E. Homsirikamol, and M. Rogawski. Comprehensive Comparison of Hardware Performance of Fourteen Round 2 SHA-3 Candidates with 512-bit Outputs Using Field Programmable Gate Arrays. In *The Second SHA-3 Candidate Conference*, 2010.

[54] K. Gaj, E. Homsirikamol, and M. Rogawski. Fair and Comprehensive Methodology for Comparing Hardware Performance of Fourteen Round Two SHA-3 Candidates using FPGAs. In *Cryptographic Hardware and Embedded Systems — CHES 2010*, volume 6225 of *Lecture Notes in Computer Science*, pages 264–278. Springer, 2010.

[55] F. Garnier, R. Hajlaoui, A. Yassar, and P. Srivastava. All-Polymer Field-Effect Transistor Realized by Printing Techniques. *Science*, 265(5179):1684–1686, 1994.

[56] H. Gilbert and H. Handschuh. Security Analysis of SHA-256 and Sisters. In *Selected Areas in Cryptography — SAC 2004*, volume 3357 of *Lecture Notes in Computer Science*. Springer, 2004.

[57] T. Good and M. Benaissa. Hardware Results for Selected Stream Cipher Candidates. In *ECRYPT Workshop, The State of the Art of Stream Ciphers — SASC 2007*, pages 191–204, 2007.

[58] J. Guajardo, T. Güneysu, S. Kumar, C. Paar, and J. Pelzl. Efficient Hardware Implementation of Finite Fields with Applications to Cryptography. *Acta Applicandae Mathematicae*, 93:75–118, 2006.

[59] T. Güneysu and C. Paar. Ultra High Performance ECC over NIST Primes on Commercial FPGAs. In *Cryptographic Hardware and Embedded Systems — CHES 2008*, volume 5154 of *Lecture Notes in Computer Science*, pages 62–78. Springer, 2008.

[60] X. Guo, S. Huang, L. Nazhandali, and P. Schaumont. Fair and Comprehensive Performance Evaluation of 14 Second Round SHA-3 ASIC Implementations. In *The Second SHA-3 Candidate Conference*, 2010.

[61] S. Halevi, W. E. Hall, and C. S. Jutla. The Hash Function Fugue. Submission Document, 2008. Available at `http://domino.research.ibm.com/comm/research_projects.nsf/pages/fugue.index.html`.

[62] P. Hämäläinen, T. Alho, M. Hännikäinen, and T. D.Hämäläinen. Design and Implementation of Low-Area and Low-Power AES Encryption Hardware Core. In *Euromicro Conference on Digital System Design*, pages 577–583. IEEE Computer Society, 2006.

[63] H. Handschuh and D. Naccache. SHACAL. Preproceedings of NESSIE first workshop, Leuven, 2000.

[64] L. Hars. Long Modular Multiplication for Cryptographic Applications. In *Cryptographic Hardware and Embedded Systems — CHES 2004*, volume 3156 of *Lecture Notes in Computer Science*, pages 218–254. Springer, 2004.

[65] P. Hawkes, M. Paddon, and G. Rose. On Corrective Patterns for the SHA-2 Family. Cryptology ePrint Archive, Report 2004/207, 2004. Available at `http://eprint.iacr.org/2004/207`.

[66] D. Hein, J. Wolkerstorfer, and N. Felber. ECC Is Ready for RFID – A Proof in Silicon. In *Selected Areas in Cryptography — SAC 2009*, volume 5867 of *Lecture Notes in Computer Science*, pages 401–413. Springer, 2009.

[67] M. Hell, T. Johansson, A. Maximov, and W. Meier. The Grain Family of Stream Ciphers. In *The eSTREAM Finalists*, volume 4986 of *Lecture Notes in Computer Science*, pages 179–190. Springer, 2008.

[68] L. Henzen, J.-P. Aumasson, W. Meier, and R. C.-W. Phan. VLSI Characterization of the Cryptographic Hash Function BLAKE, 2010. Available at `http://131002.net/data/papers/HAMP10.pdf`.

[69] 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 2010*, volume 6225 of *Lecture Notes in Computer Science*, pages 248–263. Springer, 2010.

[70] A. Hodjat and I. Verbauwhede. Minimum Area Cost for a 30 to 70 Gbits/s AES Processor. In *IEEE Computer Society Annual Symposium on VLSI (ISVLSI 2004)*, pages 498–502. IEEE, 2004.

[71] D. Hong, J. Sung, S. Hong, J. Lim, S. Lee, B. Koo, C. Lee, D. Chang, J. Lee, K. Jeong, H. Kim, J. Kim, and S. Chee. HIGHT: A New Block Cipher Suitable for Low-Resource Device. In *Cryptographic Hardware and Embedded Systems — CHES 2006*, volume 4249 of *Lecture Notes in Computer Science*, pages 46–59. Springer, 2006.

[72] K. Ideguchi, T. Owada, and H. Yoshida. A Study on RAM Requirements of Various SHA-3 Candidates on Low-Cost 8-bit CPUs. Cryptology ePrint Archive, Report 2009/260, 2009. Available at `http://eprint.iacr.org/2009/260`.

[73] S. Indesteege, N. Keller, O. Dunkelman, E. Biham, and B. Preneel. A Practical Attack on KeeLoq. In *Advances in Cryptology — EUROCRYPT 2008*, volume 4965 of *Lecture Notes in Computer Science*, pages 1–18. Springer, 2008.

[74] S. Indesteege, F. Mendel, B. Preneel, and C. Rechberger. Collisions and Other Non-random Properties for Step-Reduced SHA-256. In *Selected Areas in Cryptography — SAC 2008*, volume 5381 of *Lecture Notes in Computer Science*, pages 276–293. Springer, 2009.

[75] Intel. Moore's Law. Available at `http://www.intel.com/technology/mooreslaw/`.

[76] International Organization for Standardization. ISO/IEC 10118-3, "Information technology - Security techniques - Hash functions - Part 3: Dedicated hash functions". 2003.

[77] International Technology Roadmap for Semiconductors. 2010 Update, Overview, 2010. Available at `http://www.itrs.net/Links/2010ITRS/2010Update/ToPost/2010_Update_Overview.pdf`.

[78] T. Isobe. A Single-Key Attack on the Full GOST Block Cipher. In *Fast Software Encryption, 18th International Workshop — FSE 2011*, Lecture Notes in Computer Science, pages 273–286. Springer, 2011. Pre-proceedings.

[79] M. Izadi, B. Sadeghiyan, S. Sadeghian, and H. Khanooki. MIBS: A New Lightweight Block Cipher. In J. Garay, A. Miyaji, and A. Otsuka, editors, *Cryptology and Network Security*, volume 5888 of *Lecture Notes in Computer Science*, pages 334–348. Springer Berlin / Heidelberg, 2009.

[80] A. Joux, P. Carribault, W. Jalby, and C. Lemuet. Collisions in SHA-0. In *Rump session of CRYPTO 2004*, 2004.

[81] M. Joye. RSA Moduli with a Predetermined Portion: Techniques and Applications. In *Information Security Practice and Experience — ISPEC 2008*, volume 4991 of *Lecture Notes in Computer Science*, pages 116–130. Springer, 2008.

[82] A. Juels and S. A. Weis. Authenticating Pervasive Devices with Human Protocols. In *Advances in Cryptology — CRYPTO 2005*, volume 3621 of *Lecture Notes in Computer Science*, pages 293–308. Springer, 2005.

[83] M. E. Kaihara and N. Takagi. Bipartite Modular Multiplication. In *Cryptographic Hardware and Embedded Systems — CHES 2005*, volume 3659 of *Lecture Notes in Computer Science*, pages 201–210. Springer, 2005.

[84] A. Karatsuba and Y. Ofman. Multiplication of Many-Digital Numbers by Automatic Computers. *Soviet Physics - Doklady*, 7:595–596, 1963. Translation from Doklady Akademii Nauk SSSR, 145:2, 293–294, 1962.

[85] J. Kelsey, B. Schneier, and D. Wagner. Related-Key Cryptanalysis of 3-WAY, Biham-DES,CAST, DES-X, NewDES, RC2, and TEA. In *International Conference on Information and Communications Security*, pages 233–246. Springer, 1997.

[86] E. Khan, M. W. El-Kharashi, F. Gebali, and M. Abd-El-Barr. Design and Performance Analysis of a Unified, Reconfigurable HMAC-Hash Unit. *IEEE Transaction on Circuits and Systems*, 54(12):2683–2695, 2007.

[87] J. S. Kilby. Miniature Semiconductor Integrated Circuit, December 1963. U.S. Patent 3,115,581.

[88] M. Kim and J. Ryou. Power Efficient Hardware Architecture of SHA-1 Algorithm for Trusted Mobile Computing. In *Proceedings of the 9th international conference on Information and communications security*, ICICS'07, pages 375–385. Springer, 2007.

[89] M. Kim, J. Ryou, and S. Jun. Efficient Hardware Architecture of SHA-256 Algorithm for Trusted Mobile Computing. In *Information Security and Cryptology*, volume 5487 of *Lecture Notes in Computer Science*, pages 240–252. Springer, 2009.

[90] S. Knellwolf, W. Meier, and M. Naya-Plasencia. Conditional Differential Cryptanalysis of NLFSR-Based Cryptosystems. In *Advances in Cryptology — ASIACRYPT 2010*, volume 6477 of *Lecture Notes in Computer Science*, pages 130–145. Springer, 2010.

[91] M. Knežević, L. Batina, and I. Verbauwhede. Modular Reduction Without Precomputational Phase. In *IEEE International Symposium on Circuits and Systems — ISCAS 2009*, pages 1389–1392. IEEE, 2009.

[92] M. Knežević, K. Kobayashi, J. Ikegami, S. Matsuo, A. Satoh, Ü. Kocabaş, J. Fan, T. Katashita, T. Sugawara, K. Sakiyama, I. Verbauwhede, K. Ohta, N. Homma, and T. Aoki. Fair and Consistent Hardware Evaluation of Fourteen Round Two SHA-3 Candidates. *IEEE Transactions on VLSI*. 13 pages, 2011. To appear.

[93] M. Knežević, K. Sakiyama, J. Fan, and I. Verbauwhede. Modular Reduction in GF($2^n$) Without Pre-Computational Phase. In *Arithmetic of Finite Fields, Second International Workshop — WAIFI 2008*, volume 5130 of *Lecture Notes in Computer Science*, pages 77–87. Springer, 2008.

[94] M. Knežević, K. Sakiyama, Y. K. Lee, and I. Verbauwhede. On the High-Throughput Implementation of RIPEMD-160 Hash Algorithm. In *IEEE International Conference on Application-Specific Systems, Architecture and Processors — ASAP 2008*, pages 85–90. IEEE Computer Society, 2008.

[95] M. Knežević and I. Verbauwhede. Hardware Evaluation of the Luffa Hash Family. In *Workshop on Embedded Systems Security — WESS 2009*, page 6, 2009.

[96] M. Knežević, F. Vercauteren, and I. Verbauwhede. Faster Interleaved Modular Multiplication Based on Barrett and Montgomery Reduction Methods. *IEEE Transactions on Computers*, 59(12):1715–1721, 2010.

[97] M. Knežević, F. Vercauteren, and I. Verbauwhede. Speeding Up Bipartite Modular Multiplication. In *Arithmetic of Finite Fields, Third International Workshop — WAIFI 2010*, volume 6087 of *Lecture Notes in Computer Science*, pages 166–179. Springer, 2010.

[98] L. Knudsen, G. Leander, A. Poschmann, and M. Robshaw. PRINTcipher: A Block Cipher for IC-Printing. In *Cryptographic Hardware and Embedded Systems — CHES 2010*, volume 6225 of *Lecture Notes in Computer Science*, pages 16–32. Springer, 2010.

[99] N. Koblitz. Elliptic Curve Cryptosystem. *Math. Comp.*, 48:203–209, 1987.

[100] G. Leander, C. Paar, A. Poschmann, and K. Schramm. New Lightweight DES Variants. In *Fast Software Encryption, 14th International Workshop — FSE 2007*, volume 4593 of *Lecture Notes in Computer Science*, pages 196–210. Springer, 2007.

[101] Y. Lee, H. Chan, and I. Verbauwhede. Throughput Optimized SHA-1 Architecture Using Unfolding Transformation. In *IEEE International*

*Conference on Application-Specific Systems, Architecture and Processors — ASAP 2006*, pages 354–359. IEEE Computer Society, 2006.

[102] Y. 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, 8th International Workshop — WISA 2007*, volume 4867 of *Lecture Notes in Computer Science*, pages 102–114. Springer, 2007.

[103] Y. K. Lee, L. Batina, K. Sakiyama, and I. Verbauwhede. Elliptic Curve Based Security Processor for RFID. *IEEE Transactions on Computers*, 57(11):1514–1527, 2008.

[104] Y. K. Lee, H. Chan, and I. Verbauwhede. Design Methodology for Throughput Optimum Architectures of Hash Algorithms of the MD4-class. *Journal of Signal Processing Systems*, 53:89–102, November 2008.

[105] Y. K. Lee, M. Knežević, and I. Verbauwhede. Hardware Design for Hash functions. In *Secure Integrated Circuits and Systems*, Integrated Circuits and Systems, pages 79–104. Springer, 2010.

[106] C. Leiserson, F. Rose, and J. Saxe. Optimizing Synchronous Circuitry by Retiming. In *Third Caltech Conference on VLSI*, pages 87–116, 1983.

[107] A. Lenstra. Generating RSA Moduli with a Predetermined Portion. In *Advances in Cryptology — ASIACRYPT '98*, volume 1514 of *Lecture Notes in Computer Science*, pages 1–10. Springer, 1998.

[108] R. Lidl and H. Niederreiter. *Introduction to Finite Fields and Their Applications.* Cambridge University Press, New York, NY, USA, 1986.

[109] C. Lim and T. Korkishko. mCrypton – A Lightweight Block Cipher for Security of Low-Cost RFID Tags and Sensors. In *Information Security Applications, 6th International Workshop — WISA 2005*, volume 3786 of *Lecture Notes in Computer Science*, pages 243–258. Springer, 2006.

[110] L. Lu, M. O'Neil, and E. Swartzlander. Hardware Evaluation of SHA-3 Hash Function Candidate ECHO. Presentation at the Claude Shannon Institute Workshop on Coding and Cryptography 2009, 2009. Available at `http://www.ucc.ie/en/crypto/CodingandCryptographyWorkshop/TheClaudeShannonInstituteWorkshoponCodingCryptography2009/DocumentFile-75649-en.pdf`.

[111] F. Macé, F.-X. Standaert, and J.-J. Quisquater. ASIC Implementations of the Block Cipher SEA for Constrained Applications. In *RFID Security Workshop*, 2007.

[112] A. May. *New RSA Vulnerabilities Using Lattice Reduction Methods*. PhD Thesis, University of Paderborn, Germany, 2003.

[113] A. May. Using LLL-Reduction for Solving RSA and Factorization Problems: A Survey, 2007. Avaliable at: `http://www.informatik.tu-darmstadt.de/KP/publications/07/lll.pdf`.

[114] F. Mendel, N. Pramstaller, C. Rechberger, and V. Rijmen. Analysis of Step-Reduced SHA-256. In *Fast Software Encryption, 13th International Workshop — FSE 2006*, volume 4047 of *Lecture Notes in Computer Science*, pages 126–143. Springer, 2006.

[115] F. Mendel, N. Pramstaller, C. Rechberger, and V. Rijmen. On the Collision Resistance of RIPEMD-160. In *Information Security 9th International Conference — ISC 2006*, volume 4176 of *Lecture Notes in Computer Science*, pages 101–116. Springer, 2006.

[116] A. Menezes, P. van Oorschot, and S. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1997.

[117] N. Mentens, J. Genoe, B. Preneel, and I. Verbauwhede. A Low-cost Implementation of Trivium. In *ECRYPT Workshop, The State of the Art of Stream Ciphers — SASC 2008*, pages 197–204, 2008.

[118] Microchip Technology Inc. KeeLoq® Authentication Products. Available at `http://www.microchip.com/keeloq/`.

[119] S. Mikami, N. Mizushima, S. Nakamura, and D. Watanabe. A Compact Hardware Implementation of SHA-3 Candidate Luffa, 2010. Available at `http://www.sdl.hitachi.co.jp/crypto/luffa/ACompactHardwareImplementationOfSHA-3CandidateLuffa_20101105.pdf`.

[120] V. Miller. Uses of Elliptic Curves in Cryptography. In *Advances in Cryptology: Proceedings of CRYPTO'85*, volume 218 of *Lecture Notes in Computer Science*, pages 417–426. Springer, 1985.

[121] P. Montgomery. Modular Multiplication without Trial Division. *Mathematics of Computation*, 44(170):519–521, 1985.

[122] G. E. Moore. Cramming More Components onto Integrated Circuits. *Electronics*, 38(8), 1965.

[123] G. E. Moore. No Exponential is Forever: but "Forever" can be Delayed! In *IEEE International Solid-State Circuits Conference — ISSCC 2003*, volume 1, pages 20–23, 2003.

[124] A. Namin and M. Hasan. Implementation of the Compression Function for Selected SHA-3 Candidates on FPGA. In *IEEE International Symposium on Parallel Distributed Processing*, pages 1–4, 2010.

[125] National Institute of Advanced Industrial Science and Technology (AIST), Research Center for Information Security (RCIS). Side-channel Attack Standard Evaluation Board (SASEBO). Available at `http://www.rcis.aist.go.jp/special/SASEBO/`.

[126] National Institute of Standards and Technology (NIST). Announcing Development of a Federal Information Processing Standard for Advanced Encryption Standard. Available at `http://csrc.nist.gov/archive/aes/pre-round1/aes_9701.txt`.

[127] National Institute of Standards and Technology (NIST). Cryptographic Hash Algorithm Competition. Available at `http://csrc.nist.gov/groups/ST/hash/sha-3/index.html`.

[128] National Institute of Standards and Technology (NIST). ANSI C Cryptographic API Profile for SHA-3 Candidate Algorithm Submissions, 2008. Available at `http://csrc.nist.gov/groups/ST/hash/documents/SHA3-C-API.pdf`.

[129] National Institute of Standards and Technology (NIST). FIPS 186-3: Digital Signature Standard, 2009. Available at `http://csrc.nist.gov/`.

[130] National Soviet Bureau of Standards. Informtation Processing System - Cryptographic Protection. Cryptographic Algorithm GOST 28147-89, 1989.

[131] C. Ng, T. Ng, and K. Yip. A Unified Architecture of MD5 and RIPEMD-160 Hash Algorithms. In *IEEE International Symposium on Circuits and Systems — ISCAS 2007*, pages 889–892. IEEE, 2007.

[132] I. Nikolić and A. Biryukov. Collisions for Step-Reduced SHA-256. In *Fast Software Encryption, 15th International Workshop — FSE 2008*, volume 5086 of *Lecture Notes in Computer Science*, pages 1–15. Springer, 2008.

[133] K. K. Parhi. *VLSI Digital Signal Processing Systems – Design and Implementation*. Wiley, NY, 1999.

[134] A. Poschmann. *Lightweight Cryptography – Cryptographic Engineering for a Pervasive World*. PhD Thesis, Ruhr-University Bochum, Germany, 2009.

[135] A. Poschmann, S. Ling, and H. Wang. 256 Bit Standardized Crypto for 650 GE - GOST Revisited. In *Cryptographic Hardware and Embedded Systems — CHES 2010*, volume 6225 of *Lecture Notes in Computer Science*, pages 219–233. Springer, 2010.

[136] M. J. Potgieter and B. J. van Dyk. Two Hardware Implementations of the Group Operations Necessary for Implementing an Elliptic Curve Cryptosystem over a Characteristic Two Finite Field. In *IEEE AFRICON 2002*, pages 187–192, 2002.

[137] B. Preneel. *Analysis and Design of Cryptographic Hash Functions*. PhD thesis, Katholieke Universiteit Leuven, Belgium, 1993.

[138] B. Preneel, R. Govaerts, and J. Vandewalle. Hash Functions Based on Block Ciphers: A Synthetic Approach. In *Advances in Cryptology — CRYPTO '93*, volume 773 of *Lecture Notes in Computer Science*, pages 368–378. Springer, 1994.

[139] J.-J. Quisquater. Encoding System According to the So-Called RSA Method, by Means of a Microcontroller and Arrangement Implementing this System, 1992. US Patent 5,166,978.

[140] M. O. Rabin. Digitalized Signatures. In *Foundations of Secure Computations*, pages 155–166. Academic Press, 1978.

[141] R. Rivest. The MD4 Message Digest Algorithm. In *Advances in Cryptology — CRYPTO '90*, volume 537 of *Lecture Notes in Computer Science*, pages 303–311. Springer, 1991.

[142] R. L. Rivest. The MD5 Message-Digest Algorithm. Internet Engineering Task Force (IETF) Request for Comments (RFC) 1321, 1992.

[143] R. L. Rivest and A. Shamir. Efficient Factoring Based on Partial Information. In *Advances in Cryptology — EUROCRYPT '85*, volume 219 of *Lecture Notes in Computer Science*, pages 31–34. Springer, 1986.

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

[145] 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 — CARDIS 2008*, volume 5189 of *Lecture Notes in Computer Science*, pages 89–103. Springer, 2008.

[146] K. Sakiyama. *Secure Design Methodology and Implementation for Embedded Public-key Cryptosystems*. PhD thesis, Katholieke Universiteit Leuven, 2007.

[147] K. Sakiyama, M. Knežević, J. Fan, B. Preneel, and I. Verbauwhede. Tripartite Modular Multiplication. *Integration, the VLSI journal*. 14 pages, 2011. To appear.

[148] S. K. Sanadhya and P. Sarkar. 22-Step Collisions for SHA-2. arXiv e-print archive, arXiv:0803.1220v1, Mar. 2008. Available at `http://www.arxiv.org/`.

[149] S. K. Sanadhya and P. Sarkar. Attacking Reduced Round SHA-256. In *Applied Cryptography and Network Security — ACNS 2008*, volume 5037 of *Lecture Notes in Computer Science*, pages 130–143, 2008.

[150] S. K. Sanadhya and P. Sarkar. New Collision Attacks against Up to 24-Step SHA-2. In *Progress in Cryptology — INDOCRYPT 2008*, volume 5365 of *Lecture Notes in Computer Science*, pages 91–103. Springer, 2008.

[151] S. K. Sanadhya and P. Sarkar. Non-linear Reduced Round Attacks against SHA-2 Hash Family. In *Information Security and Privacy — ACISP 2008*, volume 5107 of *Lecture Notes in Computer Science*, pages 254–266. Springer, 2008.

[152] A. Satoh and T. Inoue. ASIC-Hardware-Focused Comparison for Hash Functions MD5, RIPEMD-160, and SHS. In *International Conference on Information Technology: Coding and Computing — ITCC*, pages 532–537, 2005.

[153] P. Schaumont, K. Sakiyama, A. Hodjat, and I. Verbauwhede. Embedded Software Integration for Coarse-Grain Reconfigurable Systems. *Parallel and Distributed Processing Symposium, International*, 4:137–142, 2004.

[154] P. Schaumont and I. Verbauwhede. Domain-specific Tools and Methods for Application in Security Processor Design. *Kluwer Journal for Design Automation of Embedded Systems*, 7(4):365–383, 2002.

[155] C.-P. Schnorr. Efficient Identification and Signatures for Smart Cards. In *Advances in Cryptology — CRYPTO '89*, volume 435 of *Lecture Notes in Computer Science*, pages 239–252. Springer, 1990.

[156] N. Sklavos and O. Koufopavlou. On the Hardware Implementation of RIPEMD Processor: Networking High Speed Hashing, up to 2 Gbps. *Computers and Electrical Engineering*, 31(6):361–379, 2005.

[157] F.-X. Standaert, G. Piret, N. Gershenfeld, and J.-J. Quisquater. SEA: A Scalable Encryption Algorithm for Small Embedded Applications. In *Smart Card Research and Advanced Applications — CARDIS 2006*, volume 3928 of *Lecture Notes in Computer Science*, pages 222–236. Springer, 2006.

[158] Standards for Efficient Cryptography. SEC2: Recommended Elliptic Curve Domain Parameters. `http://www.secg.org/`.

[159] M. Stevens, A. Sotirov, J. Appelbaum, A. Lenstra, D. Molnar, D. A. Osvik, and B. Weger. Short Chosen-Prefix Collisions for MD5 and the Creation of a Rogue CA Certificate. In *Advances in Cryptology — CRYPTO 2009*, volume 5677 of *Lecture Notes in Computer Science*, pages 55–69. Springer, 2009.

[160] D. Suzuki. How to Maximize the Potential of FPGA Resources for Modular Exponentiation. In *Cryptographic Hardware and Embedded Systems — CHES 2007*, volume 4727 of *Lecture Notes in Computer Science*, pages 272–288. Springer, 2007.

[161] T. Takagi. Fast RSA-Type Cryptosystem Modulo $p^k q$. In *Advances in Cryptology — CRYPTO '98*, volume 1462 of *Lecture Notes in Computer Science*, pages 318–326. Springer, 1998.

[162] The SHA-3 Zoo. SHA-3 Hardware Implementations. Available at `http://ehash.iaik.tugraz.at/wiki/SHA-3_Hardware_Implementations`.

[163] 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. Available at `http://eprint.iacr.org/2009/349`.

[164] S. Tillich, M. Feldhofer, M. Kirschbaum, T. Plos, J. 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. Available at `http://eprint.iacr.org/2009/510`.

[165] S. Tillich, M. Feldhofer, M. Kirschbaum, T. Plos, J. Schmidt, and A. Szekely. Uniform Evaluation of Hardware Implementations of the Round-Two SHA-3 Candidates. In *The Second SHA-3 Candidate Conference*, 2010.

[166] A. Toom. The Complexity of a Scheme of Functional Elements Realizing the Multiplication of Integers. *Soviet Mathematics - Doklady*, 3:714–716, 1963. Translation from Doklady Akademii Nauk SSSR, 150:3, 496–498, 1963.

[167] P. C. van Oorschot and M. J. Wiener. Parallel Collision Search with Cryptanalytic Applications. *Journal of Cryptology*, 12(1):1–28, 1999.

[168] H. C. A. van Tilborg, editor. *Encyclopedia of Cryptography and Security*. Springer, 2005.

[169] S. Vaudenay. On the Need for Multipermutations: Cryptanalysis of MD4 and SAFER. In *Fast Software Encryption, Second International Workshop — FSE '94*, volume 1008 of *Lecture Notes in Computer Science*, pages 286–297. Springer, 1995.

[170] X. Wang, X. Lai, D. Feng, H. Chen, and X. Yu. Cryptanalysis of the Hash Functions MD4 and RIPEMD. In *Advances in Cryptology — EUROCRYPT 2005*, volume 3494 of *Lecture Notes in Computer Science*, pages 1–18. Springer, 2005.

[171] X. Wang, Y. L. Yin, and H. Yu. Finding Collisions in the Full SHA-1. In *Advances in Cryptology — CRYPTO 2005*, volume 3621 of *Lecture Notes in Computer Science*, pages 17–35. Springer, 2005.

[172] X. Wang and H. Yu. How to Break MD5 and Other Hash Functions. In *Advances in Cryptology — EUROCRYPT 2005*, volume 3494 of *Lecture Notes in Computer Science*, pages 19–35. Springer, 2005.

[173] X. Wang, H. Yu, and Y. L. Yin. Efficient Collision Search Attacks on SHA-0. In *Advances in Cryptology — CRYPTO 2005*, volume 3621 of *Lecture Notes in Computer Science*, pages 1–16. Springer, 2005.

[174] D. Wheeler and R. Needham. TEA Extensions. Available at `http://www.cix.co.uk/~klockstone/xtea.pdf`.

[175] D. Wheeler and R. Needham. TEA, a Tiny Encryption Algorithm. In *Fast Software Encryption, Second International Workshop — FSE '94*, volume 1008 of *Lecture Notes in Computer Science*, pages 363–366. Springer, 1995.

[176] H. Wu. Montgomery Multiplier and Squarer for a Class of Finite Fields. *IEEE Transactions on Computers*, 51(5):521–529, May 2002.

[177] Xilinx. Virtex 2.5V FPGA Complete Data Sheet. Available at `http://www.xilinx.com/support/documentation/data_sheets/ds003.pdf`.

[178] Xilinx. Virtex-II Platform FPGA User Guide. Available at `http://www.xilinx.com/support/documentation/data_sheets/ds031.pdf`.

[179] Y. Yu, Y. Yang, Y. Fan, and H. Min. Security Scheme for RFID Tag. Technical Report, Auto-ID Labs white paper WP-HARDWARE-022.

[180] P. S. Z. Chen, S. Morozov. A Hardware Interface for Hashing Algorithms. Cryptology ePrint Archive, Report 2008/529, 2008. Available at `http://eprint.iacr.org/2008/529`.

[181] X. Zhao, Z. Wang, H. Lu, and K. Dai. A 6.35 Mbps 1024-bit RSA Crypto Coprocessor in a 0.18um CMOS Technology. In *International Conference on Very Large Scale Integration*, pages 216–221, 2006.

[182] Y. Zheng, J. Pieprzyk, and J. Seberry. HAVAL – A One-way Hashing Algorithm with Variable Length of Output. In *Advances in Cryptology — ASIACRYPT '90*, volume 453 of *Lecture Notes in Computer Science*, pages 83–104. Springer, 1990.

# List of Publications

## Book Chapters

[1] M. Knežević, L. Batina, E. D. Mulder, J. Fan, B. Gierlichs, Y. K. Lee, R. Maes, and I. Verbauwhede, "Signal Processing for Cryptography and Security Applications," in *Handbook of Signal Processing Systems*, pp. 161–177, Springer, 2010.

[2] Y. K. Lee, M. Knežević, and I. Verbauwhede, "Hardware Design for Hash Functions," in *Secure Integrated Circuits and Systems*, Integrated Circuits and Systems, pp. 79–104, Springer, 2010.

## International Journals

[1] M. Knežević, F. Vercauteren, and I. Verbauwhede, "Faster Interleaved Modular Multiplication Based on Barrett and Montgomery Reduction Methods," *IEEE Transactions on Computers*, vol. 59, no. 12, pp. 1715–1721, 2010.

[2] M. Knežević, K. Kobayashi, J. Ikegami, S. Matsuo, A. Satoh, Ü. Kocabaş, J. Fan, T. Katashita, T. Sugawara, K. Sakiyama, I. Verbauwhede, K. Ohta, N. Homma, and T. Aoki, "Fair and Consistent Hardware Evaluation of Fourteen Round Two SHA-3 Candidates," *IEEE Transactions on VLSI*. 13 pages, 2011. To appear.

[3] K. Sakiyama, M. Knežević, J. Fan, B. Preneel, and I. Verbauwhede, "Tripartite Modular Multiplication," *Integration, the VLSI journal*. 14 pages, 2011. To appear.

# International Conferences and Workshops

[1] M. Knežević, K. Sakiyama, J. Fan, and I. Verbauwhede, "Modular Reduction in GF($2^n$) Without Pre-Computational Phase," in *Arithmetic of Finite Fields, Second International Workshop — WAIFI 2008*, vol. 5130 of *Lecture Notes in Computer Science*, pp. 77–87, Springer, 2008.

[2] M. Knežević, L. Batina, and I. Verbauwhede, "Modular Reduction Without Precomputational Phase," in *IEEE International Symposium on Circuits and Systems — ISCAS 2009*, pp. 1389–1392, IEEE, 2009.

[3] M. Knežević, F. Vercauteren, and I. Verbauwhede, "Speeding Up Bipartite Modular Multiplication," in *Arithmetic of Finite Fields, Third International Workshop — WAIFI 2010*, vol. 6087 of *Lecture Notes in Computer Science*, pp. 166–179, Springer, 2010.

[4] C. D. Cannière, O. Dunkelman, and M. Knežević, "KATAN & KTANTAN — A Family of Small and Efficient Hardware-Oriented Block Ciphers," in *Cryptographic Hardware and Embedded Systems — CHES 2009*, vol. 5747 of *Lecture Notes in Computer Science*, pp. 272–288, Springer, 2009.

[5] M. Knežević, K. Sakiyama, Y. K. Lee, and I. Verbauwhede, "On the High-Throughput Implementation of RIPEMD-160 Hash Algorithm," in *IEEE International Conference on Application-Specific Systems, Architecture and Processors — ASAP 2008*, pp. 85–90, IEEE Computer Society, 2008.

[6] K. Kobayashi, J. Ikegami, M. Knežević, X. Guo, S. Matsuo, S. Huang, L. Nazhandali, Ü. Kocabaş, J. Fan, A. Satoh, I. Verbauwhede, K. Sakiyama, and K. Ohta, "Prototyping Platform for Performance Evaluation of SHA-3 Candidates," in *IEEE International Workshop on Hardware-Oriented Security and Trust — HOST 2010*, pp. 60–63, IEEE, 2010.

[7] M. Knežević and I. Verbauwhede, "Hardware Evaluation of the Luffa Hash Family," in *Workshop on Embedded Systems Security — WESS 2009*, p. 6, 2009.

[8] D. Karaklajić, M. Knežević, and I. Verbauwhede, "Low Cost Built-In Self Test for Public Key Crypto Cores," in *Fifth International Workshop on Fault Diagnosis and Tolerance in Cryptography — FDTC 2010*, pp. 97–103, IEEE, 2010.

[9] A. Das, M. Knežević, S. Seys, and I. Verbauwhede, "Challenge-Response Based Secure Test Wrapper for Testing Cryptographic Circuits," in *16th IEEE European Test Symposium — ETS 2011*, IEEE. 6 pages, 2011. To appear.

[10] J. Fan, M. Knežević, D. Karaklajić, R. Maes, V. Rožić, L. Batina, and I. Verbauwhede, "FPGA-based Testing Strategy for Cryptographic Chips: A

Case Study on Elliptic Curve Processor for RFID Tags," in *IEEE International On-Line Testing Symposium — IOLTS 2009*, pp. 189–191, IEEE, 2009.

[11] M. Knežević, V. Velichkov, B. Preneel, and I. Verbauwhede, "On the Practical Performance of Rateless Codes," in *International Conference on Wireless Information Networks and Systems — WINSYS 2008*, p. 4, 2008.

[12] M. Knežević and V. Velichkov, "Demonstration of Unobservable Voice Over IP," in *International Workshop on Adaptive and Dependable Mobile Ubiquitous Systems — ADAMUS 2008*, p. 3, 2008.

## Local Conferences and Journals

[1] M. Knežević, V. Rožić, and I. Verbauwhede, "Design Methods for Embedded Security," in *Telecommunications Forum (TELFOR 2008)*, p. 4, 2008.

[2] M. Knežević, V. Rožić, and I. Verbauwhede, "Design Methods for Embedded Security," *Telfor Journal*, vol. 1, no. 2, pp. 69–72, 2009.

[3] D. Watanabe, H. Sato, C. D. Cannière, and M. Knežević, "A New Hash Funciton Family Luffa," in *Symposium on Cryptography and Information Security*, p. 6, 2009.

[4] Y. K. Lee, L. Batina, J. Fan, D. Karaklajić, M. Knežević, Ü. Kocabaş, V. Rožić, and I. Verbauwhede, "Tiny Public-Key Security Processor," ISSCC09 Student Forum, 2009.

## Technical Reports and Deliverables

[1] D. Karaklajić, M. Knežević, and I. Verbauwhede, "Multiplier Based Built In Self Test for Cryptographic Applications," submitted to *IEEE Transactions on Computers*, 2010.

[2] A. Bogdanov, O. Dunkelman, M. Knežević, C. Rechberger, and I. Verbauwhede, "On the Key Schedules of Lightweight Block Ciphers for RFID Applications," submitted to *IEEE Communications Letters*, 2011.

[3] M. Knežević, "Hardware Evaluation of Lane," COSIC internal report, 2009.

[4] G. Danezis, C. Diaz, M. Knežević, M. Kohlweiss, S. Nikova, S. Schiffner, C. Troncoso, and V. Velichkov, "Algorithmic Solutions for Privacy, Anonymity and Identity," aeolus deliverable, 2007.

# Curriculum Vitae

Miroslav Knežević was born in Sarajevo, Bosnia and Herzegovina on September 27, 1982. He received the MSc. degree in Electrical Engineering from the University of Belgrade, Serbia in July 2006. In December 2006, he joined the COSIC research group at the Katholieke Universiteit Leuven, Belgium as a predoctoral student. He was working on the IBBT QoE project, implementing a software for "Anonymous Voice over IP". In October 2007, he started his PhD program with the emphasis on "Efficient Hardware Implementations of Cryptographic Primitives".

**Arenberg Doctoral School of Science, Engineering & Technology**
Faculty of Engineering
Department of Electrical Engineering (ESAT)
COmputer Security and Industrial Cryptography (COSIC)
Kasteelpark Arenberg 10 — 2446, 3001 Heverlee, Belgium