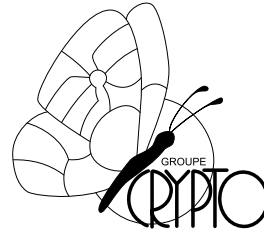




UCL
Université
catholique
de Louvain

UNIVERSITÉ CATHOLIQUE DE LOUVAIN
ÉCOLE POLYTECHNIQUE DE LOUVAIN
LABORATOIRE DE MICROÉLECTRONIQUE
UCL CRYPTO GROUP



Mathematical and Physical Concerns regarding Cryptographic Key Length

Philippe Bulens

*Thèse présentée en vue de l'obtention du grade de
Docteur en Sciences de l'Ingénieur*

Composition du jury:

Pr. Jean-Jacques Quisquater (UCL - DICE) – Promoteur
Dr. Marc Joye (Thomson R&D)
Pr. Jean-Pierre Tignol (UCL - MATH)
Pr. Louis Goubin (Université de Versailles)
Pr. François-Xavier Standaert (UCL - DICE)
Pr. Luc Vandendorpe (UCL - TELE) – Président

Louvain-la-Neuve, Belgique
Août 2009

To Elodie,

Adrien,

Victor,

Alice,

Cyriac,

Alexis,

and their generations,

For I wish them

The courage to absorb our knowledge,

And the will to develop it even further ...

Acknowledgements

THE end of a Ph. D. thesis is such a relief. However, like an ICEBERG, this little book is only the visible part of what happened during this period. I would like to thank people without whom this document would never have been achieved.

I am deeply grateful to Jean-Jacques Quisquater, my advisor, who gave me the opportunity to carry out this work at the UCL Crypto Group. Looking back at our e-mail discussions, it turns out that most of them enclose encouragements. Thank you for being such a support and distilling so much motivation.

I also wish to thank my committee and jury members for their advices, guidelines and the time they spent for me, as well as the administrative people who worked behind the scene.

Members of the UCL Crypto Group deserve my strongest gratitude for the fruitful discussions, the developed ideas and the nice atmosphere they create. Some amongst them are worth a special thank for being co-workers, co-authors and even friends. François-Xavier has earned my eternal esteem for being the lighthouse while I was cast away. Sylvie, you are a blessing for the group, for trying so hard to teach us English and how to grow up.

Thank you, my friends and family, for being so patient and seemingly so attentive while I was trying to explain you what I was doing. You were of incredible help too when it came to taking my mind off things and entertaining me as well as keeping me in touch with reality.

All my love to Mandine and Grandpa. Mandine taught me the parable of the talents¹ and pushed me to reach the farthest shores. Grandpa always impressed me with his insatiable thirst of knowledge and he has passed part of it on to me.

My lifetime will not be long enough to thank you, Isabelle, for supporting me all along this journey. You (and Cyriac) are the most wonderful gift I may ever have dreamed of.

And finally, I would like to thank you, reader for you ensure that “*all those moments will not be lost in time, like tears in rain*”².

¹*The Holy Bible*, Matthew 25:14-30 or Luke 19:12-27

²Roy Batty in *Blade Runner*, slightly altered though.

Abstract

Every security design involves choosing adequate parameters. When dealing with cryptography, one of the most important choices is the length of the key to be used. Such emphasis comes from Kerckhoffs' principle stating that "the security must only rely on the key" while the system should be considered public knowledge. This is indeed the case for our public standards, anyone can access their specifications. However, in those standards, no information is given about how to choose the key length given an application's security requirements.

Using overwhelming large keys in order to work on the safe side is not the way to go as it crushes performances, especially for asymmetric schemes. A commonly used solution is to use recommendations from known and famous organizations, like the NIST or ECRYPT. Another option is to use the mathematical model proposed by Lenstra and Verheul to derive a key length corresponding to a desired security margin.

Questions related to the appropriate selection of keys are at the basis of the work contained in this dissertation. First, we made the proposal of Lenstra and Verheul available on www.keylength.com, together with other recommendations.

Afterwards, we pointed out that Lenstra and Verheul's analysis is based on software data points only. Therefore, we decided to introduce a new data point for hardware in order to compare it with software. For this purpose, we mounted an attack against an elliptic curve discrete logarithm problem using a cluster of low-cost FPGAs.

Then, we noted that only mathematical attacks were considered, that is without taking into account any implementation defect of which an attacker could take advantage. Such a defect led to the now largely studied side-channel and fault attacks. We explored the potential threat fault attacks really represent and dealt with the more pernicious kinds of fault attacks that aim at disturbing the public elements used in discrete logarithm based schemes.

Finally, we studied how keys can be generated, but from a novel perspective: fuzzy extractors. This recently introduced tool allows building reproducible keys usable in cryptography from physics where measurements are error prone. Using them, we built a system to strongly link information to its medium, using secure paper and low-cost hardware. This shows that physics cannot only be used to break security systems but also to build them.

Contents

Introduction to Cryptographic Key Length	1
1 Classical Cryptanalysis Example:	
Solving the Elliptic Curve Discrete Logarithm Problem	9
1.1 Introduction	9
1.2 Previous work	11
1.3 Mathematical Background	11
1.4 Hardware Background	12
1.5 Collision Search	13
1.6 Collision Search Architecture	15
1.7 Elliptic Curve Arithmetic	17
1.8 Elliptic Curve Processors	19
1.9 Chosen Elliptic Curve Processor	22
1.10 Hardware Results and Cost Assessment	24
1.11 Conclusion	27
2 Physical Cryptanalysis Example:	
Fault Attacks on DLP-based Schemes	35
2.1 Introduction	35
2.2 A New Fault Attack on DSA	37
2.3 A New Fault Attack on ElGamal Signature Algorithm	41
2.4 Open Discussion about Recovering <i>Hits</i>	44
2.5 Conclusions and Future works	45
3 Turning Physics to Advantage with Fuzzy Extractors	51
3.1 Introduction	51
3.2 Overview	53
3.3 Ingredients	54
3.4 The Paper Case	58
3.5 Conclusion	67
Conclusion	73
Publications	77

Appendix

A	Modular Division	81
A.1	Introduction	81
A.2	Elliptic Curve Operations over $\text{GF}(p)$	82
A.3	Algorithm	83
A.4	Implementation	85
A.5	Results	88
A.6	Conclusion	90
B	Montgomery Modular Inversion	95
B.1	Introduction	95
B.2	Elliptic Curve Operations over $\text{GF}(p)$	96
B.3	FPGA Constraints	96
B.4	The Montgomery Modular Inverse	97
B.5	Implementation Results	102
B.6	Conclusion	103
C	Stream Ciphers on FPGA	109
C.1	Introduction	109
C.2	Ciphers Description	110
C.3	Hardware Implementation	112
C.4	Implementation Results	114
C.5	Conclusion	116
D	AES on Virtex5	121
D.1	Introduction	121
D.2	Cipher Description	122
D.3	Target Platform	123
D.4	Architecture	124
D.5	Results	128
D.6	Conclusion	130

List of Figures

1.1	Available slice configurations.	12
1.2	Top half slice of a Xilinx Virtex-II.	12
1.3	Pollard's ρ method.	14
1.4	Parallel search amongst different ρ	14
1.5	Elliptic curve processors (EC- μ P).	16
1.6	Medium processor, <i>Point Update</i> unit (squarers for $m = 163$)	23
2.1	Bound (2.4) for all r power of prime smaller than 5000.	40
2.2	$\log_2(\text{lcm}(\{r_j\}))$ for all r_j which have probability larger than $b_s = 10^{-s}$ for $4 \leq s \leq 10$	40
2.3	$\log_2(\text{lcm}(\{r_j\}))$ for all r_j which have probability larger than $b_s = 10^{-s}$ for $4 \leq s \leq 10$	43
3.1	Making paper.	53
3.2	Extracting physical features.	53
3.3	Adding content.	53
3.4	Signing container + content.	53
3.5	Verification when the owner of the document is trusted.	54
3.6	Full verification with extraction of the medium's physical features.	54
3.7	A datamatrix encodes large amount of data in small areas.	55
3.8	Turning a secure sketch into a fuzzy extractor.	57
3.9	Comparing the capacitance inter- and intra-class distributions.	57
3.10	Using the data helper W to recover a key from a measurement.	57
3.11	Global view of the scheme: Enrollment and Reconstruction phases.	58
3.12	Splitting fiber-enhanced paper into two, four and eight sensors.	59
3.13	Characterizing the inter-class distribution and Gray codes.	59
3.14	Intra- and inter-class distributions for the amount of fibers N	60
3.15	Intra- and inter-class distributions for the orientation of the fibers O	61
3.16	Success rate <i>without</i> error-correcting codes (N feature).	62
3.17	Success rate <i>with</i> error-correcting codes (N feature).	62
3.18	Success rate <i>without</i> error-correcting codes (4 features).	63
3.19	Success rate <i>with</i> error-correcting codes (4 features).	63
3.20	Success rate with ECC: BCH(15, 7, 2) (4 features).	64
3.21	Success rate with ECC: BCH(31, 16, 3) (4 features).	64

3.22	Success rate <i>without</i> ruler (4 features, 10 sheets, BCH(7, 4, 1)).	64
3.23	Success rate <i>with</i> a ruler (4 features, 6 sheets, BCH(7, 4, 1)).	64
3.24	One set to estimate the distribution and build the bins.	65
3.25	Each sheet of the second set is placed in its corresponding bin.	65
3.26	Sensor-based entropy estimation using histograms for the N feature. . . .	66
A.1	stage AB	86
A.2	stage UV	86
A.3	counter P	87
A.4	counter D	87
A.5	Typical pipeline organisation	88
A.6	Modification of the 2 MSB input of each stage	88
B.1	A simplified Xilinx Virtex Logical Element	97
B.2	Early implementation for UV stage, more ASIC related	100
B.3	Montgomery inverse operative part of the basic UV stage	101
B.4	Montgomery inverse operative part of the basic RS stage	101
C.1	Available slice configurations.	112
C.2	Top half slice of a Xilinx Virtex-II.	112
D.1	SubBytes transformation.	122
D.2	ShiftRows transformation.	122
D.3	MixColumns transformation.	123
D.4	AddRoundKey transform	123
D.5	AES 128-bit key expansion round.	123
D.6	Virtex-5 mixed view of top-half sliceL and bottom-half sliceM.	125
D.7	AES encryption architecture.	126

List of Tables

1.1	Area and throughput estimations for each processor	22
1.2	Scheduling of point additions for $m = 163$, without initialization	25
1.3	Medium EC- μ P, Place and Route results from Xilinx ISE 9.1.	26
1.4	Cost assessments to solve different $\text{GF}(2^m)$ ECDLP using COPACOBANA	27
2.1	Toy example of the attack on DSA	38
2.2	Experimental software results.	41
2.3	Experimental software results.	43
3.1	Entropy estimations in entropy bits per fingerprint X	66
A.1	Timings of different coordinate systems	83
A.2	FPGA implementation results	89
A.3	Performance comparisons	89
B.1	Virtex-E and Spartan3 implementation results	102
B.2	Performance comparisons	103
C.1	Implementation results.	115
C.2	Previous block and stream ciphers implementations.	115
D.1	Implementation Results: encryption/decryption designs.	129
D.2	Previous implementations.	129

A Short Introduction to Cryptographic Key Length

Every security design involves choosing adequate parameters. When dealing with cryptography, one of the most important choice is the length of the key to be used. Such emphasis on the key dates back to 1883, when Auguste Kerckhoffs wrote his article “La Cryptographie Militaire” [Ker83], where he gives the following requirements for a secure communication system:

1. The system must be undecipherable;
2. The system must not require secrecy and can, without trouble, fall in enemy’s hands;
3. The keys must be easy to exchange and to remember (without requiring written notes), and can be changed and modified at will;
4. The system must be compliant with telegraph communication;
5. The system must be portable, and its handling must not require more than one person;
6. Given the application’s circumstances, the system must be easy to use, without requiring intense brain activity nor the knowledge of a long series of rules.

Emphasizing that secrecy is the main defect of all the proposed systems at that time, Kerckhoffs has surely influenced the way his principle is known today. The statements are often summarized in a single one: “The security must only rely on the key” while the system should be considered public knowledge. This is indeed what is observed in our public standards. For example, everyone can access the description of the widely used symmetric block cipher, the Advanced Encryption Standard. The AES is able to handle three different key sizes: 128, 192 and 256 bits. For asymmetric system, taking RSA as an example, the key length can greatly vary, from 1024 to anyone’s desired size. However, in those standards, no information is given about how to choose the key length given an application’s security requirement.

A tempting approach would be to use an overwhelming long key, hence working on the safe side by an unknown but assumed large enough margin. Let us observe what happens when doubling the length of the key for both cases, that is moving from AES-128 to AES-256 and from RSA-1024 to RSA-2048.

AES From the standard, the main difference lies in the number of rounds, from 10 to 14 rounds keeping the same structure. The key schedule remains pretty much the same and its cost quickly vanishes when enciphering multiple blocks of data with the same key. Therefore, it is not taken into account. Going from 128-bit key to 256-bit key yields a 40% increase in the number of operations.

RSA The most expensive operation of RSA is the full modular exponentiation. Depending on the algorithm used, the number of multiplications can vary but all of them imply n squarings (where n is the size of the modulus). Multiplications and squarings have complexity $O(n^2)$ using the method learnt at school and $O(n \log n)$ using FFT. Therefore, switching to a $2n$ -bit modulus gives a 300% increase in the number of operations.

These small evaluations show that increasing the key length involves additional computation time. Nowadays this may sound negligible as our computers can easily handle them. However, this is not the case for small devices and even sometimes impossible in really small devices³. This shows that choosing the appropriate key length is of great importance and great care should be taken when dealing with the security/performance trade-off it involves. This choice is usually left to the security expert and is driven by his own knowledge and experience.

In order to fill in the gap, Arjen Lenstra and Eric Verheul proposed a model to derive key length corresponding to a desired security margin [LV99]. Before giving some insight about this model, some remarks are given.

First, only the best mathematical attacks are considered. That recommendation arising from the model does not take protocol flaws, password deficiencies nor human failures into account. Therefore, the choice of the key length should draw the attention only once the overall security of the design is convincing.

Then, the suggestions given are based on the observation of cryptography during the last decades. As no particular breakthrough appeared, it seems reasonable to hope none will happen. Of course, this might prove wrong and all cryptography (or part of it) may collapse. Only cryptosystems for which such a disaster is believed to be unlikely to happen are taken into account. For those systems, cryptanalysis is assumed to progress at an unsteady but regular pace against which stretching the key remains an effective countermeasure. How long should those key be lengthened is another aspect of Lenstra and Verheul's work. Furthermore, neither quantum computer nor molecular computing are taken into account for the first one would ruin (a largely used part of) asymmetric cryptography [Sho94] if they could be built and the second one can definitely be considered impractical at the time of writing.

Finally, building such a model is doomed to criticisms as soon as a particular choice does not suit the reader's feeling. To accommodate for this, all the parameters of the model can be tuned according to anyone's taste.

For each of the cryptographic primitives studied, information is gathered, comprising a short description, the best attack known, software and hardware data points (when available), the effectiveness of guessing the key, the success probability of an incomplete

³Whether or not an inexpensive RFID tag can perform an AES encryption is still an open debate.

attack and speculation about the cryptanalytic progress in the area given its evolution. The aim of this dissertation is certainly not to provide a description of all the work that was done in [LV99]. To give an idea on how the model works, the case of symmetric key will follow. Symmetric key system representatives include IDEA, RC5, AES and the DES⁴ that should no longer be used, but is at the basis of the estimations.

No description of block cipher is given here, DES is specified in [FIPS463] while AES is specified in [FIPS197] or in Appendix D. For the sequel of the discussion, it is worth keeping some data in mind: the assumed best attack known against symmetric systems is exhaustive key search. In case a shortcut is discovered allowing a faster attack, it is assumed that the algorithm is changed in favour of another one where exhaustive search is the best attack. In the case of the DES where the key is 56-bit long, the expected number of trials for this attack is 2^{55} . Such a key recovery was successfully achieved in software in 1997. It took around 4 months of a distributed internet search, such a computing power is estimated at 0.5 MMY (Million Mips-Year).

The model is briefly sketched here, the interested reader will find all the details in their carefully written paper [LV99]. The selection of the key length mainly depends on the following four points:

Life span reflects the user's wishes to protect the data until some year y . Depending on the value of those data, the user may go for a short-, medium- or long-term protection.

Security margin offers the user to decide how loose or tight is his perception of a sufficient unfeasible attack. It is defined as the year until which the user is willing to trust the DES and is defaulted to $s = 1982$.

Computing environment takes into account the increase in computational power an attacker can get as time goes by. A usual way to do so is to follow Moore's law, stating that the computational power doubles every 18 months⁵. Whether this *law* can hold any longer is a matter of personal belief. Therefore, a more flexible way to handle this is to define m as the amount of time (in months) it takes to double speed and memory. Default value is $m = 18$. In addition, it can be made less dependent on the technology evolution by introducing another parameter, t , expressing whether such doubling is relative to the price you pay it or not. In case $t = 1$, that is the increase in computational power you get for one dollar doubles every m month, it is worth estimating the budget increase too. This is captured by parameter b , defined as the amount of time (in years) it takes to get a budget doubled.

Cryptanalysis estimates the progress that can be expected to improve attacks. It is formulated in the same fashion as Moore's law, by defining both r and c as the amount of time (in months) expected for attacks to become twice as effective for classical

⁴This is of course not an exhaustive list.

⁵It seems the right way to formulate it is that the density of integrated circuits (number of transistors per area unit) doubles every 18 months. But "doubling the computing power every 18 month" is a commonly used interpretation.

asymmetric systems⁶ and elliptic curve systems respectively. The default setting are $r = 18$ and $c = 0$.

In order to derive a formula that will estimate the key length required to protect data up to a given year y , the Infeasible number of Mips-Year (IMY) corresponding to that year y is computed as:

$$\text{IMY}(y) = \underbrace{5 \cdot 10^5}_1 \cdot \underbrace{2^{12(y-s)/m}}_2 \cdot \underbrace{2^{t(y-s)/b}}_3. \quad (1)$$

The first term is the estimated effort (0.5 MMY) to break the DES in 1997 and is so assumed to be infeasible in 1982. The second one is the estimated increase in computation power from 1982 to y and the third one takes into account the fact that you can afford for more computational power. Therefore, equation 1 estimates the amount of computation that is infeasible for year y . Such a value is at the basis of the derivation of key lengths suitable to protect data till year y .

For the case of symmetric key, another parameter, v , is introduced. It links the speed difference between the DES and the block cipher actually used. Indeed, if the symmetric encryption scheme used is twice as fast as the DES, the exhaustive key search is also going to be twice as fast, therefore requiring an even larger key. The lower bound for the key length is

$$56 + \log_2 \frac{\text{IMY}(y)}{5 \cdot 10^5 \cdot v} = 56 + (y - s) \left(\frac{12}{m} - \frac{t}{b} \right) - \log_2 v \quad (\text{in bits}), \quad (2)$$

where 56-bit is the key length of the DES and the second part is the number of bits that should be added in order to render an exhaustive search unfeasible.

Note that equation 2 does not make use of neither r nor c , the expected cryptanalytic progresses. Indeed, as already stated, in case the chosen symmetric cipher is subject to a faster attack than a brute force attack, it is assumed that such acipher will be replaced by another one not suffering from such a flaw. No cryptanalytic progress is expected. However, those parameters are used for asymmetric systems.

The same kind of process is performed for asymmetric key primitives and hash functions in order to derive formulas to somehow extrapolate on the key length to achieve acceptable security till a given year. All the details were not given here, the paper of Lenstra and Verheul contains much more. To make those estimates easily available, we – Damien Giry and I – gathered them through the `keylength.com` website [GB]⁷ together with some recommendations from other sources like the NIST, ECRYPT, and so on.

Questions related to the appropriate selection of keys is at the basis of the work contained in this dissertation. First, we address the “software versus special-purpose hardware attacks” conversion problem untold in previous introduction to Lenstra and Verheul’s model. Their analysis is based on software data points only, even though hardware data

⁶It is worth mentioning that “classical asymmetric systems” in this context regroupes RSA and Traditional Discrete Logarithm (TDL).

⁷Damien is the current maintainer of `keylength.com`.

points are mentioned. They do note that such conversion “*is a mere coincidence without practical merit*”. We decided therefore to introduce a new data point for hardware in order to confront such opinion. This is the subject of chapter 1, Solving the Elliptic Curve Discrete Logarithm Problem, where an attack is mounted against an ECC challenge using a cluster of FPGAs.

Then, we note that only mathematical attacks are considered, that is without taking into account any implementation defect of which an attacker can take advantage. Such a defect leads to the now largely studied side-channel and fault attacks. Both fields continue to receive great attention from the community. We explore the potential threats fault attacks really represent. Chapter 2, Fault Attacks on DLP-based Schemes, deals with the more pernicious kinds of fault attacks that aim at disturbing the public elements used in schemes based on the Discrete Logarithm Problem.

Finally, we study how keys can be generated, but from a novel perspective: fuzzy extractors. This recently introduced tool allows building reproducible keys usable in cryptography from physics where measurements are error prone. Using them, we build a system to strongly link information to its medium, using secure paper and low-cost hardware. This shows that physics can not only be used to break security systems, as in Chapter 2, but also to build them, as in Chapter 3: Turning Physics to Advantage with Fuzzy Extractors.

Bibliography

- [ECRYPT] ECRYPT. European Network of Excellence for Cryptology. <http://www.ecrypt.eu.org/>.
- [FIPS197] National Institute of Standards and Technology. Advanced Encryption Standard (AES). Federal Information Processing Standards Publications – FIPS 197, <http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf>, November 2001.
- [FIPS463] National Institute of Standards and Technology. Data Encryption Standard (DES). Federal Information Processing Standards Publications – FIPS 46-3, <http://csrc.nist.gov/publications/fips/fips46-3/fips46-3.pdf>, November 1999.
- [GB] Damien Giry and Philippe Bulens. Cryptographic Key Length Recommendation. <http://www.keylength.com>.
- [Ker83] Auguste Kerckhoffs. La Cryptographie Militaire. *Journal des Sciences Militaires*, 9th serie:(Jan) pp 5–38 and (Feb) pp 161–191, 1883. <http://www.petitcolas.net/fabien/kerckhoffs/>.
- [Len05] Arjen K. Lenstra. *Key lengths in Handbook of Information Security*, volume 2. Wiley, 2005. pp 617–635.
- [LV99] Arjen K. Lenstra and Eric R. Verheul. Selecting Cryptographic Key Sizes. *Journal of Cryptology*, 14:255–293, 1999.
- [NIST] NIST. National Institute of Standards and Technology. <http://www.nist.gov>.
- [Sho94] P. W. Shor. Algorithms for Quantum Computation: Discrete Logarithms and Factoring. *Proc. 35th Annual Symposium on Foundations of Computer Science*, pages 124–134, IEEE Computer Society Press 1994.

Classical Cryptanalysis Example: Solving the Elliptic Curve Discrete Logarithm Problem

In this last decade, Elliptic Curve Cryptography (ECC) has gained increasing acceptance in the industry and the academic community and has been the subject of several standards. This interest is mainly due to the high level of security with relatively small keys provided by ECC. Indeed, no sub-exponential algorithms are known to solve the underlying hard problem: the Elliptic Curve Discrete Logarithm.

The aim of this work is to explore the possibilities of dedicated hardware implementing the best known algorithm for generic curves: the parallelized Pollard's ρ method. This problem has specific constraints and requires therefore new architectures. Four different strategies were investigated with different FPGA families in order to provide the best area-time product, depending on the capabilities of the chosen platforms. The approach yielding the best throughput over hardware cost ratio is then fully described and was implemented in order to estimate the cost of an attack. Such results should help improving the accuracy of the security level offered by a given key size, especially for the shorter parameters proposed for resource constrained devices.

1.1 Introduction

Since their introduction in cryptography in 1985 by Neal Koblitz [Kob87] and Victor Miller [Mil86], elliptic curves have raised increasing interest. This rich mathematical tool has been used to set up new asymmetric schemes able to compete with the well established RSA. Such schemes allow many useful functionalities like digital signature, public key encryption, key agreement, ... For those needs, Elliptic Curve Cryptography (ECC) is indeed an attractive solution as the provided public key scheme is currently one of the most secure per bit.

The underlying hard problem of ECC is the intractability of the Elliptic Curve Discrete Logarithm Problem (ECDLP). Let $E(\mathbb{F})$ be an elliptic curve over a finite field $\mathbb{F}$ and let P be a point of $E(\mathbb{F})$. For any point $Q \in \langle P \rangle$ (the subgroup generated by P), the problem is to determine the integer k , $0 < k < n$, (with n the order of P) verifying $k \cdot P = Q$.

In day-to-day life utilizations of public key cryptography, choosing security parameters is a major concern¹. Knowing the difficulty of solving ECDLP instances is therefore essential to reach good trade-offs between security and computation power. Until 2006, the only released attacks were performed on general purpose processors. Hardware platforms can obviously be lower power and faster than software ones but the cost improving factor, for given $\text{GF}(2^m)$ problems, is currently unknown. The aim of this work is therefore to evaluate the complexity and the cost of solving ECDLP by means of dedicated hardware. Such results should help improving the accuracy of the security level offered by a given key size, especially for small key proposed for resource-constrained devices [CRb].

This work focusses on curves over $\text{GF}(2^m)$ instead of $\text{GF}(p)$. Indeed, $\text{GF}(2^m)$ arithmetic is well suited for hardware platforms (addition is a simple bitwise xor) while software platforms are optimized for $\text{GF}(p)$. A higher improvement factor over software solutions is therefore expected. A survey about hardware for attacking ECC, based mainly on results of [GPP07] about $\text{GF}(p)$ curves can be found in [Pel06].

The framework here is a general attack on curves over $\text{GF}(2^m)$, independently of the representation of the underlying group. Specific attacks, like MOV [MOV91], Pohlig-Hellman [Men93] and the exploitation of weak fields $\text{GF}(2^{mn})$ [GHS02, MTW04] are not handled. For general attacks, the methods of interest are Pollard's ρ [Pol78] or Shanks' Baby-step Giant-step [Coh93]. The algorithm used in this work is the parallelized ρ method of van Oorschot and Wiener [vOW99] with Teske's observations [Tes00].

For the hardware platform, FPGAs were preferred over ASICs. Indeed, today's FPGAs got rid of most of the electrical and thermal problems they were suffering from in the last half decade. Prices for large FPGA devices are now also very affordable. As a result, their flexibility and performance over cost (for low volume) makes FPGAs an attractive solution. In particular, the FPGA-based COPACOBANA engine [KPP⁺06] was used for the cost assessment of this work.

Existing ECC architectures cannot be easily tailored to meet the specific requirements of the problem (cf. 1.5.2). New architectures with new trade-offs have therefore to be considered. In this work, four kinds of processors for solving ECDLP were investigated on different FPGA device families: *tiny* and *small* with low area requirements for low-cost FPGAs, *medium* for low-cost FPGAs and *large* for high-performance FPGAs. The aim is to provide architectures with different area-time complexities in order to best fit the capabilities of the chosen platforms. As a result, a high-performance device is not reduced to the sum of small devices. For that purpose, an original approach based on a prior theoretical analysis of area requirements of algorithms was used. This method allows selecting the best options before implementations. Then, based on preliminary implementation results, the best processor is selected and fully described. Finally, timings of a software-based implementation are used to compare the performance-cost ratio of hardware platforms and general purpose processors.

This chapter is structured as follows: Section 1.2 deals with previous attempts for solving ECDLP. Section 1.3 reminds the mathematical background of elliptic curves. Then, Section 1.5 explains the algorithm and improvements used for the collision search. The

¹A platform gathering current recommended key sizes can be found in [GB].

description of the architecture of the whole system stands in Section 1.6. After that, algorithms necessary for the arithmetic on elliptic curve are studied in Section 1.7. Based on this theoretical analysis, the four kinds of processors are presented in Section 1.8. Afterwards, the most efficient processor is fully described in Section 1.9. The hardware results and cost assessments are presented in Section 1.10 and finally, conclusions are given in Section 1.11.

1.2 Previous work

Until now, the hardest Certicom challenges [CRa] solved have been performed on 109-bit fields using general purpose processors. The ECC2-109 challenge was solved by the team of Chris Monico in 2004. The effort required 2600 computers and took 17 months. The gross CPU time used was estimated equivalent to that of an Athlon XP 3200+ working nonstop for about 1200 years [CRa]. Those results suggest the use of dedicated hardware for attacking higher fields like ECC2-131 for example.

A rough estimation about using dedicated hardware for solving ECDLP on curves over $\text{GF}(2^{155})$ with elements represented on an Optimal Normal Basis (ONB) can be found in [vOW99]. Their aim was to show that the setup of [AMV93] was insecure: that a largest prime factor with a size of only 2^{120} was insufficient. With a budget of \$10 million for their $1.5 \mu\text{m}$ ASICs clocked at 40 Mhz, the authors estimated that solving this particular ECDLP should only take 32 days².

Unlike this work, we focus on polynomial basis with low weight irreducible polynomials (as those recommended in standards like [FIPS182, CRb]). Further analysis of ONB-based systems is left for another work.

The first results about concrete hardware implementation were presented in [GPP07] for $\text{GF}(p)$ and in [BMQ06] for $\text{GF}(2^m)$. Compared to the (unpublished) results presented in [BMQ06], this work completes the study in many aspects. In particular, in addition to a *large* processor for high-performance FPGAs, *small* processors for low-cost FPGAs are studied. Moreover, power consumption and performance analysis in terms of the throughput-hardware cost ratio are given.

1.3 Mathematical Background

Let $p(z) \in \text{GF}(2)[z]$ be an irreducible polynomial of degree m generating the field $\text{GF}(2^m)$. A non supersingular elliptic curve E over $\text{GF}(2^m)$ using affine coordinates can be defined as the set of solutions of the reduced Weierstraß equation:

$$E : y^2 + xy = x^3 + ax^2 + b$$

where $a, b \in \text{GF}(2^m)$, $b \neq 0$, together with the point at infinity $\mathcal{O}$.

The inverse of point $P = (x_1, y_1)$ is $-P = (x_1, x_1 + y_1)$. In affine coordinates, the sum $P + Q$ of points $P = (x_1, y_1)$ and $Q = (x_2, y_2)$ (assuming that $P, Q \neq \mathcal{O}$ and

²However, the processor should be used in affine mode instead of projective (cf. 1.5.2).

$P \neq -Q$) is point $R = (x_3, y_3)$ where:

$$\begin{cases} x_3 &= \lambda^2 + \lambda + x_1 + x_2 + a \\ y_3 &= \lambda \cdot (x_1 + x_3) + x_3 + y_1 \end{cases}$$

With

$$\text{with } \lambda = \begin{cases} \frac{y_1 + y_2}{x_1 + x_2} & \text{when } P \neq Q \\ \frac{y_1}{x_1} + x_1 & \text{when } P = Q \end{cases}$$

Those modular arithmetic computations involve an expensive division, a multiplication, a squaring and several additions. When performing a scalar multiplication, the inversion is usually deferred to the end of the whole computation by using different coordinate systems [CMO98]. However, as an invariant is required while solving ECDLP, the use of affine coordinates is cheaper (cf. 1.5.2).

For a thorough description of elliptic curves, the reader is referred to [BSS99].

1.4 Hardware Background

For the sake of simplicity, let us say that an FPGA is a well structured array of slices³. The slice is the logic unit that is used to evaluate a design's area requirements. Such a slice, depicted in Fig. 1.1 is made of 2 Look-Up Tables (LUTs), 2 flip-flops and a few additional gates. According to the user's choice, any of these LUTs can be configured in one out of three possible ways: RAM16 that acts like a 16-bit RAM storage, SRL16 that implements a 16-bit linear shift register and LUT that is capable of computing any 4-to-1 boolean function. A more detailed view of half a slice is given in Fig. 1.2. This is used as the basic building block of every design.

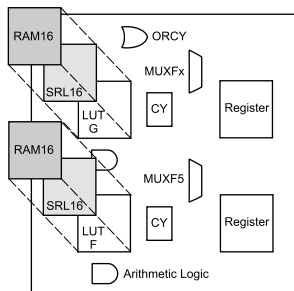


Figure 1.1: Available slice configurations.

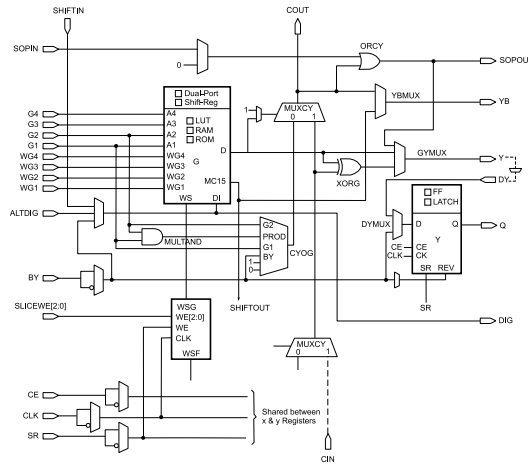


Figure 1.2: Top half slice of a Xilinx Virtex-II.

³For simplicity purpose, real FPGAs are more complex and can even embed microprocessors within their structure.

1.5 Collision Search

The main algorithms for solving a generic ECDLP are Pollard's rho [Pol78] and Shanks' Baby-step Giant-step [Coh93] methods. However, Pollard's rho has the great advantage of requiring only a little amount of memory. The parallelized version of this method, with distinguished points, was chosen for this work as it is the best known algorithm to solve a generic ECDLP instance.

1.5.1 Pollard's ρ Algorithm

The basic idea to solve DLP is to walk in the group as randomly as possible until a collision is found, i.e. once a group element is reached twice, coming from different ways. The algorithm shown here is a general adaptation of Pollard's method to the case of the ECDLP.

To recover the unknown k , with $Q = k \cdot P$, a random chain is initialized by a point $R_0 = c_0 \cdot P + d_0 \cdot Q$. Then, the following function is iteratively applied:

$$R_{i+1} = \begin{cases} R_i + M_u & \text{when } R_i \in T_u \\ 2 \cdot R_i & \text{when } R_i \in T_v \end{cases}$$

for some point $M_u = e_u \cdot P + f_u \cdot Q$ with e_u, f_u randomly chosen. T_u (resp. T_v) is the set of partitions for which an addition (resp. doubling) is performed. In order to solve the DLP, c and d have to be updated accordingly:

$$c_{i+1} = \begin{cases} c_i + e_u & \text{when } R_i \in T_u \\ 2 \cdot c_i & \text{when } R_i \in T_v \end{cases} \quad \text{and} \quad d_{i+1} = \begin{cases} d_i + f_u & \text{when } R_i \in T_u \\ 2 \cdot d_i & \text{when } R_i \in T_v \end{cases}$$

As time goes by, the probability to find $R_i = R_j$ increases, yielding the value of the scalar multiplier k :

$$\begin{aligned} c_i P + d_i Q &= c_j P + d_j Q \\ c_i P + d_i k P &= c_j P + d_j k P \\ k &= \frac{c_j - c_i}{d_i - d_j} \bmod \# \langle P \rangle \end{aligned}$$

An appropriate number of T_u and T_v partitions has to be chosen. Teske's experimental results [Tes00] suggest that setting $u \geq 20$ and $v = 0$ yields performances close to the optimal "true" random walk. This is particularly interesting as it allows the point addition to be solely implemented. In practice, a power of 2 is a convenient choice for u : a small chunk of bits of the point can be regarded as the index of the partition.

1.5.2 Improvements

The improvements related to Pollard's ρ method of solving ECDLP are given here. For the sake of completeness, they are all discussed even though only a part of them are included in our final design. To differentiate between them, a ✓ tags the improvements taken into account, while a ✗ denotes room left for potential further work.

✓ – **Distinguished Points:** The concept of Distinguished Points (DPs), attributed to Rivest by Denning [Den82], can be included to improve the overall efficiency of the algorithm. A distinguished point criterion, or property, is chosen and each computed point satisfying this criterion is stored. The collision search is now limited to a comparison between the distinguished points.

However, a given point can be represented in many ways using different coordinate systems. Without an invariant in point representation, DP criterion cannot be checked. Moreover, chains do not merge after a collision, preventing its detection while reaching a DP. To our knowledge, there is no computationally cheaper invariant than a modular division. As a result, affine coordinates are more efficient as point representation in this system is unique.

A well chosen DP criterion⁴ will ensure that enough DPs are encountered to limit the number of steps ($\frac{1}{\theta}$ term below), but not too much (as those DPs are stored). More precisely: if θ is the proportion of points in $\langle P \rangle$ having the DP property, the expected number of elliptic curve operations before a collision of DP is observed is $\sqrt{\frac{\pi \cdot n}{2}} + \frac{1}{\theta}$.

To deal with the case where a chain falls into a loop without DP, chains exceeding 20 times the average length are dropped. This corresponds to a proportion of $(1 - \theta)^{20/\theta} \approx e^{-20}$ chains with a waste of work of $20 e^{-20} < 5 \cdot 10^{-8}$ [vOW99].

✓ – **Parallelization:** Using a large number of computers, van Oorschot and Wiener [vOW99] showed that it was interesting to make them share their knowledge. The collision search can therefore be viewed as a search between several chains, as shown in Fig. 1.4, instead of only one, see Fig. 1.3. The complexity is reduced by the full number of computers C taking part in the collision search. Resorting to DP is particularly useful with the parallelization technique.

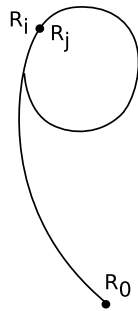


Figure 1.3: Pollard's ρ method.

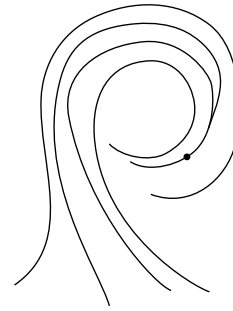


Figure 1.4: Parallel search amongst different ρ .

✗ – **Negation Map:** For elliptic curves over both $\text{GF}(p)$ and $\text{GF}(2^m)$, given a point $P = (x, y)$, its opposite can easily be computed by either $-P = (x, -y)$ or $-P = (x, x \oplus y)$. Wiener and Zuccherato [WZ99] made the clever observation that this can reduce the cost of a collision search. Actually, each computed point during the search gives (almost)

⁴As pointed out in [Sma00], some DP criteria lead to inefficient attacks on $\text{GF}(2^m)$.

immediately another point: its opposite. As a result, each step produces two points. This allows limiting the collision search to one half of the space.

This is practically done by replacing R_i by the point $\pm R_i$ with the smallest y -coordinate, for some notion of small in $\text{GF}(2^m)$. Accordingly, the coefficients c_i and d_i are replaced by $\pm c_i$ and $\pm d_i$ to ensure the relation $R_i = c_i \cdot P + d_i \cdot Q$ is maintained. For a discussion about trivial 2-cycles, the reader is referred to [WZ99].

As the running time is in the square root of the group order, this trick reduces the time complexity by a factor $\sqrt{2}$.

X – Frobenius Map: If the ECDLP has to be solved for the specific case of anomalous binary curves, also called Koblitz curves, another improvement is available [WZ99]. This kind of curves, first proposed for cryptographic use in [Kob91], provides an efficient implementation of the scalar multiplication when the scalar is represented as a complex algebraic number. Those curves are defined by $E : y^2 + xy = x^3 + ax^2 + 1$, with either $a = 0$ or $a = 1$ and coordinates over $\text{GF}(2^m)$. The advantage provided by the Koblitz curves is that point doubling can be easily performed by the Frobenius map operation.

Consider the trace equation $\tau^2 + 2 = \mu\tau$ where $\mu = (-1)^{1-a}$ with a being the curve parameter. The solution of this equation is $\tau = \frac{\mu + \sqrt{-7}}{2}$. If point $P = (x, y)$ is on a Koblitz curve, the application of the Frobenius map $\tau P = (x^2, y^2)$, produces a point on the same curve. As a result, a point doubling only needs 2 squarings instead of 2 multiplications, 2 squarings, 1 inversion and 5 additions (in affine coordinates).

As explained in 1.5.1, Teske recommends using only point additions. This operation will therefore not be used for the random walks but to efficiently compute all the points equivalent to a given point. Practically, after each point addition, the Frobenius map is applied $m - 1$ times over the current point. The resulting point yielding the smallest x -coordinate is kept, again for some notion of small in $\text{GF}(2^m)$. The negation map could then be applied. As a result, m points, or $2m$ points are covered at each point addition for a *relatively* small hardware cost. This allows limiting the collision search in $1/m$ of the space or in $1/2m$ with the joint use of the negation map.

This is practically done by replacing R_i by the point $\pm \tau^j R_i$ with the smallest x -coordinate. Accordingly, the (complex) coefficients c_i and d_i are replaced by $\pm \tau^j c_i$ and $\pm \tau^j d_i$ to ensure the relation $R_i = c_i \cdot P + d_i \cdot Q$ is maintained. The powers of τ^j should be precomputed in order to increase the efficiency of the method.

As the running time is in the square root of the group order, the joint use of the negation map and the Frobenius map reduces the time complexity by a factor $\sqrt{2m}$.

1.6 Collision Search Architecture

The global collision search architecture is made of a server (e.g. a PC) and an arbitrary number of clients (for instance FPGAs). The hardware clients take care of the computationally intensive part of the work, namely the random walks on the elliptic curve. The software server handles the low-throughput operations: computation and dispatching of the starting points (SPs), recovery of the DPs, check for their correctness and sorting in

order to find a collision. A software-based platform is a natural choice as it can provide the resources to sort the DPs and a huge memory to store them.

1.6.1 Software Program

The software is organized as follows: python language is used as the high level programming language and wraps, with swigwin [SWIG], the low level C/C++ functionalities. The C++ modules use NTL [Sho] and deal with the number theory aspects while the C modules handle the communications. The performances of those modules are not critical.

1.6.2 Hardware Circuit

Each hardware client embeds a communication interface (basically a FIFO), a main controller and some numbers of elliptic curve processors with their own communication buffers (EC- μ P). A ring communication topology is the most scalable approach, especially for a set of many small EC- μ P on a large platform.

For the communication, each data set is made of a header, the x - and y -coordinates of a point and its c and d components in the P, Q basis. The header is used to determine the address of the EC- μ P and the kind of data carried (either SP or DP). A more detailed view of an EC- μ P is given in Fig. 1.5.

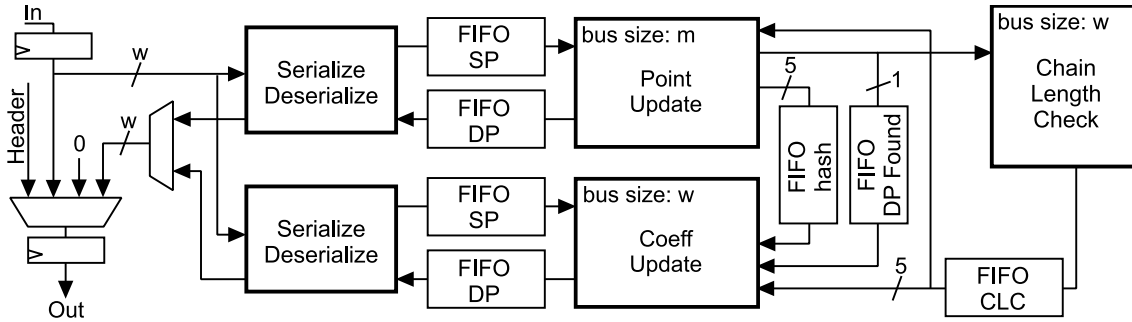


Figure 1.5: Elliptic curve processors (EC- μ P).

During the initialization phase, the data are sent from the main controller to the first EC- μ P which collects one SP for each chain. Additional SPs are stored to allow starting a new chain when it exceeds the allowed length ($20/\theta$). SPs are deserialized to feed FIFOs in order to load RAMs of the *Point Update* and *Coeff Update* computing units. In most architectures, the throughput of the *Coeff Update* unit is sufficiently low to work on w -bit data (which avoids a specific (de)serialize circuit). When a DP is found, the point and its coefficients are stored in other FIFOs in order to serialize and output the data as soon as possible. In this case, the corresponding chain length is reset as well.

The default behaviour of the EC- μ P is to compute the point additions, update the coefficients and track the length of the chains. The number u of partitions (cf. Section 1.5.1) was set to 2^5 in order to directly map the 5 LSBs of the x -coordinate (called hash) to 32 partitions. For each point addition, this hash is sent through a FIFO to the *Coeff Update* unit in order to update the two coefficients of each chain accordingly. The *Point Update*

unit also sent (through a FIFO) one bit for each chain to tell whether a DP was found. When a chain exceeds the allowed length, its ID is sent to the *Point Update* unit through another FIFO.

1.7 Elliptic Curve Arithmetic

This section reviews different algorithms for the operations needed to compute an EC point addition (PA). Hardware implementations usually focus on inversion-free coordinates (e.g. [AH08]) as they are potentially more area-time efficient in the high-speed domain. Nevertheless, affine coordinates are cheaper and all the trade-offs have therefore to be reconsidered. Moreover, as a lot of area is potentially available in the targeted circuits, a lot of area-time trade-offs are achievable. In order to select the best algorithms, this analysis begins by reviewing their theoretical area requirements in terms of FPGA *slices*.

In this work, low-weight binary irreducible polynomials of degree m are considered. Such trinomials and pentanomials are recommended for implementations in [CRb, FIPS182] and are used in challenges proposed in [CRa]. As sketched out in Section 1.2, addition in $\text{GF}(2^m)$ is a simple bitwise *xor* operation. Reconfigurable logic is used for the EC- μP , the polynomials can therefore be hardwired. As a result, each bit of the parallel modular reduction is computed with mainly 5-input *xor* gates for pentanomials and 3-input *xor* gates for trinomials. This leads to an inexpensive modular squaring circuit as the squaring itself is obtained by inserting a ‘0’ bit between consecutive bits of the binary representation of the input polynomial [HMOV04]. Other relevant operations are multiplication and inversion/division.

1.7.1 Squaring

Most of the bits of a hardwired parallel squarer are computed with 2-input and 3-input *xor* gates for pentanomials and 1-input (no gate) and 2-input gates for trinomials. This corresponds to an area of $m/2$ slices. For circuits composed of parallel multiple squarers, logical simplifications occur. As a result, they are more area-time efficient than the iterative use of a single squarer.

1.7.2 Multiplication

For an iterative architecture, a digit-serial by parallel multiplier [SP98] is an attractive solution. With a D -bit digit, the computation needs $\lceil \frac{m}{D} \rceil$ cycles. The circuit is composed of a register for the parallel input, a shift register for the serial input and a registered $2D + 1$ -input sum of product (*and-xor* network) for the accumulated partial product and output. The modular reduction can be neglected. The following areas, in function of D , are required: $3m/2$ (1), $2m$ (2,3), $5m/2$ (4), $3m$ (5,6), $7m/2$ (7), $4m$ (8,9), $9m/2$ (10), $5m$ (11,12), $11m/2$ (13), $6m$ (14,15), $13m/2$ (16).

For a parallel multiplier, a natural way to reach a sub-quadratic complexity is to use the so-called divide-and-conquer Karatsuba-Ofman Algorithm (KOA) [KO86]. This method recursively splits the problem into several smaller problems. More precisely, a $2^l \times 2^l$

multiplication is reduced to $3^k (2^{l-k} \times 2^{l-k})$ multiplications and additions. These small multiplications can then be achieved with simple array multipliers. More details on parallel implementation of the KOA for $\text{GF}(2^m)$ can be found in [RHK03] for example. Nevertheless, a slightly different and simpler approach has been chosen.

A first solution is to take a generic $2^l \times 2^l$ KOA multiplier, with $l = \lceil \log_2(m) \rceil$. The unused inputs are hardwired to 0 and the corresponding outputs are not connected. Thanks to the logic optimization provided by the synthesis tool, all the unnecessary logic is removed. While not strictly optimal, this solution is simple and already exhibits good performances. For the number of recursive calls of the KOA method, practical results show that $2^3 \times 2^3$ array multipliers provide good results.

A second solution is to use a KOA multiplier with a recursive division of the operands into two parts instead of parts being power of two. This is possible since there is no constraint about the size of the array multipliers. For instance, a $\text{GF}(2^{79})$ multiplication is designed to use 10×10 array multipliers. A 80×80 multiplier is therefore built and the unnecessary logic is removed by the implementation tools. If the size of the problem slightly exceeds what can be achieved with such a kind of array multipliers, the trivial multiplication method can be used at a higher level. For instance, a $\text{GF}(2^{163})$ multiplication is designed to use 10×10 array multipliers for a 160×160 KOA multiplier. Then, two 160×3 and one 3×3 array multipliers are added in order to achieve the 163×163 multiplication.

This second solution performs slightly better than the first. Moreover, it is easier to handle for the implementation tools. Experimental results show that the area required by the modular multiplication is roughly smaller than $(m/2)^2$ FPGA slices.

1.7.3 Extended Euclidean Division

A divider based on the binary extended Euclidean algorithm [Ste67] was presented in [WWSH04]. An iterative approach requires $2m - 1$ cycles to manipulate the 4 variables involved in the computation. Two of them, U and V , are required for the binary GCD transformation over the divisor. These transformations are then mirrored on the R and S variables to compute the result. The area complexity of this approach is $2m$ slices [MQ06].

At the other extremity, a fully pipelined circuit would require to fully unroll the $2m - 1$ iterations of the loop. In the worst case scenario, the maximum degree of both U and V starts decreasing after m iterations. As a result, the area complexity of this block is $m^2 + \frac{m^2}{2}$ slices. For the R and S variables, the full size is immediately reached while computing a division. Therefore, $2m^2$ slices are required for this second block. While computing an inversion, the area complexity of the first iterations will depend on the complexity of the irreducible polynomial. Nevertheless, as a division is required, this solution is less attractive since an additional multiplication will be required. A fully unrolled implementation would need the quite huge amount of $3m^2 + m^2/2$ slices.

A solution based on the Euclidean algorithm seems to be quite expensive. Fortunately, another efficient method based on Fermat's little theorem could be used.

1.7.4 Montgomery Almost Inverse

Another solution would be the use of a Montgomery almost inverse algorithm [Kal95]. With this method, the maximum degree of both R and S linearly increases from the first iteration to the m^{th} iteration. As a result, $m^2/2$ slices can be saved. Unfortunately, an additional multiplication is required to map this almost inverse to the appropriate domain. It can be done with a regular multiplication and “precomputed” powers of 2, or repeated modular shifts. Depending on the selected input and output domains, the repeated modular shift method will require $(m/2)^2$ slices (Montgomery $\rightarrow$ Montgomery) or $(2 \cdot m/2)^2$ slices (Classical $\rightarrow$ Classical). Adding the cost of another multiplication to achieve a division, the overall area complexity of this technique renders it useless.

1.7.5 Fermat’s Little Theorem Inverse

Another method for the computation of a modular inverse, using a modular exponentiation, is based on Fermat’s little theorem. The following relationship holds for any $\beta \neq 0 \in \text{GF}(2^m)$:

$$\beta^{-1} = \beta^{2^m-2} = (\beta^{2^{m-1}-1})^2$$

As the $\text{GF}(2^m)$ fields have a special group order, an interesting improvement is available. In order to minimize the number of multiplications (the squarings are much cheaper), the multiplication chain technique of Itoh and Tsujii [IT88] can be employed. This method is based on the identities:

$$\beta^{2^{m-1}-1} = \begin{cases} \beta^{(2^{\frac{m-1}{2}}-1)(2^{\frac{m-1}{2}}+1)} = (\beta^{2^{\frac{m-1}{2}}-1})^{2^{\frac{m-1}{2}}} \cdot \beta^{2^{\frac{m-1}{2}}-1}, m \text{ odd} \\ \beta \cdot \beta^{2^{m-1}-2} = \beta \cdot (\beta^{2^{m-2}-1})^2, m \text{ even} \end{cases}$$

The number of multiplications required to compute this expression is $\lfloor \log_2(m-1) \rfloor + \text{HW}(m-1) - 1$, where $\text{HW}()$ stands for the hamming weight function. This method has been selected for the implementation of the pipelined point addition circuit.

1.7.6 Montgomery Trick

As inverters are usually more expensive than multipliers, the Montgomery trick [Coh93] could be used. If the throughput of multiplier(s) is increased by a factor $3t - 3$ (*resp.* $4t - 3$), the throughput of an inverter (*resp.* divider) can be reduced by a factor t . More memory is also required to store the different inputs and temporary data.

Given n elements $a_1, a_2, \dots, a_n$, Montgomery trick may be used to compute their inverse $b_1, b_2, \dots, b_n$ in the following way [Coh93]:

1.8 Elliptic Curve Processors

In order to best fit the capabilities of different FPGA families, several kinds of architectures are considered. The aim is to achieve the best area-time complexity on a given platform, resulting in a fair throughput/hardware cost comparison. For instance, a big

Algorithm 1 Montgomery Trick.**Inputs:** $a_1, a_2, \dots, a_n$ **Output:** Inverses $b_1, b_2, \dots, b_n$ of $a_1, a_2, \dots, a_n$

```

 $c_1 \leftarrow a_1$ 
for  $i = 2$  to  $n$  do
     $c_i \leftarrow c_{i-1} \cdot a_i$ 
 $u \leftarrow c_n^{-1}$ 
for  $i = n$  downto  $2$  do
     $b_i \leftarrow c_{i-1} \cdot u$ 
     $u \leftarrow u \cdot a_i$ 
 $b_1 \leftarrow u$ 

```

high-performance device is not reduced to the sum of small devices: the high internal data bandwidth is exploited. For each of the four design principles, the algorithms of the architecture are selected using the theoretical area requirements provided in Section 1.7. The cost of control, RAMs, ROMs and the interface also have to be taken into account.

To ease this analysis, the operating frequency is supposed to be the same for all architectures. For small architectures, the bottleneck is the control logic while the frequency is limited by place and route problems in big architectures.

To avoid data dependency problems within an EC- μ P, several chains can also be used in parallel. This is important to keep arithmetic operators busy (like the critical divider). It is even required when pipelining is introduced within the operators. The drawback is the increase in memory requirements.

1.8.1 Tiny

The first idea is to build a small footprint ALU, able to compute all the $\text{GF}(2^m)$ arithmetic functions. As the behaviour of serial by parallel multiplication and Euclidean division only differs in the shift direction, hybrid circuits are achievable [KKKH03, KYLS04]. The squaring can be translated into a multiplication and the addition made of small additional logic. Beside the ROM storing the points of the u partitions, 3 memory locations are required to store x_3, y_3 and a temporary variable. A small $\text{GF}(p)$ unit for updating coordinates and the chain length is also needed. Unfortunately, considering the area of the whole EC- μ P, theoretical results show that this solution leads to an inefficient area-time product.

1.8.2 Small

Another approach is to allow several small parallel arithmetic units: a Euclidean divider/inverter (Div/Inv), a digit-serial by parallel multiplier (Mult) and a squarer (Sqr) for instance. A dedicated squarer is however not needed as Mult performs faster than Div. Two chains are used to never let the divider idle. This architecture requires therefore

$2m + 1$ cycles to compute 1 PA.

As the latency of the multiplier with $D = 2$ is 4 times smaller than the divider, it could be worth it to use Montgomery trick. However, while processing 1 chain requires 1 Div and 2 Mults, 2 chains require 2 Divs and 4 Mults, or 1 Inv and 9 Mults using Montgomery trick⁵ with $t = 2$ and $D = 5$. Four chains are used with this approach and require $12 (\leq 2^4)$ memory locations: 8 for the 4 points data and 2 for temporary variables. Roughly speaking, $3m/2$ slices can be theoretically saved (without taking into account the increase of the logic needed by the control machine). As a result, Montgomery trick could be interesting.

1.8.3 Medium

If more area is available, it could be interesting to use sub-quadratic multiplication algorithms in order to improve the area-time product. As sketched out in Section 1.7, an inverter based on Itoh-Tsujii with parallel multipliers could perform better than a Euclidean-based divider. For this architecture, one parallel Karatsuba multiplier (Mult) with a multi-squarer (Msqr) is used for the inversion and the two multiplications of the PA algorithm (cf. Fig. 1.6). The inverter architecture and especially the principle of a Msqr unit is similar to [RHMLSCC]. The main difference lies in the purpose of the circuit: area-time efficiency is required here, not only speed. This is achieved by using several chains and pipelining the arithmetic units. The number of cycles needed to compute one PA is $2 + \lfloor \log_2(m - 1) \rfloor + \text{HammingWeight}(m - 1) - 1$ ($= 11$ for $m = 163$).

The number of parallel chains used is equal to 16: the pipeline depth of both the Mult (8) and the Msqr (8). To maximize the utilization of the Msqr, the two multiplications can be computed *during* the inversion. The computational requirements of Msqr can therefore be reduced as two multi-squarings can be both computed in two iterations. This requires doubling the number of chains.

As an inversion instead of a division is computed anyway, Montgomery trick could prove efficient in this case. For $m = 163$, the number of cycles becomes 8 for one PA (25 % improvement). Twice the number of chains is required. The architecture is also more complicated as there are more different operations. However, the computational requirements, and therefore the area, of Msqr can be reduced if the seven multiplications are computed *during* the inversion.

1.8.4 Large

If a large amount of area is available, a fully pipelined design can be implemented in order to reach the best area-time product. The circuit can indeed be data-driven and fully specialized. Moreover, economy of scale can also be achieved. The number of chains is simply equal to the number of register stages in the pipeline, resulting in a PA every clock cycle. In particular, Itoh-Tsujii technique is used in parallel with repeated squarers and parallel Karatsuba multipliers. The reader is referred to [BMQ06] for further information about this strategy.

⁵Throughput of the Mult and the Inv must be equalized to maximize the efficiency.

1.9 Chosen Elliptic Curve Processor

Appropriate algorithms were chosen thanks to the theoretical area requirements of Section 1.7. The *tiny* processor was discarded on that basis. Now, based on preliminary implementation results, area and throughput of each solution are evaluated. Such numbers are reported in Table 1.1 together with prices of corresponding FPGAs. “Preliminary results” means here that all the control structures were not fully implemented. However, performances are based on working components (Mult, Div, Sqr, RAMs, ...) and behaviour correctness was checked on software. Based on those estimates, the *medium* processor on a low-cost Spartan3E appears as the solution with the best throughput over hardware cost (device price). According to the author, those results are sufficient to focus only here on the *medium* architecture. Nevertheless, exact description and full implementation of other processors could be the subject of further work.

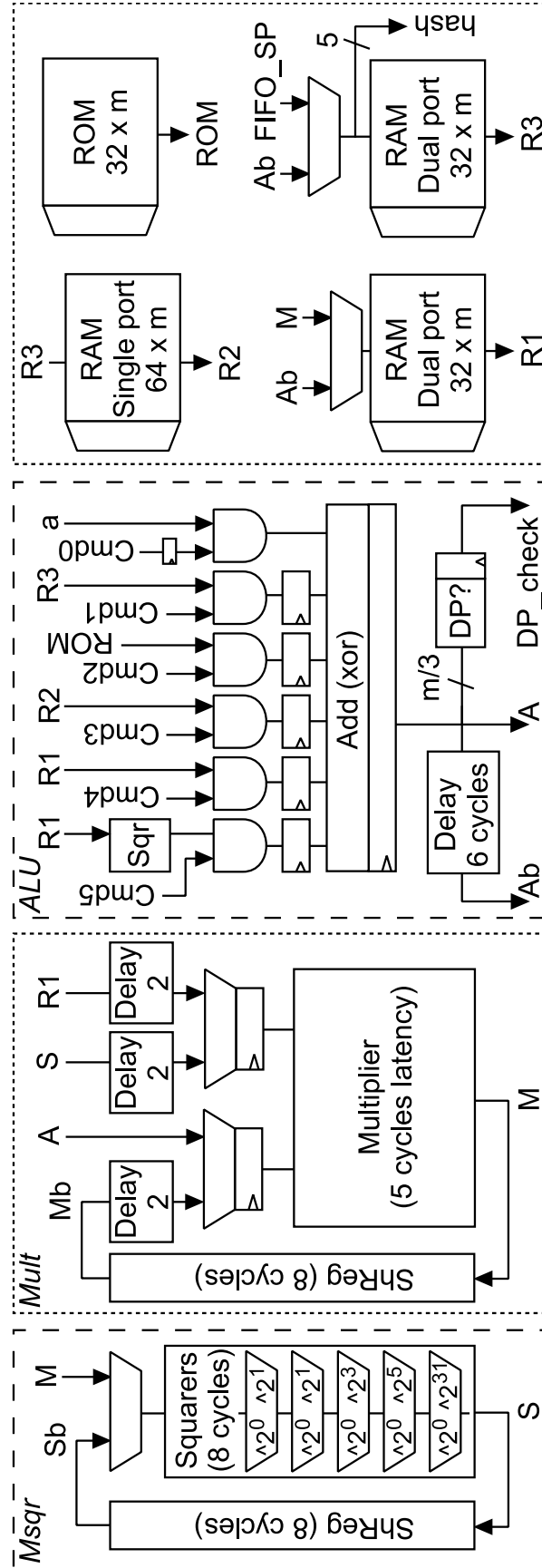
Table 1.1 reports the preliminary results for the different design strategies on a low-cost Spartan3E (XC3S1600E-4, 14 kSlices, 38\$) and a high performance Virtex4 (XC4LX200-10, 89 kSlices, 2070\$) FPGA. Those Xilinx’s prices stand for 1000 devices. For the different extension fields, those results suggest that the most cost-effective choice is a *medium* architecture on low-cost FPGAs. The price of a big Virtex4 is dominated by die and yield costs. As architectures do not really benefit from V4 specific features, it seems inappropriate to solve ECDLP.

	<i>small</i>		<i>medium</i>		<i>large</i>
FPGA	S3E1600	V4LX200	S3E1600	V4LX200	V4LX200
Area $m = 79$	1.2		4.8		22.2
[kSlices] $m = 109$	1.6		7.4		41.3
$m = 163$	2.4		13.8		81.8
Frequency [Mhz]	100	250	100	250	250
Thr./EC- μ P	Freq./ $(2m + 1)$		$9 \cdot 10^6$	$22.7 \cdot 10^6$	$250 \cdot 10^6$
#EC- μ P	5 to 11	36 to 72	1 to 2	6 to 17	1 to 3
Thr./cost $m = 79$	182 (11)	55 (72)	478 (2)	187 (17)	362 (3)
[kPA/s\$] $m = 109$	96 (8)	30 (54)	239 (1)	120 (12)	242 (2)
$m = 163$	40 (5)	13 (36)	239 (1)	66 (6)	120 (1)

Table 1.1: Area and throughput estimations for each processor

1.9.1 The Medium Processor

The *medium* architecture is presented in Fig. 1.6 while the scheduling for $m = 163$ is available as Table 1.2. The main components are a Mult, a Msqr and an ALU. 3 RAMs and a ROM are also available to store coordinates and variables. Shift registers (Delays) are used to render the latency independent from the computed operation. A buffer is used to accumulate the partial result of the inversion in Mult. The buffer of Msqr is employed when a multi squaring is performed in multiple iterations. A comparator is also employed to check the DP criterion.

Figure 1.6: Medium processor, Point Update unit (squarers for $m = 163$)

The scheduling of the PA for $m = 163$ is presented in Table 1.2. The method is similar for other extension degrees. Each PA_i state represents 2 times 8 cycles. As Msqr follows Mult, its computations are delayed by 8 cycles. Write operations happen 8 cycles later as latency of arithmetic units is 8 cycles. Two passes are necessary to compute PAs. One prime on a variable means that data of one set of 16 chains are processed. No prime or two primes means the other set of 16 chains is treated. The role of those sets is exchanged at each pass. The a constant is the curve parameter (cf. Section 1.3).

Based on synthesis results for $m = 163$ on a Spartan3E-1600, it is clearly interesting to perform the 2 multiplications during the inversion: the area saving is 50% (22% of the FPGA) for the Msqr and the area loss is 45% (5% of the FPGA) for the memory. 32 chains are therefore used. The Montgomery trick was currently not fully implemented. Using this strategy (64 chains), the memory increase is 70% (8% of the FPGA) and surprisingly, no significant savings are measured for the Msqr. As a result, if there is enough remaining area in the FPGA, the Montgomery trick approach could be interesting.

The *Coeff Update* and *Chain Length Check* units serially process the data. They are simply made of 4 bRAMs (1 ROM and 2 RAMs for c, d and 1 for chain lengths) and a few logic for 16-bit modular adders, counters and comparators.

1.10 Hardware Results and Cost Assessment

In this section, hardware implementation results and cost assessment for attacking $GF(2^m)$ ECDLP were achieved for 3 security levels recommended in [CRb].

1.10.1 Hardware results

All the hardware modules were written in VHDL and placed & routed on either XC3S1200E-4FT256 (21 US\$) or XC3S1600E-5FG320 (33 US\$) low-cost FPGAs⁶. Table 1.3 reports the different performances for a clock period constrained to 10 ns. For $m = 113$, two cores are embedded in the FPGA. Notice that bRAMs requirements depend on whether enough of such RAMs was available to implement 32 or $64 \times m$ memories. The throughput over FPGA cost is used to allow fair comparison with other FPGA platforms. The power consumption was also estimated using the *Xilinx Spartan-3E Web V8.1* power tool. The electricity cost for one year, assuming a price of 0.1 US\$ per kWh, is also reported.

1.10.2 Cost Assessments

It was planned to compare hardware with homemade software results. Unfortunately, our code based on the C++ NTL library [Sho] has poor performances compared to Certicom challenge results. Chris Monico's challenge results for $GF(2^{109})$ will therefore be used for further comparisons.

For the hardware cost assessment on FPGA, the COPACOBANA engine [KPP⁺06] was chosen. It embeds 120 low-cost Spartan3-1000 FPGAs. The Spartan3E-1600 FPGA

⁶FPGA costs are 2007/2008 Xilinx prices for 1000 devices.

	S (Msqr)	M (Mult)	R1 ($32 \times m$ RAM)	R2 ($64 \times m$ RAM)	R3 ($32 \times m$ RAM)	A, Ab (ALU)
PA ₁	2Sqr(M)	$A \times R1$ $Mb \leftarrow M$	$R1_0 (x_1 + x_2)$	$R2_0 \leftarrow R3$	$R3_0 (x'_3)$	Sqr(R1) $((x_1 + x_2)^2)$
PA ₂	Sqr(M)	$Mb \times S$ $Mb \leftarrow M$	$R1_1 (\sqrt{Inv'})$ $R1_1 \leftarrow Ab$	$R2_1 \leftarrow R3$	$R3_1 (y'_3)$	Sqr(R1) (Inv')
PA ₃	5Sqr(M)	$Mb \times R1$ $Mb \leftarrow M$	$R1_0 (x_1 + x_2)$			
PA ₄	10Sqr(M)	$Mb \times S$ $Mb \leftarrow M$				
PA ₅	10Sqr(M)	$Mb \times S$ $Mb \leftarrow M$				
PA ₆	$Sb \leftarrow S$ 10Sqr(M)	$A \times R1$	$R1_1 (Inv')$ $R1_1 \leftarrow M$	$R2_1 (y'_1)$		Add(R2,ROM) $(y'_1 + y'_2)$
PA ₇	40Sqr(M)	$Mb \times S$ $Mb \leftarrow M$	$R1_1 (\lambda')$	$R2_0 (x'_1)$	$R3_0 \leftarrow Ab$	Add(Sqr(R1), R1,R2,ROM,a)
PA ₈	Sqr(M)	$Mb \times S$ $Mb \leftarrow M$				
PA ₉	40Sqr(M)	$Mb \times R1$ $Mb \leftarrow M$	$R1_0 (x_1 + x_2)$ $R1_0 \leftarrow Ab$		$R3_0 (x'_3)$	Add(R3,ROM) $(x'_3 + x'_2)$
PA ₁₀	$Sb \leftarrow S$ 41Sqr(M)	$A \times R1$	$R1_1 (\lambda')$ $R1_1 \leftarrow M$	$R2_0 (x'_1)$	$R3_0 (x'_3)$	Add(R2,R3) $(x'_1 + x'_3)$
PA ₁₁		$Mb \times S$	$R1_1 (\lambda'(x'_1 + x'_3))$ $R1_1 \leftarrow M$	$R2_1 (y'_1)$	$R3_0 (x'_3)$ $R3_1 \leftarrow Ab$	Add(R1,R2,R3) (y'_3)

Table 1.2: Scheduling of point additions for $m = 163$, without initialization

m	FPGA	Area [kSlices]	Area [bRAMs]	Freq. [Mhz]	Throughput [PA/s]	Thr./cost [PA/s\$]	Cons. [W]	Elec. price [\$/year]
113	S3E1600-5	13.9 (94%)	18 (50%)	100	$2 \times 10^{10^6}$	$6 \cdot 10^5$	4.2	3.7
131	S3E1200-4	7.9 (91%)	21(75%)	100	$10 \cdot 10^6$	$4.8 \cdot 10^5$	3.2	2.8
163	S3E1600-5	10.9 (74%)	25 (69%)	100	$9.1 \cdot 10^6$	$2.7 \cdot 10^5$	3.8	3.3

Table 1.3: Medium EC- μ P, Place and Route results from Xilinx ISE 9.1.

selected for this work has twice the area of a Spartan3-1000 FPGA. Nevertheless, the price of both devices is the same (for 1000 parts) and their form factor is similar. The price of one COPACOBANA engine is therefore assumed to remain 10 kUS\$. Concerning the power consumption, the xilinx web power tool suggests that the power consumption is doubled with the S3E FPGAs. As a result, doubling the power consumption of the original engine seems safe. Based on power consumption needed to attack DES with COPACOBANA (600 Watts), we will assume a power consumption of 1.2 kWatts for our application. Exact power consumption is left for further work.

For the cost assessment reported in Table 1.4, achievable performances are estimated through the expected running time (ERT). It is derived from performances of the architecture and expected number of group operations (ENO) before a collision occurs (cf. 1.5.2). For ENO, n is set as 2^m and θ as $2^{-m/3}$. The expenses for the power consumption were added for the total cost. For an attack during 1 year, it represents a 10^{th} of the purchase cost of the device.

Those results show that $\text{GF}(2^{113})$ is far from secure. For that attack, the number of required FPGA's is sufficiently small to overcome an ASIC-based solution. While the cost for the $\text{GF}(2^{131})$ attack is still tractable, an ASIC-based platform should be more cost-effective. For $m = 163$, a standard level security parameter, it is currently impossible to mount an FPGA-based attack.

Based on those results and [KR07], a rough ASIC extrapolation for a 90 nm CMOS techno using standard cells can be sketched⁷. The following parameters are assumed: area factor is 20, speed factor is 3.5, consumption factor is 14, die size of S3E-1600 is 2.5×2.5 mm and cost of one 300 mm wafer is 30 k\$ (multiplied by two for overheads like packaging, cooling, ...). This rough estimation leads to a cost of $2.2 \cdot 10^9$ \$ neglecting NREs for $m = 163$, which is still currently out of reach. Notice that half the price is for power consumption.

Another goal of this cost assessment is to compare hardware and software-based solutions. To solve the $\text{GF}(2^{109})$ challenge in 6 months, 2400 Athlon XP 3200+ were needed. By modifying the $m = 113$ result of Table 1.4, this problem is expected to be solved by one COPACOBANA engine in 6 months. Assuming a price of 150 \$ for a computer and a consumption of 250 Watts, the purchase price ratio is **35**. This is a bit less than expected, provided those general purpose CPUs have no dedicated $\text{GF}(2^m)$ ALU. From a power consumption prospect, the ratio is as big as **500**. Those results clearly justify the use of hardware.

⁷Assuming the circuits are built to attack a given ECDLP instance many times, meaning that both the field size and the irreducible polynomial can be hardwired.

m	ENO [PA]	Thr. [PA/s]	ERT [s]	ERT [s] on 1 Copa	#Copa for 1-year	Cost [US\$] of power supply	Total cost [US\$] 1-year
113	$128 \cdot 10^{15}$	$20 \cdot 10^6$	$6.4 \cdot 10^9$	$53.3 \cdot 10^6$	2	$2.1 \cdot 10^3$	$22.1 \cdot 10^3$
131	$65.4 \cdot 10^{18}$	$10 \cdot 10^6$	$6.54 \cdot 10^{12}$	$54.5 \cdot 10^9$	1728	$1.8 \cdot 10^6$	$19.1 \cdot 10^6$
163	$4.23 \cdot 10^{24}$	$9.1 \cdot 10^6$	$471 \cdot 10^{15}$	$4 \cdot 10^{15}$	$125 \cdot 10^6$	$131 \cdot 10^9$	$1.4 \cdot 10^{12}$

Table 1.4: Cost assessments to solve different $\text{GF}(2^m)$ ECDLP using COPACOBANA

As a $\text{GF}(2^m)$ attack is supposed to perform better in hardware than a $\text{GF}(p)$ one, it is interesting to compare them. From [GPP07], the throughput for $\text{GF}(p)$ and $k = 160$ (Table 1) is 93.6 kPA/s. Multiplying this result by 2 (to estimate performances on a S3E-1600) and comparing with the $m = 163$ result of Table 1.3, it appears that the throughput ratio is near **50**. For fairness purpose, notice that no parameters are hardwired in their architecture.

1.11 Conclusion

This work presented a thorough analysis of an FPGA-based solution to attack elliptic curve cryptosystems over $\text{GF}(2^m)$. The provided results complete in many aspects the few previous studies. In particular, the performance-cost optimum for four different architectures was investigated on two different families of Xilinx FPGAs. The selected architecture, based on a low-cost Spartan3E FPGA, was described and implemented. Cost assessments were then performed on the basis of a cluster of low-cost FPGAs.

Compared to the software used to solve the $\text{GF}(2^{109})$ challenge, the hardware exhibits an improvement factor of 35 for purchase costs and 500 for power consumption. Considering current architectures to attack $\text{GF}(p)$ and $\text{GF}(2^m)$ instances, a throughput ratio of 50 was also measured. This follows the expectations as $\text{GF}(2^m)$ arithmetic is particularly well suited for hardware platforms.

According to the data points from Lenstra and Verheul’s paper, an ECC attack of one MMY is comparable to [\$70000, 1 day, 1996]-hardware. If we extend this device to this 2006 attack, we find that one MMY is roughly [\$1100, 1 day, 2006]-hardware. In this chapter, the performance achieved corresponds to [\$230000, 1 day, 2006] for the same one MMY. We can therefore emphasize the feeling that their “software-hardware conversion” is indeed pure coincidence. This also emphasize that attack evaluation is something of great importance as it is our best tool to evaluate our own security. This is underlined once more here by reminding that the lowest security level of the SECG standard, based on $\text{GF}(2^{113})$, was found to be *easily* breakable. However, even by taking the rough ASIC extrapolation, a standard security level like $\text{GF}(2^{163})$ stays out of reach of current attacks. Breaking those systems would require either far too much time given a moderate amount of money, or a prohibitive price to raise a solution in a reasonable amount of time. Given today’s know-how and understanding of ECC, it is considered that current standard security keys do not suffer from any threat for the time being.

Bibliography

- [AH08] B. Ansari and M. A. Hasan. High Performance Architecture of Elliptic Curve Scalar Multiplication. *IEEE Transactions on Computers*, 57(11):1443–1453, 2008.
- [AMV93] G. B. Agnew, R. C. Mullin, and S. A. Vanstone. An Implementation of Elliptic Curve Cryptosystems over $F_{2^{155}}$. *IEEE Journal on Selected Areas in Communications*, 11(5):804–813, June 1993.
- [BMQ06] P. Bulens, G. Meurice de Dormale, and J.-J. Quisquater. Hardware for Collision Search on Elliptic Curve over $GF(2^m)$. *Special-purpose Hardware for Attacking Cryptographic Systems – SHARCS’06*, 2006.
- [BSS99] I. F. Blake, G. Seroussi, and N. P. Smart. *Elliptic Curves in Cryptography*. Cambridge University Press, 1999.
- [CMO98] H. Cohen, A. Miyaji, and T. Ono. Efficient Elliptic Curve Exponentiation Using Mixed Coordinates. *Advances in Cryptology — ASIACRYPT’98*, Lecture Notes in Computer Science 1514:51–65, 1998.
- [Coh93] H. Cohen. *A course in computational algebraic number theory*. Springer-Verlag, 1993.
- [CRa] Certicom Research. Certicom. <http://www.certicom.com>.
- [CRb] Certicom Research. SEC 2: Recommended Elliptic Curve Domain Parameters, v1.0. <http://www.secg.org>.
- [Den82] D. E. Denning. *Cryptography and Data Security*. Addison Wesley, 1982.
- [FIPS182] U.S. Department of Commerce/National Institute of Standards and Technology (NIST). Digital Signature Standard (DSS), FIPS PUB 182-2change1.
- [GB] Damien Giry and Philippe Bulens. Cryptographic Key Length Recommendation. <http://www.keylength.com>.
- [GHS02] P. Gaudry, F. Hess, and N. P. Smart. Constructive and Destructive Facets of Weil Descent on Elliptic Curves. *Journal of Cryptology*, 15(1):19–46, 2002.

- [GKB⁺06] K. Gaj, S. Kwon, P. Baier, P. Kohlbrenner, H. Le, M. Khaleeluddin, and R. Bachimanchi. Implementing the Elliptic Curve Method of Factoring in Reconfigurable Hardware. *Special-purpose Hardware for Attacking Cryptographic Systems – SHARCS’06*, 2006.
- [GLV00] R. Gallant, R. Lambert, and S. Vanstone. Improving the Parallelized Pollard Lambda Search on Binary Anomalous Curves. *Mathematics of Computation*, 69:1699–1705, 2000.
- [GPP07] T. Güneysu, C. Paar, and J. Pelzl. Attacking Elliptic Curve Cryptosystems with Special-Purpose Hardware. *International symposium on Field programmable gate arrays — FPGA’07*, pages 207–215, 2007.
- [Gut02] A. A.-A. Gutub. New Hardware Algorithms and Designs for Montgomery Modular Inverse Computation in Galois Fields $GF(p)$ and $GF(2^n)$. *Ph.D. Thesis*, 2002.
- [HMOV04] D. Hankerson, A. Menezes, and S. Vanstone. *Guide to Elliptic Curve Cryptography*. Springer, 2004.
- [IT88] T. Itoh and S. Tsujii. A Fast Algorithm for Computing Multiplicative Inverses in $GF(2^m)$ Using Normal Bases. *Information and Computation*, 78:171–177, 1988.
- [Kal95] B. S. Kaliski Jr. The Montgomery Inverse and its Applications. *IEEE Transactions on Computers*, 44(8):1064–1065, 1995.
- [KKKH03] C. H. Kim, S. Kwon, J. J. Kim, and C. P. Hong. A New Arithmetic Unit in $GF(2^m)$ for Reconfigurable Hardware Implementation. *International conference on Field Programmable Logic — FPL’03*, LNCS 2778:670–680, 2003.
- [KO86] A. Karatsuba and Y. Ofman. Multiplication of Multidigit Numbers on Automata. *Soviet Physics Doklady*, 7(7):595–596, 1986.
- [Kob87] N. Koblitz. Elliptic curve cryptosystems. *Mathematics of computation*, 48:203–209, 1987.
- [Kob91] N. Koblitz. CM curves with good cryptographic properties. *Advances in Cryptology — CRYPTO’91*, LNCS 576:279–287, 1991.
- [KPP⁺06] S. Kumar, C. Paar, J. Pelzl, G. Pfeiffer, and M. Schimmler. Breaking Ciphers with COPACOBANA – A Cost-Optimized Parallel Code Breaker. *Cryptographic Hardware and Embedded Systems — CHES’06*, LNCS 4249:101–118, 2006.
- [KR07] I. Kuon and J. Rose. Measuring the Gap Between FPGAs and ASICs. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 62(2), February 2007.

- [KYLS04] M. G. Kim, S. J. Yu, Y. S. Lee, and J. S. Song. A Fast Hybrid Arithmetic Unit for Elliptic Curve Cryptosystem in Galois Fields with Prime and Composite Exponents. *IEICE Electronics Express*, 1(1):13–18, 2004.
- [Men93] A. Menezes. *Elliptic Curve Public Key Cryptosystems*. Kluwer, 1993.
- [Mil86] V. Miller. Uses of Elliptic Curves in Cryptography. *Advances in Cryptology — CRYPTO’85*, LNCS 218:417–426, 1986.
- [MOV91] A. Menezes, T. Okamoto, and S. A. Vanstone. Reducing Elliptic Curve Logarithms to Logarithms in a Finite Field. *ACM Symposium on Theory Computing*, pages 80–89, 1991.
- [MQ06] G. Meurice de Dormale and J.-J. Quisquater. Iterative Modular Division over $\text{GF}(2^m)$: Novel Algorithm and Implementations on FPGA. *International workshop on Applied Reconfigurable Computing — ARC’06*, LNCS 3985:370–382, 2006.
- [MTW04] A. Menezes, E. Teske, and A. Weng. Weak fields for ECC. *Topics in Cryptology — CT-RSA’04*, LNCS 2964:366–386, 2004.
- [Nal] Nallatech. <http://www.nallatech.com>.
- [Pel06] J. Pelzl. Exact Cost Estimates for Attacks on ECC with Special-Purpose Hardware. *Workshop on Elliptic Curve Cryptography — ECC’06*, 2006.
- [Pol78] J. M. Pollard. Monte Carlo Methods for Index computation (mod p). *Mathematics of computation*, 32(143):918–924, July 1978.
- [Pol00] J. M. Pollard. Kangaroos, Monopoly and Discrete Logarithms. *Journal of Cryptology*, 13:437–447, 2000.
- [RHK03] F. Rodríguez-Henríquez and Ç. K. Koç. On Fully Parallel Karatsuba Multipliers for $\text{GF}(2^m)$. *Computer Science and Technology*, 394, 2003.
- [RHMLSCC] F. Rodríguez-Henríquez, G. Morales-Luna, N.A. Saqib, and N. Cruz-Cortes. Parallel Itoh-Tsujii Multiplicative Inversion Algorithm for a Special Class of Trinomials. <http://eprint.iacr.org/2006/035>.
- [RHSCC05] F. Rodríguez-Henríquez, N.A. Saqib, and N. Cruz-Cortes. A Fast Implementation of Multiplicative Inversion Over $\text{GF}(2^m)$. *Information Technology: Coding and Computing – ITCC’05*, IEEE Computer Society:574–579, 2005.
- [Sho] Victor Shoup. NTL: A Library for doing Number Theory. <http://www.shoup.net/>.
- [Sma00] N. P. Smart. A note on the x-coordinate of points on an elliptic curve in characteristic two. *Technical Report CSTR-00-019*, 2000.

- [Sol01] J. A. Solinas. Low-Weight Binary Representations for Pairs of Integers. *CACR Combinatorics and Optimization Research Report 2001-41*, 2001.
- [SP98] L. Song and K. K. Parhi. Low energy digit-serial/parallel finite field multipliers. *Journal of VLSI Signal Processing*, 19(2):149–166, 1998.
- [Sta03] F.-X. Standaert. PCI to user interface core for BenNUEY. *UCL Crypto Group Technical Report Series*, 2003.
- [Ste67] J. Stein. Computational problems associated with Racah algebra. *Journal of Computational Physics*, 1:397–405, 1967.
- [SWIG] SWIG. SWIG: Simplified Wrapper and Interface Generator. <http://www.swig.org/>.
- [Tes98] E. Teske. Speeding Up Pollard’s Rho Method for Computing Discrete Logarithms. *Algorithmic Number Theory Symposium III*, LNCS 1423:541–554, 1998.
- [Tes00] E. Teske. On Random Walks for Pollard’s Rho Method. *Mathematics of computation*, 70(234):809–825, 2000.
- [Tes01] E. Teske. Computing Discrete Logarithms with the Parallelized Kangaroo Method. *CACR Combinatorics and Optimization Research Report 2001-01*, 2001.
- [vOW94] P. C. van Oorschot and M. J. Wiener. Parallel Collision Search with Application to Hash Functions and Discrete Logarithms. *2nd ACM Conference on Computer and Communications Security*, pages 210–218, 1994.
- [vOW99] P. C. van Oorschot and M. J. Wiener. Parallel Collision Search with Cryptanalytic Applications. *Journal of Cryptology*, 12:1–28, 1999.
- [vzGS06] J. von zur Gathen and J. Shokrollahi. Fast Arithmetic for Polynomials over $\mathbb{F}_2$ in Hardware. *IEEE Information Theory Workshop — ITW’06*, pages 107–111, 2006.
- [Wu99] H. Wu. Low Complexity Bit-Parallel Finite Field Arithmetic Using Polynomial Basis. *Cryptographic Hardware and Embedded Systems – CHES’99*, LNCS 1717:280–291, 1999.
- [Wu02] H. Wu. Montgomery Multiplier and Squarer for a Class of Finite Fields. *IEEE Transaction on Computers*, 51(5):521–529, 2002.
- [WWSH04] C.-H. Wu, C.-M. Wu, M.-D. Shieh, and Y.-T. Hwang. High-Speed, Low-Complexity Systolic Designs of Novel Iterative Division Algorithms in $\text{GF}(2^m)$. *IEEE Transaction on Computers*, 53(3):375–380, 2004.

-
- [WZ99] M. J. Wiener and R. Zuccherato. Faster Attacks on Elliptic Curve Cryptosystems. *Selected Areas of Cryptography — SAC'99*, LNCS 1556:190–200, 1999.

Physical Cryptanalysis Example: Fault Attacks on DLP-based Schemes

Although not included in Lenstra and Verheuls model, physical attacks are usually considered as a powerful threat. Indeed, many cryptosystems have been suffering from fault attacks when implemented in physical devices such as smart cards. The original purpose of fault attacks was to induce errors during computations involving a secret key or change the value of the secret key. Recently another approach appeared, i.e., corruption of one of the public elements of public key cryptosystems. Protocols relying on ECDLP (Elliptic Curve Discrete Logarithm Problem) and IFP (Integer Factorization Problem) have already been shown to be vulnerable to such attacks. Consequently, it was worth wondering whether the approach could be applied to other schemes like DLP (Discrete Logarithm Problem) based schemes. In this chapter, we present the first attacks on DSA and ElGamal signature schemes by a corruption of only public key elements. Our attacks totally break these systems in the sense that they allow recovering the whole secret element.

2.1 Introduction

From a *classical* point of view, cryptanalysis is an abstract mathematical object. However, in practice, algorithms have to be implemented on real physical devices that are exposed to side-channel attacks, like timing attacks [DKL⁺98], power attacks [KJJ99], electromagnetic attacks [AARR02] as well as fault attacks [BDL97].

After the discovery of fault attacks (1970's), various mechanisms for fault production and propagation have been discovered and researched. Some of the most popular fault injection techniques include variations in supply voltage, clock frequency, temperature or the use of white light, X-ray or ion beams [BECN⁺04].

It is not sufficient to just induce any fault in the cryptographic device during a calculation involving a secret. Depending on the algorithm, it is crucial to generate the *right* fault at the *right* time and on the *right* place within the chip. The generated fault must be exploitable, meaning that the attacker has to be able to extract some information about the secret from the erroneous result of the algorithm. Indeed, the first attack [BDL97] that used a fault to derive secret information from a cryptosystem targeted a Chinese Remain-

der Theorem (CRT) based RSA implementation. After that, DES, RSA and ElGamal, ECC, AES and DSA have also been attacked by faults.

All these attacks targeted secret keys and computations involving secret keys. Recently, several papers considered fault attacks on public key elements. These papers demonstrate the necessity to protect public keys against fault attacks at least for ECDLP (Elliptic Curve Discrete Logarithm Problem) [BMM00] and IFP (Integer Factorization Problem) based algorithms [BCMCC06, Mui05, Sei05]. In this chapter, we consider two DLP (Discrete Logarithm Problem) based algorithms, namely the ElGamal and DSA signature schemes, and present the first attacks by corruption of only public key elements. Our attacks totally break these systems in the sense that they allow recovering the whole secret element.

Fault attack model We assume that the attacker is able to enforce random register faults or disturb the transfer of the public keys from EEPROM to RAM, therefore resulting in a random transient change of public keys. Therefore we can think of *words* modified by faults. A single bit flip is not so easy to make in practice whereas changing a word in an undetermined way is the most simple fault to induce.

Previous Attacks on ECDLP Consider a smart card that has to compute $d \cdot P$, for d a scalar and P a point on the curve E . If a fault is induced on P , it is changed into point P^* of curve

$$E^* \equiv y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6^*$$

The attack exploits the fact that the parameter a_6 is not used in the usual point addition formula on Weierstrass elliptic curves. As a consequence, the whole computation is performed on the curve E^* , and $d \cdot P^*$ is obtained. If now E^* is an insecure curve (typically because $\text{ord}(E^*)$ has a small factor), the discrete logarithm problem can be solved, which gives information about d . Faults occurring during the computation of $d \cdot P$ are also exploitable. We refer to [BMM00] for more details.

Previous Attacks on IFP In Seifert's RSA attack [Sei05], the attacker tries to turn RSA public key N into N^* during signature verification, where N^* is prime. He can then simply compute the private exponent, d^* , as $e^{-1} \bmod (N^* - 1)$, assuming e is relatively prime to $(N^* - 1)$. In the off-line part, he can choose bits of N to modify to create N^* . In the on-line part, he repeatedly queries the device with a specially constructed message-signature pair and causes data faults until his particular N^* is used as the modulus in the signature verification algorithm. After that, Muir [Mui05] showed that we do not need to limit ourselves to prime modulus, N^* ; what we are really after are modulus that can be easily factored.

Recently Brier et al. showed another approach [BCMCC06]. The attacker tries to corrupt RSA public key N into N^* similarly to Seifert's attack, but here he tries to attack signature generation process, $m^d \bmod N$. Finally, he can get the secret key, d . The basic idea is based on the fact that for a relatively small value of r – say from 15 to 20 digits –, the discrete logarithm can be computed efficiently by square root methods such as *baby-step giant-step* or *Pollard's rho*. The attacker gathers some faulty pairs (m_i, s_i) ,

corresponding to unknown moduli $N_i^* \neq N$, related by $s_i = m_i^d \bmod N_i^*$. From this data, the private exponent d is retrieved off-line, by progressively determining $d \bmod r_j$, for some small prime powers r_j . When the product $R = \prod_j r_j$ exceeds the modulus N (and so unknown $\varphi(N)$), d can be recovered by means of the Chinese Remainder Theorem.

The rest of this chapter is organized as follows. Sections 2.2 and 2.3 respectively present our new attacks on the DSA and ElGamal signature schemes assuming additionally that the attacker knows the values of the faulty numbers he has induced on the device. Section 2.4 deals with removing this hypothesis: three techniques proposed by [BCMCC06] for RSA are analyzed for DSA and ElGamal. Section 2.5 concludes and discusses further work.

2.2 A New Fault Attack on DSA

2.2.1 The Digital Signature Algorithm

The system parameters for DSA [FIPS182] are $\{p, q, g\}$, where p is prime (at least 512 bits), q is a 160-bit prime dividing $(p - 1)$ and $g \in \mathbb{Z}_p^*$ has order q . The private key is an integer $x \in \mathbb{Z}_q^*$ and the public key is $y = g^x \bmod p$.

Signature

To sign a message m , the signer picks a random $k < q$ and computes:

$$u \leftarrow (g^k \bmod p) \bmod q \quad \text{and} \quad v \leftarrow \frac{h(m) + xu}{k} \bmod q.$$

The signature of m is the pair (u, v) .

Verification

To check (u, v) the verifier ascertains that:

$$u = (g^{wh(m)} y^{wu} \bmod p) \bmod q, \quad \text{where } w = v^{-1} \bmod q.$$

2.2.2 A new attack by a fault induction on p and q

Let us suppose that an attacker succeeded in invoking transient faults before starting signature generation and changing the values of p, q into p^*, q^* , respectively. Assuming the generated k satisfies $\gcd(k, q^*) = 1$, he gets:

$$u^* \leftarrow (g^k \bmod p^*) \bmod q^* \quad \text{and} \quad v^* \leftarrow \frac{h(m) + xu^*}{k} \bmod q^*.$$

Consider any integer t dividing p^* and q^* , such that $\varphi(t)$ divides q^* and g is not a root of zero modulo t . Then, we have

$$(u^*)^{v^*} \equiv g^{h(m) + xu^*} \bmod t.$$

Consequently,

$$\frac{(u^*)^{v^*}}{g^{h(m)}} \equiv (g^{u^*})^x \pmod{t}. \quad (2.1)$$

Denote $DL(\alpha, \beta, t)$ the discrete logarithm of $\beta \pmod{t}$ with respect to α . For any r_j dividing the multiplicative order of (g^{u^*}) modulo t we have

$$x \equiv DL(\alpha, \beta, t) \pmod{r_j},$$

where $\beta = \frac{(u^*)^{v^*}}{g^{h(m)}} \pmod{t}$ and $\alpha = g^{u^*} \pmod{t}$ can be computed from the faulty signature.

Assuming the attacker knows the value of the faulty (p^*, q^*) induced, this suggests an algorithm to recover the secret key x , by successively recovering x modulo r_j with various faulty signatures until the lowest common multiple of those r_j is large enough to recover x by use of Chinese Remainder Theorem.

Toy example Take $p = 124540019$, $q = 17389$, $g = 10083255$. Let the private key be $x = 12496$, so $y = g^x \pmod{p}$. The message to be signed verifies $h(m) = 5246$, and $k = 9557$ are chosen. The signature is the pair $(u = 34, v = 13049)$. Now suppose that the attacker succeeded in making faults and changing the value of p into p^* and q into q^* before generating the signature. Then, we can find x with the values shown in Table 2.1 .

p^*	q^*	k	u^*	v^*	t	α	β	r_j	$x \pmod{r_j}$	$\text{lcm}(\{r_j\})$
124539997	17030	41	13384	9140	131	33	28	65	16	65
124539973	17110	6171	12501	422	59	47	7	58	26	3770
124539983	16536	9961	11694	1214	53	20	36	26	16	3770
124539989	17296	491	6434	202	47	24	18	23	7	86710

Table 2.1: Toy example of the attack on DSA

2.2.3 Analysis of the attack

Our attack generates faulty p^* and q^* , and possibly retrieves information from the ones for which there exists t such that $t \mid q^* \wedge \varphi(t) \mid q^* \wedge t \mid p^*$. We want to evaluate the number of faults needed to recover the whole secret x .

First of all, depending on how the algorithm is implemented, the device may fail to compute the faulty signature if $\gcd(k, q^*) \neq 1$. For a given q^* , the probability that $\gcd(k, q^*) = 1$ is $\varphi(q^*)/q^*$. If q^* is uniformly distributed in $[0, 2^n - 1]$, the condition $\gcd(k, q^*) = 1$ is satisfied with probability $\frac{1}{2^n} \sum_{q=1}^{2^n} \frac{\varphi(q)}{q} = \frac{6}{\pi^2} + \mathcal{O}(\frac{n}{2^n}) \approx 0.6$. Next, we consider the probability that for a given t , the condition $t \mid q^* \wedge \varphi(t) \mid q^* \wedge t \mid p^*$ holds. By Chinese Remainder Theorem, the first two parts simultaneously occur with probability $\frac{1}{\text{lcm}(t, \varphi(t))}$, and by independence of the faults on p and q , the whole condition holds with probability $P_t = \frac{1}{t \cdot \text{lcm}(t, \varphi(t))}$.

Now suppose that $\gcd(k, q^*) = 1$ and $t \mid q^* \wedge \varphi(t) \mid q^* \wedge t \mid p^*$. We want to know the distribution of $r_{\alpha,t}$, the multiplicative order of $\alpha^* = g^{u^*} \pmod{t}$, because equation (2.1) leaks more or less information depending on it. As p^* is uniformly random, the remainder of $u = g^k \pmod{p^*}$ modulo $\varphi(t)$ is nearly uniformly distributed in $\mathbb{Z}_{\varphi(t)}$. We deduce that α is nearly uniformly distributed in the subgroup of $\mathbb{Z}_t^*$ generated by g .

Write $r_{g,t} = \prod_{i=1}^I p_i^{e_i}$ for the multiplicative order of g modulo t , where p_i are distinct primes. Then, for any divisor $r = \prod_{i=1}^I p_i^{e'_i}$ of $r_{g,t}$, the probability that $r_{\alpha,t} = r$ is $\frac{\varphi(r)}{r_{g,t}} = \prod p_i^{-e_i} \prod_{e'_i \neq 0} (p_i - 1) p_i^{e'_i - 1}$, and the probability that r divides $r_{\alpha,t}$ is $\prod_{e'_i \neq 0} p_i^{-(e_i - e'_i + 1)} (p_i^{e_i - e'_i + 1} - 1)$. We will write $\Pr[r \mid t, g]$ for this last probability. In particular, if $r = p_r^{e'_r}$, then $\frac{1}{2} \leq \Pr[r \mid t, g] = p_r^{-(e_r - e'_r + 1)} (p_r^{e_r - e'_r + 1} - 1) \leq 1$.

For any set of integers T , write P_T for the probability that all $t \in T$ verify condition $t \mid q^* \wedge \varphi(t) \mid q^* \wedge t \mid p^*$, and no other integer. Write $\Pr[r \mid g]$ for the probability to recover the secret modulo $r = p_r^{e'_r}$ if the attack is performed once. Then,

$$\Pr[r \mid g] \approx 0.6 \sum_{T=\{t_i\}} \left(1 - \prod_{t_i \in T} (1 - \Pr[r \mid t_i, g]) \right) P_T.$$

By its expression above, $\Pr[r \mid t_i, g]$ is usually close to 1, and always larger than $1/2$, so we can approximate

$$\Pr[r \mid g] \approx 0.6 \sum_{\substack{T=\{t_i\} \\ \exists t \in T \text{ s.t. } r \mid r_{g,t}}} P_T \approx 0.6 \sum_{\substack{T=\{t_i\} \\ \exists t \in T \text{ s.t. } r \mid \varphi(t)}} P_T. \quad (2.2)$$

The last approximation is valid with a large probability, by arguments identical to those used to evaluate $\Pr[r \mid t_i, g]$. Now we will use the following lower bound:

$$\sum_{\substack{T=\{t_i\} \\ \exists t \in T \text{ s.t. } r \mid \varphi(t)}} P_T > \sum_{\substack{T=\{t_i\} \\ t_r \in T}} P_T = P_{t_r} \quad \text{where } t_r = \arg \max_{t, r \mid \varphi(t)} P_t. \quad (2.3)$$

We point out that a closer look on it shows that this bound is not too rough but by a small factor. Equations (2.2) and (2.3) together lead to a lower bound on $\Pr[r \mid g]$.

$$\Pr[r \mid g] > \frac{0.6}{t_r \cdot \text{lcm}(\varphi(t_r), t_r)} \quad \text{where } t_r = \arg \max_{t, r \mid \varphi(t)} P_t. \quad (2.4)$$

Predictions Figure 2.1 shows the value of bound (2.4) for all r_j smaller than $5 \cdot 10^4$. In the table of Figure 2.2, we computed $\log_2(\text{lcm}(\{r_j\}))$ for all r_j which have probability larger than $b_s = 10^{-s}$ for $4 \leq s \leq 10$. An approximation for the number of faults needed is roughly given by b_s^{-1} . The last row in the table corresponds to 160 security bits and predicts that about 80 million faults are needed to recover the secret. The table and the figure additionally show that, at least for this range of parameters, the attack complexity is a bit more than cubic in the size of the secret.

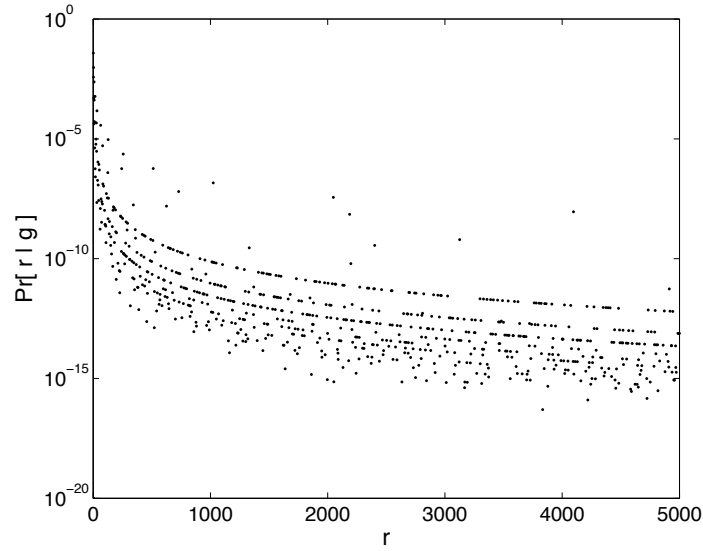


Figure 2.1: Bound (2.4) for all r power of prime smaller than 5000.

b_s^{-1}	$\log_2(\text{lcm}(\{r_j\}))$
10^4	10
10^5	19
10^6	43
10^7	90
10^8	182
10^9	323
10^{10}	589
$8.3 \cdot 10^7$	160

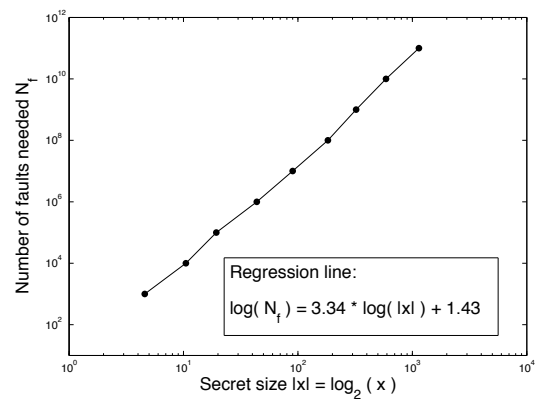


Figure 2.2: $\log_2(\text{lcm}(\{r_j\}))$ for all r_j which have probability larger than $b_s = 10^{-s}$ for $4 \leq s \leq 10$.

2.2.4 Experimental results

In order to check the soundness of our theoretical analysis, some practical experiments were run by software. For each of these, p is a 1024-bit prime, q is a 160-bit prime and g is about the size of p . The results are shown in table 2.2, where each column gives the amount of faulty couples (p^*, q^*) used to retrieve m bits of the private key x . As can be seen, the required amount of faults is below the $8.3 \cdot 10^7$ theoretical bound.

$m = 32$	$m = 64$	$m = 96$	$m = 128$	$m = 160$
811215	8898520	14945495	22174790	34861119
721742	8762969	20601681	38398403	73933924
282811	8363054	14303797	26594960	39066576
284706	3336328	5577267	11579152	20960118
463986	5676441	8549083	26476140	38621903
453623	3174393	10220088	17145832	22623221
798230	6438425	21496166	31049856	61711260

Table 2.2: Experimental software results.

2.3 A New Fault Attack on ElGamal Signature Algorithm

2.3.1 ElGamal Signature Algorithm

In the ElGamal signature scheme [ElG85], to generate a private and public key pair, we first choose a prime p , and two random numbers, g and x , such that both g and x are less than p . The private key is x and the public key is $(y = g^x \bmod p, g, p)$.

Signature

To generate a signature on a message m , the signer first picks a random k such that k is relatively prime to $(p - 1)$. Then, he computes

$$u \equiv g^k \bmod p \quad \text{and} \quad v \equiv \frac{m - xu}{k} \bmod (p - 1).$$

Verification

The signature is the pair (u, v) and is verified by checking that

$$y^u u^v \equiv g^m \bmod p.$$

2.3.2 A new attack by fault injection on p^*

Suppose that there are faults before the signature generation and the modulus p is changed into p^* . Then, we have

$$u^* \equiv g^k \pmod{p^*} \quad \text{and} \quad v^* \equiv \frac{m - xu^*}{k} \pmod{p^* - 1}.$$

Let t divide p^* and $\varphi(t)$ divide $(p^* - 1)$. Then, we have

$$(u^*)^{v^*} \equiv g^{(m-xu^*)} \pmod{t}.$$

Consequently,

$$\frac{(u^*)^{v^*}}{g^m} \equiv (g^{-u^*})^x \pmod{t}.$$

For any r_j dividing the multiplicative order of $(g^{-u^*}) \pmod{t}$, we have

$$x \equiv \text{DL}(\alpha, \beta, t) \pmod{r_j}, \tag{2.5}$$

where $\beta = \frac{(u^*)^{v^*}}{g^m} \pmod{t}$ and $\alpha = g^{-u^*} \pmod{t}$ can be computed from the faulty signature.

Assuming that the attacker knows the value of the faulty p^* induced, equation (2.5) allows recovering some information about x . By retrieving $x \pmod{r_j}$ for sufficiently many r_j satisfying $\text{lcm}(\{r_j\}) > x$, the whole secret can be recovered by use of Chinese Remainder Theorem.

2.3.3 Analysis and Experimental Results

The analysis of ElGamal case is similar to the DSA case. Therefore we omit the details. In the table of Figure 2.3, we computed $\log_2(\text{lcm}(\{r_j\}))$ for all r_j which have probability larger than $b_s = 10^{-s}$ for $4 \leq s \leq 10$. The last row in the table corresponds to 1024 security bits and predict that about three million faults are needed to recover the secret. The table and the figure additionally show that, at least for this range of parameters, the attack complexity is a bit more than quadratic in the size of the secret.

Table 2.3 exhibits the amount of faulty primes generated in order to recover m bits of the private key x in various simulations. In these experiments, each faulty p^* was factorized up to a bound that is indicated in the first column of the table (our predictions considered a bound of $5 \cdot 10^4$). As can be seen, the practical results match pretty well the theoretical expectations. Moreover, the table shows that considering more factors of p^* slightly decreases the number of faulty primes required to break the system.

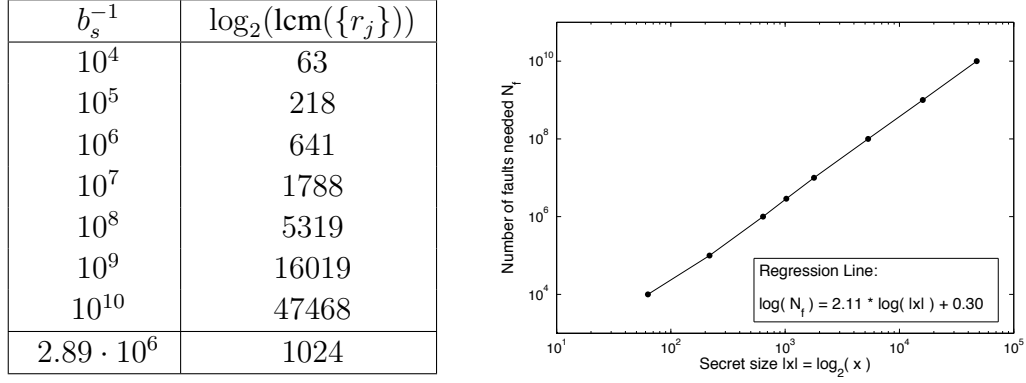


Figure 2.3: $\log_2(\text{lcm}(\{r_j\}))$ for all r_j which have probability larger than $b_s = 10^{-s}$ for $4 \leq s \leq 10$.

Bound on t	$m = 64$	$m = 128$	$m = 256$	$m = 512$	$m = 1024$
10^4	4646	29644	135298	637468	3394102
10^4	5794	39790	104258	786038	3366136
10^4	7420	23290	143946	969046	3613162
$2 \cdot 10^4$	4902	24054	126230	611494	2893428
$2 \cdot 10^4$	4646	29644	135298	635890	2413394
$2 \cdot 10^4$	3756	24822	153722	757148	3138244
$5 \cdot 10^4$	4902	24054	126230	571170	2688882
$5 \cdot 10^4$	7420	23290	142810	696046	3011938
$5 \cdot 10^4$	8810	21742	102974	572170	2931512
10^5	3078	43274	135194	636864	2808468
10^5	7420	25546	142810	649606	2909970
10^5	6062	25352	105240	653434	2623514

Table 2.3: Experimental software results.

2.4 Open Discussion about Recovering *Hits*

So far, either for DSA and ElGamal we assumed that the attacker knew the faulty values he induced. In [BCMCC06], Brier et al. described three modes of their attack on RSA that deals with removing this hypothesis. The first mode supposed absolutely no knowledge about a *Dictionary* while the two others assumed one and made use of it. Here we would like to discuss whether all these modes are applicable in the ElGamal and DSA cases.

2.4.1 Recovering the secret without Dictionary (“blind” method)

As we have seen above, if the condition $t \mid q^* \wedge \varphi(t) \mid q^* \wedge t \mid p^*$ is verified for (p^*, q^*) , we get some information on the secret (namely, we get x modulo a divisor of $\varphi(t)$). Now suppose the condition is not verified, for example, suppose that $t \mid q^* \wedge t \mid p^*$ but $\varphi(t) \nmid q^*$. In this case, β and α reduce to

$$\beta = g^{k \left[\frac{h(m)+xu^*}{k} \bmod q^* \right] - h(m)} \bmod t \quad \text{and} \quad \alpha = g^{u^*} \bmod t.$$

Since $\varphi(t) \nmid q^*$, we admit that $k \left[\frac{h(m)+xu^*}{k} \bmod q^* \right] - h(m)$ is roughly uniformly distributed modulo $\varphi(t)$. We point out that, unlike in the RSA case, the discrete logarithm of β with respect to α may not exist. In this case, the attacker knows that the condition $t \mid q^* \wedge \varphi(t) \mid q^* \wedge t \mid p^*$ was not verified and simply discards the fault. Otherwise, we admit that all values for the discrete logarithm are as likely. Similar analysis shows that both cases still occur if any combination of the three conditions $t \mid q^*$, $\varphi(t) \mid q^*$ and $t \mid p^*$ are verified, except when all three hold. Consequently, a bias-based attack may be built for DSA. We remark however, that due to the small probability for condition $t \mid q^* \wedge \varphi(t) \mid q^* \wedge t \mid p^*$ to occur, the bias will be smaller than for RSA, and consequently the attacker will need more faults. A similar analysis leads to the same conclusion for ElGamal.

2.4.2 Recovering the secret with a Dictionary, Collision method

The second mode in [BCMCC06] makes use of a model of faults, namely, a *Dictionary* of possible faulty values. For our attack on DSA, this dictionary $\mathcal{D}$ will of course be two-dimensional. In [BCMCC06], the authors use the notion of *marker of an element of the dictionary*, which for DSA should be defined as couples (t, r) , where $t \mid q^* \wedge \varphi(t) \mid q^* \wedge t \mid p^*$ and r is a *not too small* factor of $\varphi(t)$. For every fault $(p^*, q^*) \in \mathcal{D}$ and every marker (t, r) , the attacker keeps the discrete logarithm $DL(\alpha, \beta, t, r)$. As soon as two identical values are found, this common value is believed to be the true value, that is $x \bmod r$. In order to avoid false positives, $\sqrt{r}$ should be much larger than $|\mathcal{D}|$ [BCMCC06].

Now we upper bound the probability that a word of $\mathcal{D}$ has a marker. This probability is smaller than $\sum_{t=|\mathcal{D}|^2}^{\infty} P_t$ and we have

$$\sum_{t=|\mathcal{D}|^2}^{\infty} P_t = \sum_{t=|\mathcal{D}|^2}^{\infty} \frac{1}{t \cdot \text{lcm}(t, \varphi(t))} \leq \sum_{t=|\mathcal{D}|^2}^{\infty} \frac{1}{t^2} = \frac{1}{|\mathcal{D}|^2}.$$

This shows that this mode can not be applied in the context of our attack on DSA, simply because no element of the dictionary will have an appropriate marker. The same conclusion holds for ElGamal.

2.4.3 Recovering the secret with a Dictionary, Optimal method

A third mode is proposed in [BCMCC06] to “optimally recover hits”. This mode uses the whole information available from faults, in the sense that it keeps all possible values for the sequence of faults $((p_1^*, q_1^*), (p_2^*, q_2^*), (p_3^*, q_3^*), \dots)$ that are coherent with the faulty signatures $((u_1, v_1), (u_2, v_2), (u_3, v_3), \dots)$. Moreover it is “optimal” in the sense that it associates a selectivity parameter related to a “maximum-likelihood selection” to each possible sequence of faults.

In theory, this method can be applied to our attacks on DSA and ElGamal. However, handling the lists of fault values will most likely become intractable in practice. This limitation was already present in the RSA case but is still enforced for DSA and ElGamal because much more faults are needed.

2.4.4 Using a Dictionary for ElGamal and DSA, Further Methods

From the previous paragraphs, it seems that there is no way to exploit the additional information given by the dictionary in order to improve our attacks on DSA and ElGamal. This may not be totally true. The Collision method and the Optimal method presented in [BCMCC06] are respectively impossible and most likely impractical, but other methods may exist. For example, we could relax the condition $\sqrt{r} \gg |\mathcal{D}|$ to allow some false positives in the collision method (and discriminate between false and true positives via biases as in the blind method). In our attack on DSA, we could also think about having a dictionary for p^* only. These hybrid methods are not explored here, but left for further work.

2.5 Conclusions and Future works

There have recently been researches on fault attacks by modifying not secret but public keys. As a result, it is sure that we should protect public keys against fault attacks in ECDLP and IFP based schemes. In this chapter, we have shown that fault attacks on two DLP based schemes, DSA and ElGamal signature schemes, are possible by inducing faults on the public key. Both attacks have been carefully analyzed in order to derive their probabilities of success. In each case, the analysis has been validated by software experiments.

Although our analysis and experiments supposed an attacker able to know which faulty values he generated, we showed in section 2.4 that this hypothesis could be removed, for example by the blind method. We expect other methods to be developed in the future to reduce the number of faulty signatures required, in particular when the existence of a dictionary provides additional information.

We have already pointed out that the attacks presented here may not seem really frightening compared to attacks like in [PQ03] where only 2 ciphertexts are required to recover the key used during AES encryption. However, taking those attacks into account is definitely an important concern when designing security systems. Protecting against them is a matter of implementation for which stretching the key is hardly the right solution. This confirms and emphasizes one aspect of Lenstra and Verheul's work: the choice of the key length should draw the attention only once the overall security of the design is convincing.

Bibliography

- [AARR02] D. Agrawal, B. Archambeault, J.R. Rao, and P. Rohatgi. The EM Side-Channel(s). *Workshop on Cryptographic Hardware and Embedded Systems — CHES’02*, Lecture Notes in Computer Science 2523:29–45, 2002.
- [BCDG08] A. Berzati, C. Canovas, J.-G. Dumas, and L. Goubin. Fault Attacks on RSA Public Keys: *Left-To-Right* Implementations are also Vulnerable. *Topics in Cryptology — CT-RSA’09*, Lecture Notes in Computer Science 5473:414–428, 2008.
- [BCG08] A. Berzati, C. Canovas, and L. Goubin. Perturbating RSA Public Keys: an Improved Attack. *Workshop on Cryptographic Hardware and Embedded Systems — CHES’08*, Lecture Notes in Computer Science 5154:380–395, 2008.
- [BCMCC06] E. Brier, B. Chevallier-Mames, M. Ciet, and C. Clavier. Why One Should Also Secure RSA Public Key Elements. *Workshop on Cryptographic Hardware and Embedded Systems — CHES’06*, Lecture Notes in Computer Science 4249:324–338, 2006.
- [BDH⁺97] F. Bao, R. H. Deng, Y. Han, A. Jeng, A. D. Narasimhalu, and T. Ngair. Breaking Public Key Cryptosystems on Tamper Resistant Devices in the Presence of Transient Faults. *Proceedings of the 5th Workshop on Secure Protocols*, Lecture Notes in Computer Science 1361:115–124, 1997.
- [BDL97] D. Boneh, R. A. DeMillo, and R. J. Lipton. On the Importance of Checking Cryptographic Protocols for Faults. *Advances in Cryptology — EURO-CRYPT’97*, Lecture Notes in Computer Science 1233:37–51, 1997.
- [BECN⁺04] H. Bar-El, H. Choukri, D. Naccache, M. Tunstall, and C. Whelan. The Sorcerer’s Apprentice Guide to Fault Attacks. IACR eprint archive: <http://eprint.iacr.org/2004/100>, 2004.
- [BMM00] I. Biehl, B. Meyer, and V. Muller. Differential Fault Attacks on Elliptic Curve Cryptosystems. *Advances in Cryptology — CRYPTO’00*, Lecture Notes in Computer Science 1880:131–146, 2000.

- [BOS05] J. Blömer, M. Otto, and J.-P. Seifert. Sign Change Fault Attacks on Elliptic Curve Cryptosystems. *Fault Diagnosis and Tolerance in Cryptography — FDTC'05*, Lecture Notes in Computer Science 4236:25–40, 2005.
- [BS97] E. Biham and A. Shamir. Differential Fault Analysis of Secret Key Cryptosystems. *Advances in Cryptology — CRYPTO'97*, Lecture Notes in Computer Science 1294:513–525, 1997.
- [CJ05] M. Ciet and M. Joye. Elliptic Curve Cryptosystems in the Presence of Permanent and Transient Faults. *Design, Codes and Cryptography*, 36(1):33–43, 2005.
- [DKL⁺98] J.-F. Dhem, F. Koeune, P.-A. Leroux, P. Mestré, J.-J. Quisquater, and J.-L. Willems. A Practical Implementation of the Timing Attack. *Smart Card Research and Advanced Applications — CARDIS'98*, Lecture Notes in Computer Science 1820:167–182, 1998.
- [ElG85] T. ElGamal. A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms. *IEEE Transactions on Information Theory*, 31(4):469–472, 1985.
- [FIPS182] U.S. Department of Commerce/National Institute of Standards and Technology (NIST). Digital Signature Standard (DSS), FIPS PUB 182-2change1.
- [Gir05] C. Giraud. Fault Resistant RSA Implementation. *Fault Diagnosis and Tolerance in Cryptography — FDTC'05*, Lecture Notes in Computer Science 4236:142–151, 2005.
- [GT04] C. Giraud and H. Thiebauld. A Survey on Fault Attacks. *Smart Card Research and Advanced Applications — CARDIS'04*, 2004.
- [KJJ99] P. Kocher, J. Jaffe, and B. Jun. Differential Power Analysis. *Advances in Cryptology — CRYPTO'99*, Lecture Notes in Computer Science 1666:388–397, 1999.
- [Koc96] P. Kocher. Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. *Advances in Cryptology — CRYPTO'96*, Lecture Notes in Computer Science 1109:104–113, 1996.
- [Mui05] J. A. Muir. Seifert's RSA Fault Attack: Simplified Analysis and Generalizations. IACR eprint archive: <http://eprint.iacr.org/2005/458>, 2005.
- [PQ03] G. Piret and J.-J. Quisquater. A Differential Fault Attack Technique Against SPN Structures, with Application to the AES and KHAZAD. *Workshop on Cryptographic Hardware and Embedded Systems — CHES'03*, LNCS 2779:77–88, 2003.

-
- [RSA78] 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.
- [SA02] S. Skorobogatov and R.-J. Anderson. Optical Fault Induction Attacks. *Workshop on Cryptographic Hardware and Embedded Systems — CHES’02*, Lecture Notes in Computer Science 2523:2–12, 2002.
- [Sei05] J.-P. Seifert. On authenticated computing and RSA-based authentication. *Proc. of the 12th ACM conference on Computer and Communications Security — CCS’05*, pages 122–127, 2005.

Turning Physics to Advantage with Fuzzy Extractors

In this chapter, we explore the *generation* of unique identifiers from physical objects. The material under consideration is secure paper. Moreover, we show how to establish a strong link between the paper medium and the data represented on it. This is an interesting alternative to defeat unauthorized copy and content modification attempts. Many applications would benefit from it, such as show tickets, contracts, banknotes or medical prescripts. In this chapter, we present a low cost solution that establishes such a link by combining digital signatures, physically unclonable functions [Pap, PRTG02] and fuzzy extractors [DORS08]. The proposed protocol provides two levels of security that can be used according to the time available for verifying the signature and the trust in the paper holder. In practice, our solution uses ultra-violet fibers that are poured into the paper mixture. Fuzzy extractors are then used to build identifiers for each sheet of paper and a digital signature is applied to the combination of these identifiers and the data to be protected from copy and modification. We additionally provide a careful statistical analysis of the robustness and amount of randomness reached by our extractors. We conclude that identifiers of 72 bits can be derived, which is assumed to be sufficient for the proposed application. However, more randomness, robustness and unclonability could be obtained at the cost of a more expensive process, keeping exactly the same methodology.

3.1 Introduction

Securing documents is an important topic in our everyday life. Bank notes are probably the most obvious example and it involves various techniques, *e.g.* micro-printing, ultra-violet inks, ... that aim to make their falsification difficult. But in fact, many other documents are concerned, *e.g.* show tickets, legal papers or medical prescripts. Even passports that now embed an RFID chip are still enhanced with such physical protections of which the goal is to prevent counterfeiting. In other words, they aim to render the effort for producing good-looking fakes prohibitively high. In general, any field where the authenticity of a document is important would benefit from a way to prevent duplication

and/or modification. But in practice, the cost of the proposed solutions also have to be traded with the security level that has to be reached.

As a matter of fact, the topic of anti-counterfeiting techniques is very broad. Even when restricting the concern to paper documents, various contributions can be found in the scientific literature and in products of companies. For example, Laser Surface Authentication (LSA) extracts a unique fingerprint from almost any document and packaging, based on intrinsically occurring randomness measured at microscopic level using a laser diode, lenses and photodetectors [BCJ⁺05]. The scheme is shown to be highly resistant to manipulation. Moreover, the authors of [Ing] suggest that the fingerprint could be stored on the document itself through an encrypted or digitally signed 2D barcode or a smart chip. Similarly, printed papers can be authenticated using the inherent non-repeatable randomness from a printing process [ZWK03]. Here, the misplacement of toner powder gives rise to a *print signature* that is experimentally shown to be unique and random. It is then explained how to exploit such a print signature in order to design a scheme ensuring the authentication of a document.

A similar idea is developed in this chapter, *i.e.* we aim to bind the fingerprint of the medium and the data lying on it. Like in the print signature, this idea is achieved by performing a digital signature on these informations as a whole. But contrary to [ZWK03] where the *fingerprint* is printed on the paper and analyzed using shape matching methods (a well-studied problem in computer vision), we make the fingerprint intrinsic to the paper. For this purpose, we incorporate ultra-violet fibers during the fabrication process of the paper. Our solution then relies on the combination of a Physically Unclonable Function (PUF) with robust cryptography and randomness extraction schemes. That is, we use fuzzy extractors to build unique identifiers from the fiber-enhanced papers. Importantly, we mention that using fibers as a PUF was motivated by low-cost applications (*e.g.* medical prescriptions, typically). Hence, the actual unclonability of the proposal is only conjectured for low-cost adversaries. But increasing the unclonability by considering more complex physical sources of randomness would be feasible at the cost of a more expensive process. Eventually and additionally to the description of our protocol for copy or modification detection, we pay a particular attention to the careful statistical analysis of the robustness and amount of randomness that are extracted from the papers.

Note that from a theoretical point of view, such a $Sign(content, container)$ scheme could be applied to any object. Making it practical only requires a way to robustly measure the intrinsic features of a medium and to embed a digital signature. But such an adaptation should also be considered with care since each ingredient of the protocol could be tuned in function of the target application. In other words, the solution we propose is general, but finding the best tradeoff between cost and security for a given application is out of our scope.

The rest of the chapter is organized as follows. Section 3.2 gives a global overview of the proposed method and points out its requirements. The ingredients of our protocol are described in Section 3.3. The main contribution is then presented in Section 3.4 in which the paper case study is investigated and evaluated. Eventually, conclusions are given in Section 3.5.

3.2 Overview

In this section, a general overview of the process that we propose for paper authentication is sketched. The components of the scheme will be discussed afterwards. First, the signature of a document works as follows.

1. Some additional agent is poured into the paper paste to make secure sheets (Fig. 3.1). For example, we use ultra-violet fibers in our application.
2. The physical features of the paper are then extracted, encoded into a tag $T1$, and printed on the paper to authenticate (Fig. 3.2).
3. Some actual content is printed on the paper (Fig. 3.3).
4. Eventually, this content is concatenated to the physical information and signed. The digital signature is encoded in a second tag, $T2$ (Fig. 3.4).

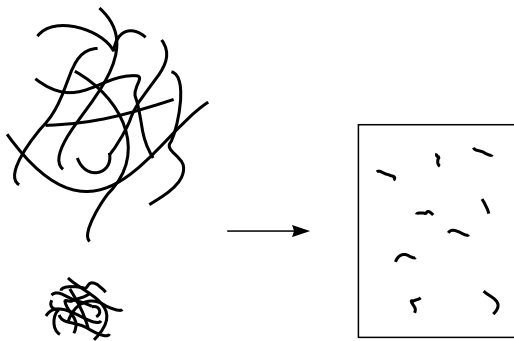


Figure 3.1: Making paper.

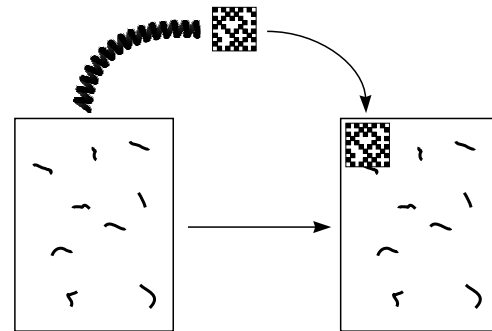


Figure 3.2: Extracting physical features.

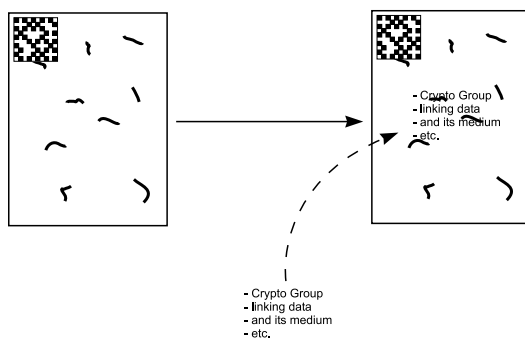


Figure 3.3: Adding content.

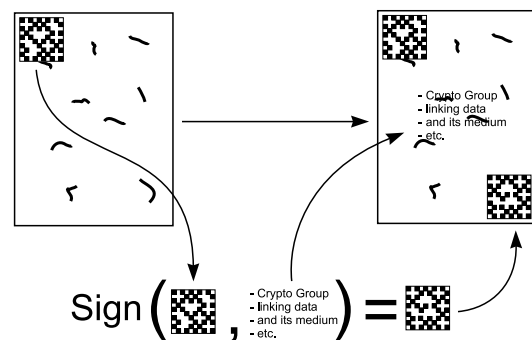


Figure 3.4: Signing container + content.

Second, in order to check whether the document is a genuine one, two levels of security can be considered. These two levels are neither mandatory nor exclusive. Whether

one or both of them should be used is driven by the application. For example, in the context of medical prescription, the pharmacist may decide to apply the first level of security for his usual customers and the second level for a person he has never seen before. The verification works as follows.

Level 1. The verifier trusts the second step of the previous procedure and only performs the verification of the digital signature using $T1$, $T2$ and the content of the document (Fig. 3.5). Interestingly, this process does not necessarily require the use of optical character recognition since the paper content might have been summarized in $T2$. Of course, this implies that the paper content fits into the tag size constraints.

Level 2. A full verification is performed (Fig 3.6).

1. The physical features of the medium are extracted and encoded in a new tag T^* . This tag T^* is then compared with $T1$ and the document is rejected if the two tags do not match.
2. If both tags do match, then the verification proceeds like before. Using T^* or $T1$ makes no difference at this stage.

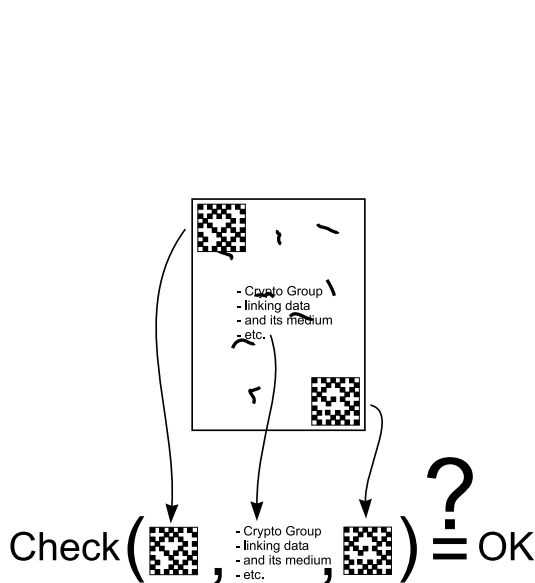


Figure 3.5: Verification when the owner of the document is trusted.

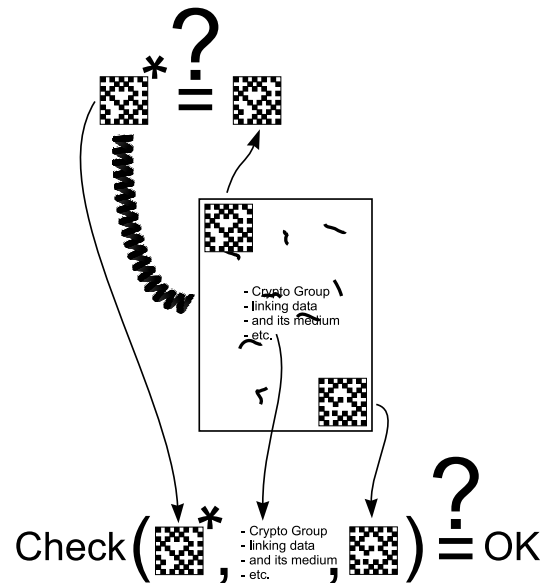


Figure 3.6: Full verification with extraction of the medium's physical features.

3.3 Ingredients

To be implemented in practice, the previous authentication process requires different ingredients, namely a digital signature algorithm, a tag encoder/decoder, a physical unclonable function (PUF) and an extraction scheme. Proposals for each of those ingredients

are discussed in the present section. We mention that these choices are not expected to be optimal but to provide a solution that reasonably fits to the low cost requirements of our target applications. Hence, they could be improved or tuned for other applications.

3.3.1 Signature

Digital signature is a well studied field in cryptography and a variety of solutions are available for different applications. When devising a real application, using a standard is a natural way to go. For this reason, the Elliptic Curve Digital Signature Algorithm (ECDSA [X962]) was chosen. Because it provides relatively short signatures, this leaves space for possible addition of content to the tag.

3.3.2 Tag En-/De- coding

As for digital signature, a large number of visual codes exist, from the old bar code to more complex 2-dimensional codes. A list of commonly used 2D bar codes is available on the web [Uni]. The Datamatrix [ISO16022] was chosen for its high density within a small area (up 1500 bytes in a single symbol whose size can reduce to a bit more than a 1-inch square with a 600 dpi printer, see Fig. 3.7).

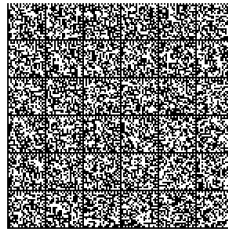


Figure 3.7: A datamatrix encodes large amount of data in small areas.

3.3.3 PUF: Physical Unclonable Function

PUF or equivalently physical one-way functions were introduced by Pappu [Pap, PRTG02] in 2001. A PUF is an object whose function can easily be computed but is hard to invert. For a more in-depth view of PUF and its use in cryptography, we refer to Pim Tuyls *et al.*'s book [TSK07]. As it will be used in this chapter, every challenge sent to a PUF (*i.e.* each time a given sheet of paper is scanned) should be answered by almost the same response (*i.e.* picture). It should then be ensured that the size of the response set is large enough to prevent collisions (*i.e.* different sheets of paper should not output the same response). Also, the protected papers should be hard enough to clone. With this respect, PUFs generally rely on some theoretical arguments borrowed from physics. In our case and as mentioned in the introduction of this chapter, we consider a weaker type of PUF that is just expected to be hard to clone by a low-cost adversary. According to the papermaker [AW], systematically generating twins of fiber-enhanced papers is an expensive

process. But there is still the option for an attacker to scan the sheet of paper under ultra-violet illumination and attempt to carefully reproduce the fibers on a clear sheet of paper. This is exactly what we assumed to be hard enough to be considered as a real threat in our context. We mention again that the focus is not in finding the best PUF but in devising a complete solution to prevent paper modification and copy and evaluate its reliability in terms of randomness and robustness.

3.3.4 Physical Extraction

The major problem when extracting fingerprints from physical objects is that they are usually not usable for cryptographic purposes. This difficulty arises from the fact that (1) the extracted fingerprint may not be perfectly reproduced, due to small variations in the measurement setup or in the physical source and (2) the extracted fingerprints from a set of similar objects may not produce the uniform distributions required by cryptography. Hopefully, turning noisy physical information into cryptographic keys can be achieved using *fuzzy extractors*.

Theory

The idea of fuzzy extractors arose from the need to deal with noisy data. Therefore, the building parts were somehow spread throughout the literature until Dodis *et al.* [DORS08] gathered them all into a general and unified theory.

Instead of devising with fuzzy extractors immediately, the notion of secure sketches is introduced. A secure sketch is a pair of functionalities: *sketch* and *recover*. First, upon input t , *sketch* outputs a string s . Then, when *recover* receives input t' and the sketch-computed s , it outputs t provided t' is close enough to t . The required property that t remains largely unknown even though s is available ensures the security of the sketch.

The main result of [DORS08] is to prove that (and show how) fuzzy extractors can be built from secure sketches using strong randomness extractors. The fuzzy extractor builds upon secure sketches as depicted in Fig. 3.8. In an enrollment phase, the input to the sketch procedure is also sent to a strong extractor, together with randomness u , which generates output R . The pair (u, s) is stored as data helper, W . Then, in a reconstruction phase, the data helper is used to regenerate the output R from a new input t' through *recover* and *extract*. In practice, the construction of secure sketches requires a metric to quantify closeness, *e.g.* hamming distance, set difference or edit distance. An example using the hamming distance metric will be discussed in the next section.

Practice

In this section, an overview of the practical approach developed by Tuyls *et al.* [TSS⁺06] is given. The same approach will be used for the paper case. Their article deals with proof-read hardware, *i.e.* a piece of hardware within which the key is not stored but is regenerated when needed. In order to achieve this, two additional layers are placed on the top of an integrated circuit. The first one is a grid of capacitive sensors and the second one is an opaque coating containing two kinds of dielectric particles. The capacitances of this

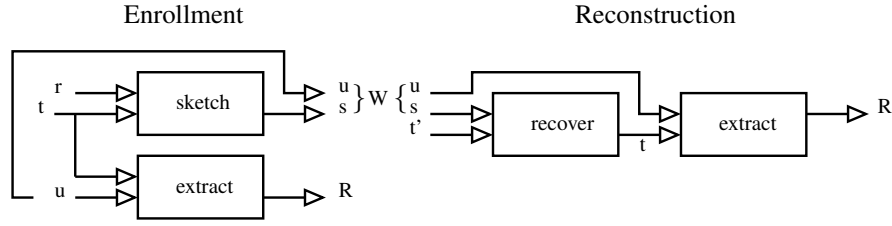


Figure 3.8: Turning a secure sketch into a fuzzy extractor.

coating are characterized both across the set of built chips and within a set of measures for the same chip in order to approximate their inter- and intra-class distributions (that are generally assumed to be Gaussian). Comparing the standard deviations of these distributions (as in Fig. 3.9) already gives an intuitive insight on the feasibility to discriminate different chips. The fuzzy extractor then works as theoretically described in the previous section: an enrollment is performed once to compute the extracted fingerprint and its data helper; then reconstruction is performed any time the key is needed. The global scheme is depicted in Fig. 3.11.

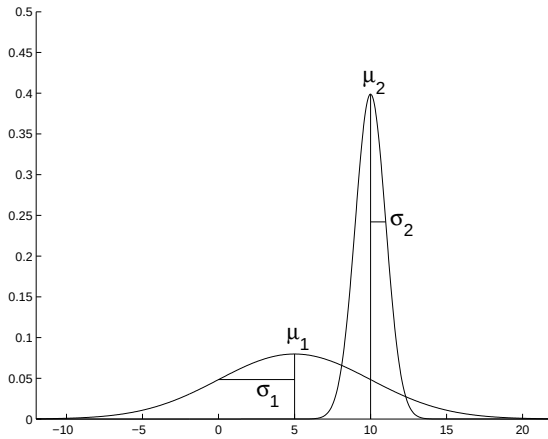
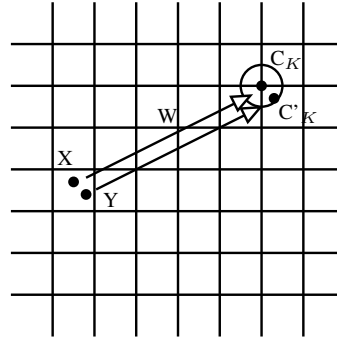


Figure 3.9: Comparing the capacitance inter- and intra-class distributions.

Figure 3.10: Using the data helper W to recover a key from a measurement.

During the enrollment phase (left part of Fig. 3.11), all sensors on a chip measure the local capacitance of the coating. A first part of the data helper, denoted as w^* , is built as the shift to center the measures in the interval they lie in. Those intervals are also used to convert each measured value to a short sequence of bits. The concatenation of those short sequences, the fingerprint X is used together with the codeword C_K (hiding the key K) to generate the second part of the data helper, denoted as $W = X \oplus C_K$.

When reconstructing the key (right part of Fig. 3.11), each output value of the sensors is corrected with the first part of the helper data and then mapped to a short sequence of bits whose concatenation is denoted Y . Provided that Y is not too far away from X (in the sense of hamming distance), C_K can be recovered by decoding $Y \oplus W$. As the map between the key K and the codeword C_K is uniquely defined, K is immediately identified by C_K (see the right part of Fig. 3.10). Hence, K can be regenerated at will

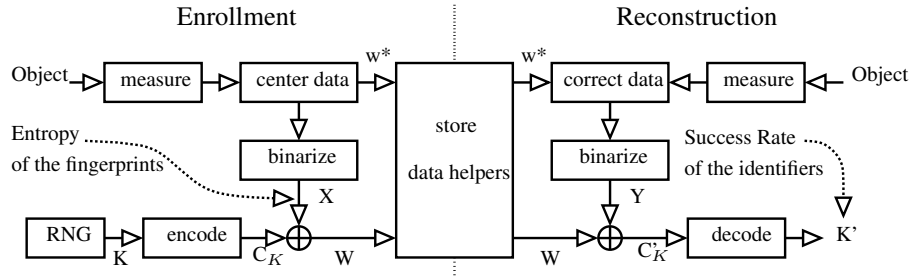


Figure 3.11: Global view of the scheme: Enrollment and Reconstruction phases.

without worrying about the measurement variations. But if the measures are too far from the ones used during the enrollment, K will not be recovered. And this is exactly the expected behaviour: if the measures are too different, the chips assume to be attacked and hence should prevent access to the key. We refer to [TSS⁺06] for more details.

3.4 The Paper Case

To avoid confusion, it is worth clarifying that in this work, *fingerprint* denotes the X or Y bit string built from the physics (during enrollment or reconstruction), whereas *identifier* stands for the K bit string generated from a random source. In Tuyls *et al.*, K is a key that could be used for encryption. In our case, K is an identifier that can be recovered. This difference will be reminded later.

From the description of previous section, there are three main steps in the extraction of a fingerprint from random physical measurements, namely the measurement of the physical feature, the characterization of its probability distribution and the generation of the bit sequences (X or Y). For the paper case, the obvious measurement tool is the one mimicking the human eye. The approach that we chose was to slightly modify a scanner by replacing its white tube by a fluorescent lamp. The measurement that is performed is thus a uv-scan of the paper sheet outputting a 24-bit colour picture. Using image processing techniques, a list of fibers is then established, each of which is described as tuple containing position, orientation, surface and colour (YUV-components). Given this as input, the characterization of the probability distributions depends on mainly three parameters that we detail in the rest of this section.

Number of sensors. As described in Fig. 3.12, a sheet of paper can be divided in different equal-area sub-sheets. By analogy with the previous coating PUF example, we denote each of those sub-sheets as a sensor. Quite naturally, one may decide to extract (a lot of) information from a single sensor or (less) information from several sensors considered separately.

Number of features per sensor. Given one sensor, we can try to measure different features from its fibers. For example, we could measure the amount of fibers N , their

orientation O , their luminance L or their overall surface S . In the following, we will consider those four exemplary features.

Number of bins per sensor. Eventually, the amount of information to extract from each sensor and feature has to be determined inter-class distributions. It generally results in a tradeoff between information and robustness. For example, Fig. 3.13 depicts an (imaginary) inter-class distribution. This distribution is sliced in 4 in its bottom part and in 8 in its upper part. These 4-bin and 8-bin strategies will result in a maximum entropy of 2 or 3 bits of information per sensor and feature. Note that the short bit sequences attached to each of the bins are the binary strings of a Gray code (the same way as in [TSS⁺06]), which allows improving the robustness of the extraction: if one of the sensors is slightly deviating from its enrollment value, moving the measure from one bin to its neighbour will result in a string Y that only differs by 1 bit position from the enrollment string X . Hence, such an error will be easily corrected when decoding.

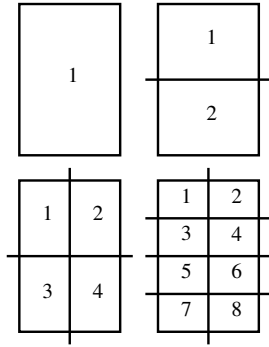


Figure 3.12: Splitting fiber-enhanced paper into two, four and eight sensors.

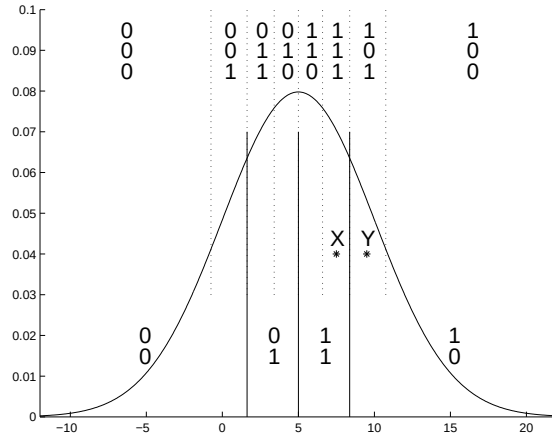
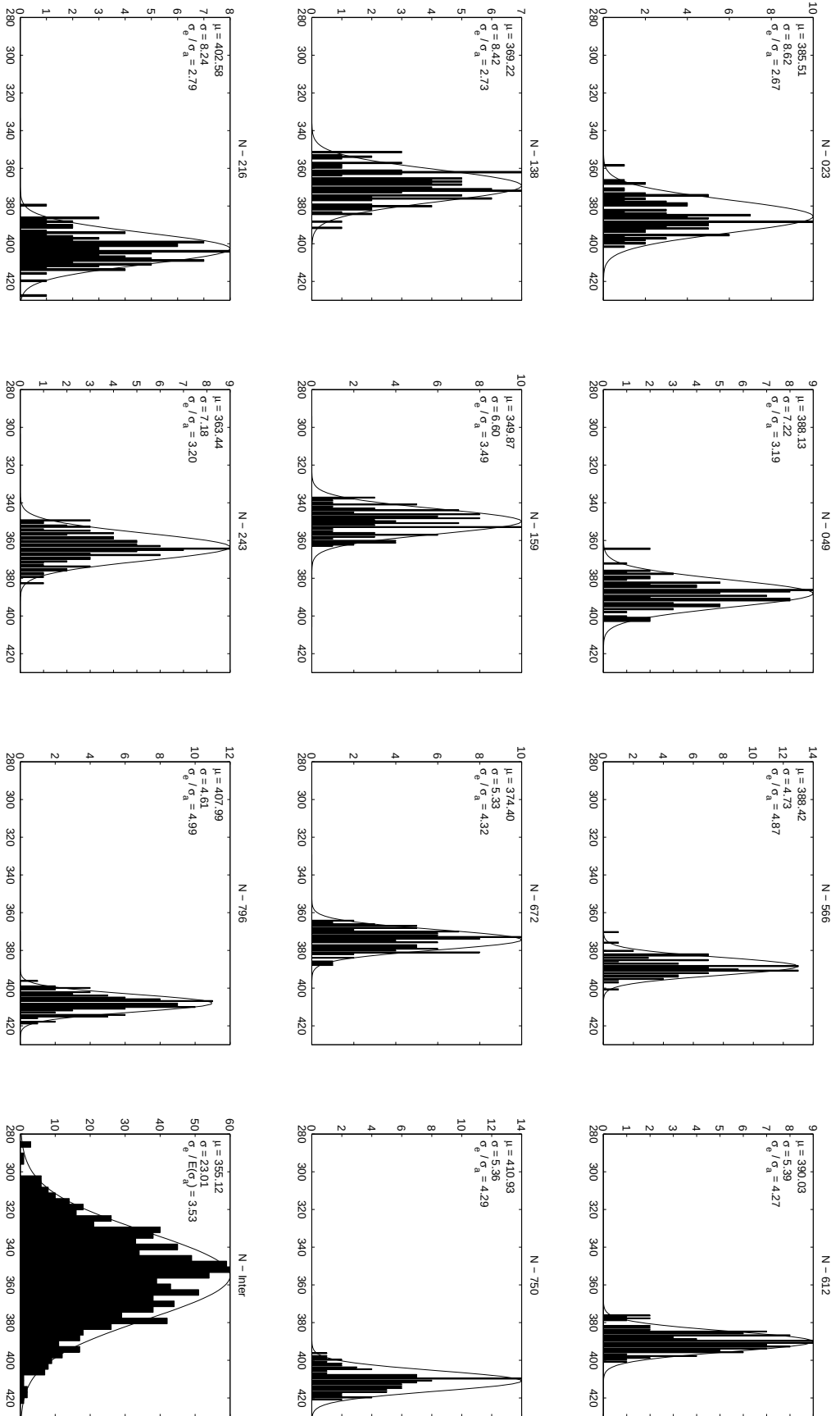


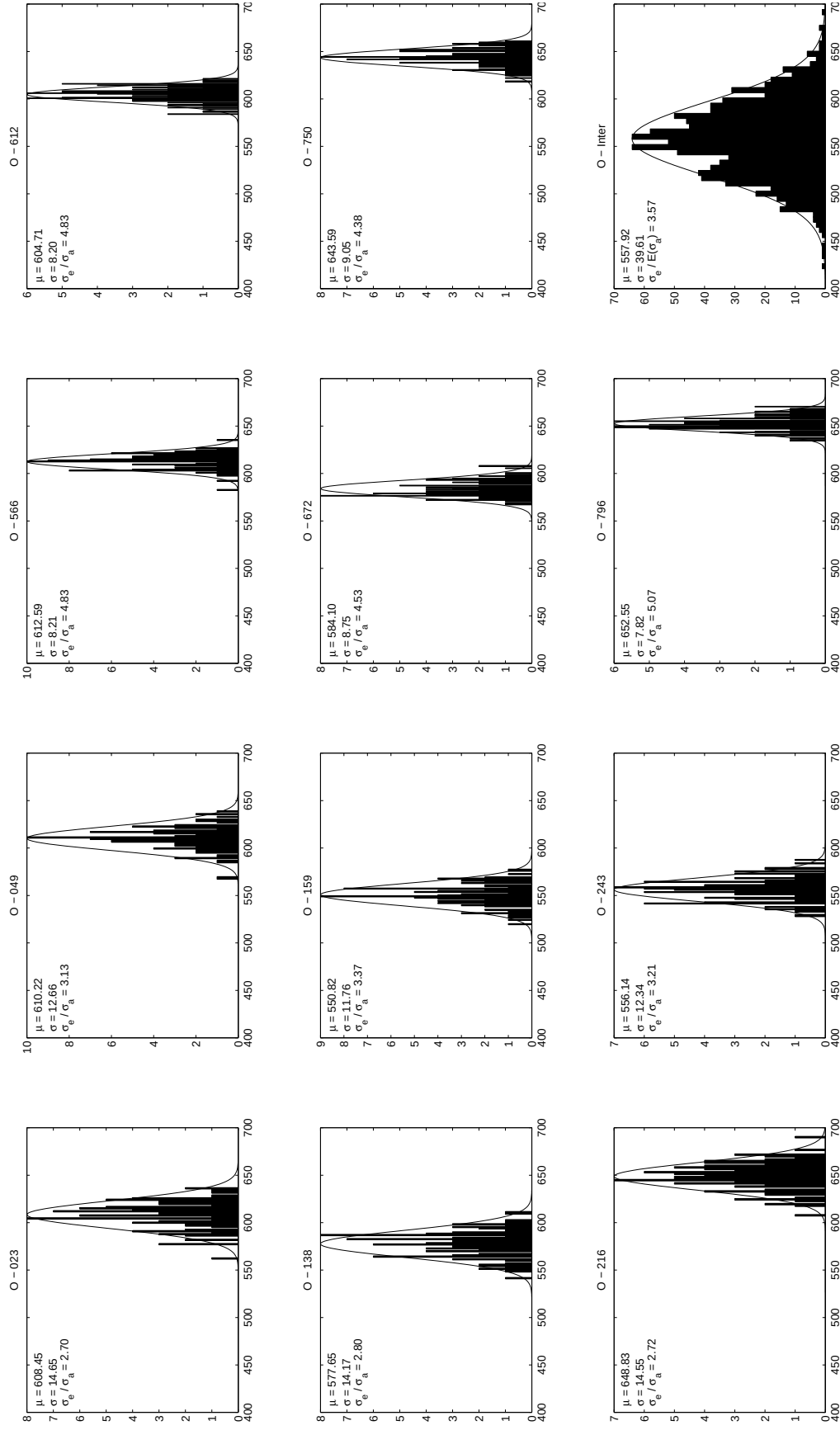
Figure 3.13: Characterizing the inter-class distribution and Gray codes.

In order to characterize the paper, the whole set of available sheets has been scanned (1000 different sheets) as well as 16 randomly chosen sheets that were scanned 100 times. The Lilliefors test was applied to all the samples to check whether the measurements match a normal distribution, which was actually the case. As an illustration, the intra- and inter-class distributions of two features (amount of fibers N and orientation O) are provided in Fig. 3.14 and Fig. 3.15 respectively. Note that we performed measurements with and without rulers in order to additionally evaluate the impact of two slightly different setups. A small but noticeable improvement when using rulers can be seen between the two columns on the left and the two on the right of the appendix figures (with the exception of the bottom right picture that represents the inter-class distribution).

3.4.1 Evaluation Criteria

To evaluate the protocol, two different aspects need to be evaluated: the robustness of the process and the entropy provided by the sheets of secure paper.

Figure 3.14: Intra- and inter-class distributions for the amount of fibers N .

Figure 3.15: Intra- and inter-class distributions for the orientation of the fibers O .

Robustness. We first need to ensure that a correctly scanned paper will be recognized immediately (without requiring additional scan of the same sheet). It is estimated through the success rate (SR). This later one is simply computed as the ratio between the amount of correct measures and the amount of measures.

Entropy. We then need to measure the amount of information that can be extracted from the secure paper. Entropy estimations will be used to evaluate the number of different fingerprints that we can expect from the process.

3.4.2 Analysis

Robustness

We first evaluated the success rate, *i.e.* the robustness, at the output of the scheme, see Fig. 3.11. This was done using the 16 sheets that were scanned 100 times. For each of those 16 sheets, the enrollment phase was performed with one of the scan to build both parts of the data helper. Then, the reconstruction phase was carried out for the remaining 99 scans. Each time the identifier generated during the enrollment is correctly recovered, it increases the success rate. The result for the N feature is shown in Fig. 3.16 without error-correcting codes (ECC) and in Fig. 3.17 when a BCH(7, 4, 1) code is applied to improve the robustness.

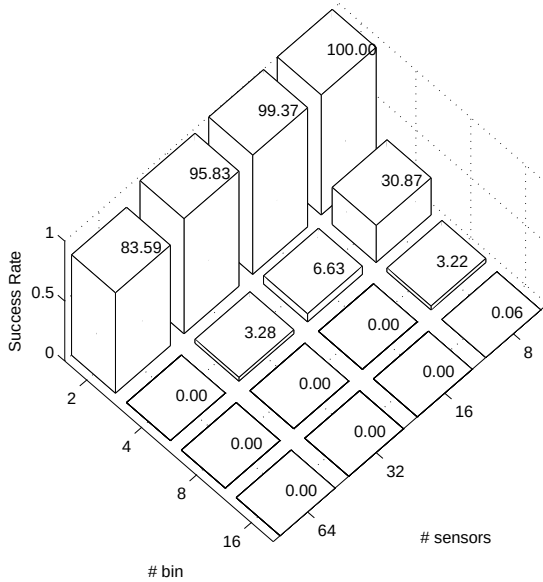


Figure 3.16: Success rate *without* error-correcting codes (N feature).

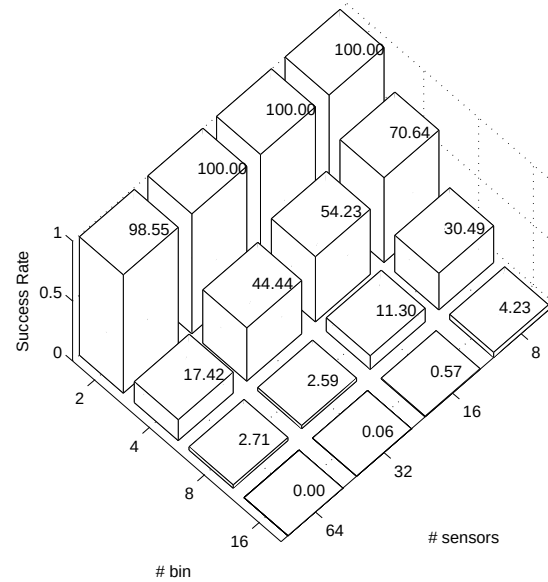


Figure 3.17: Success rate *with* error-correcting codes (N feature).

When multiple features and multiple sensors are taken into account, the fingerprint is built as the concatenation over the sensors of the concatenation over the features of the

Gray codes:

$$X = \parallel_S (\parallel_F \text{GC}(F, S)) = \underbrace{NOLS}_{S_0} \parallel \underbrace{NOLS}_{S_1} \parallel \cdots \parallel \underbrace{NOLS}_{S_S}$$

The resulting success rate is pictured Fig. 3.18 and 3.19, without and with BCH(7, 4, 1) code respectively. Quite naturally, increasing either the number of sensors or the number of bins decreases the success rate. However, it is also clear that increasing the number of sensors is a much better approach than increasing the number of bins with respect to robustness.

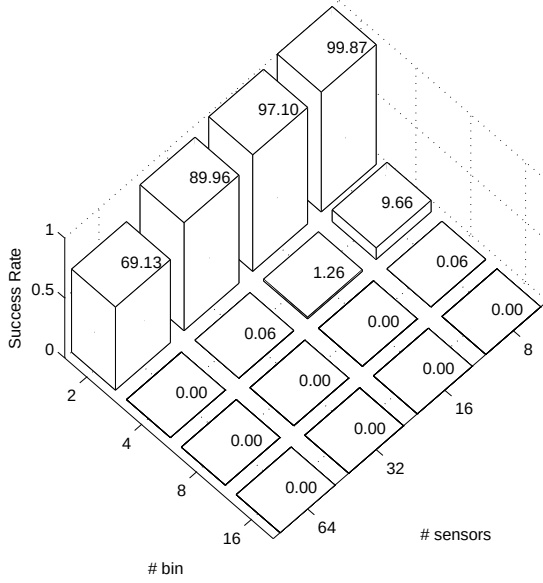


Figure 3.18: Success rate *without* error-correcting codes (4 features).

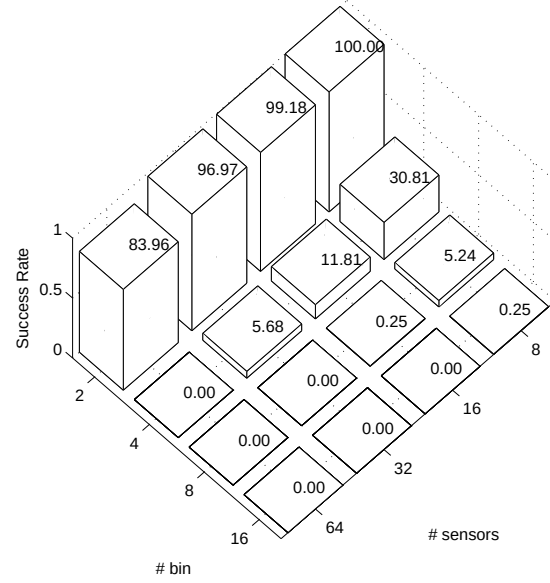
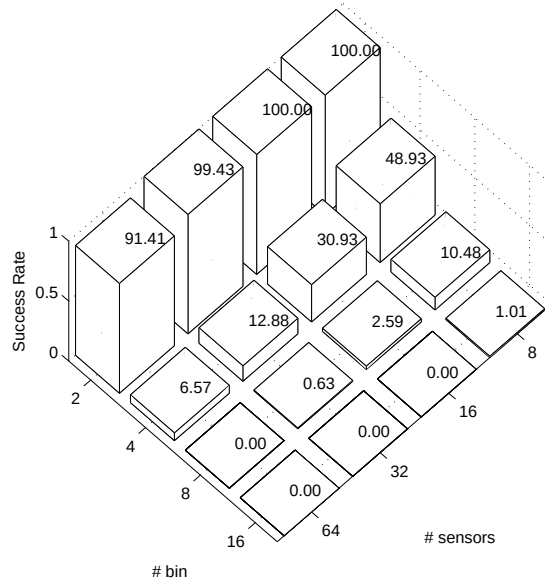


Figure 3.19: Success rate *with* error-correcting codes (4 features).

Other parameters for the error-correcting code can be plugged in. For example, Fig. 3.20 and Fig. 3.21 use BCH codes able to correct up to 2 and 3 errors, respectively. In the case of 64 sensors, 2 bins and BCH(31, 16, 3), this improves the success rate up to 95%, a reasonable target value for real applications.

Finally, we also evaluated the impact of embedding a ruler in the scanner to ensure proper positioning of the sheet before scanning. Out of the 16 sheets scanned 100 times, 6 were scanned after the ruler was setup. The difference already mentioned in the appendix is now easily observed in Fig. 3.22 and 3.23 that have to be compared with Fig. 3.19 where all the sheets were evaluated without distinguishing the use (or not) of the ruler.



Entropy

In order to estimate the entropy of the physical source, we used 1000 fingerprints generated as pictured in Fig. 3.11. We started with a simple sensor-based approach in which we evaluated the entropy using histograms. We note that computing the entropy per sensor is meaningful as long as these sensors can be considered independent. This seems a reasonable physical assumption in our context. By contrast, it is obviously not a good solution to evaluate the entropy when exploiting different features that are most likely correlated. Anyway, the histogram-based approach was just applied for intuition purposes and combined with the more advanced evaluations described in the end of this section.

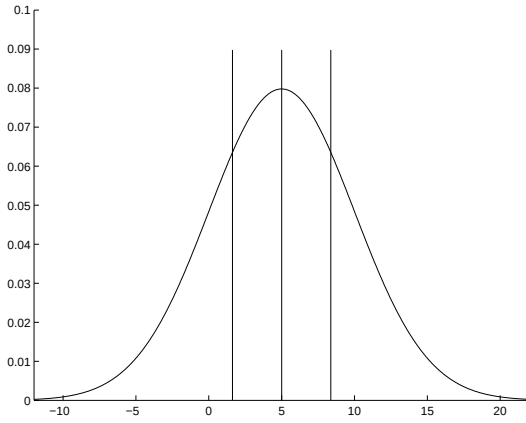


Figure 3.24: One set to estimate the distribution and build the bins.

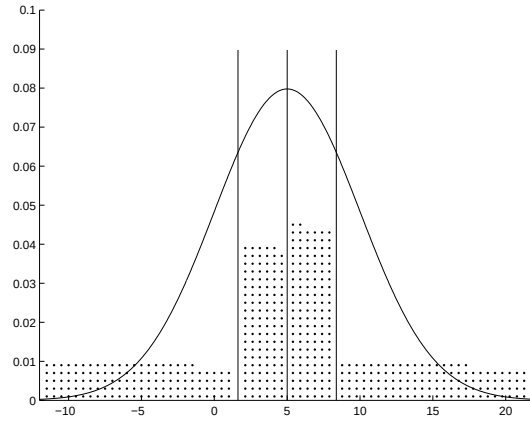


Figure 3.25: Each sheet of the second set is placed in its corresponding bin.

In practice, let us assume a single sensor per sheet and 4 bins per sensor. We first used 500 scans to determine the positions of the bins as in Fig. 3.24. Then, we used the second set of 500 scans to evaluate the probabilities of the previously determined bins as in Fig. 3.25. Eventually, we estimated the entropy per sensor using the classical formula: $H = -\sum_i p_i \log p_i$ where the p_i are the bin probabilities. Such a naive method gives the results shown in Fig. 3.26.

Note that for some choices of parameters, the entropy was stuck to zero. This is simply because the number of samples was not sufficient to fill all the bins in those cases. Indeed, given the size of the sample set, one can determine the amount of bins that should not be crossed to keep meaningful results, *e.g.* using Sturges rule¹: $\lceil 1 + \log_2 M \rceil$, with M the size of the set. In our example, it states that there should be no more than 10 bins. The entropies stuck to ground in Fig. 3.26 can be seen as the limits for the given sample size. Quite naturally, we see that the informativeness of an extractor increases with the number of sensors and bins, contrary to its robustness in the previous section.

In order to confirm these first estimations, we then ran more complex test suites, in particular: ent [Wal], Maurer's test including Coron's modification [MvOV01, CN98, Cor99] and the Context Tree Weighting (CTW) method [Wil94, WST, WST95]. The main

¹Scott's formula gives a similar result: $\frac{3.5\sigma}{M^{1/3}} = \frac{3.5 \cdot 23}{500^{1/3}} = 10.146 \dots$

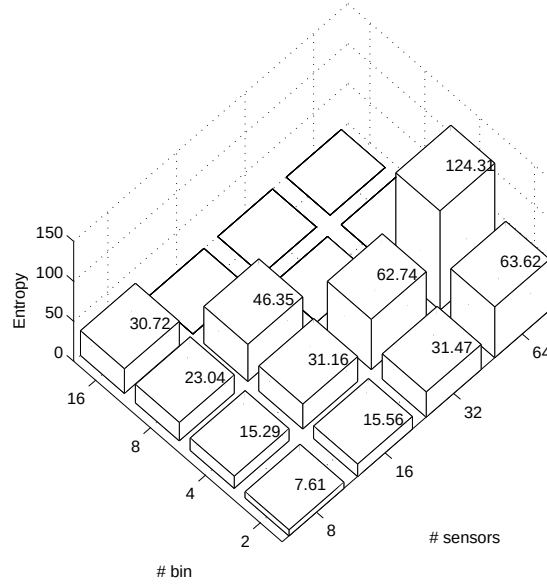


Figure 3.26: Sensor-based entropy estimation using histograms for the N feature.

idea behind these entropy estimation tools is to compare the length of an input sequence and its corresponding compressed output. For the details about how they actually process the data, we refer to the previous links.

These final results achieved are given in table 3.1, where $X(F, S, B)$ denotes the fingerprint built upon features F when cutting the paper in S sensors sliced in B bins. As previously explained, when multiple features and multiple sensors are involved, the fingerprint is built as the concatenation over the sensors of the concatenation over the features of the Gray codes, $X = \parallel_S (\parallel_F \text{GC}(F, S))$. The first column where fingerprints are only built from the amount of fibers (N) shows that almost 32 bits of entropy can be extracted from the 32-bit strings which essentially confirms that different sensors are indeed independent. By contrast, when using 4 different features as in the right part of the table, we clearly see that the entropy extracted per bit of fingerprint is reduced, *i.e.* the features (amount of fibers, orientation, luminance and surface) are actually correlated. Most importantly, we see that overall the proposed solution allows generating fingerprints with slightly more than 96 bits of entropy while ensuring a good robustness. In other words, this solution largely fulfills the goal of a low-cost authentication process that we target in this paper.

	$X(N, 32, 2)$	$X(NOLS, 32, 2)$
Ent	$1 \cdot 32 \cdot 0.99 = 31.68$	$4 \cdot 32 \cdot 0.99 = 126.72$
Maurer*	$1 \cdot 32 \cdot 0.99 = 31.68$	$4 \cdot 32 \cdot 0.63 = 80.64$
CTW	$1 \cdot 32 \cdot 0.99 = 31.68$	$4 \cdot 32 \cdot 0.75 = 96.53$

Table 3.1: Entropy estimations in entropy bits per fingerprint X .

Note finally that our use of fuzzy extractors significantly differs from the one of Tuyls *et al.*. We use physical features to build unique (but public) identifiers while [TSS⁺06]

aims to generate cryptographic keys. Therefore, we do not have to evaluate the secrecy of our identifiers but only their randomness. This is because in our protocol, the overall security comes from the digital signature that is applied both to the identifiers and to the content printed on a paper. An attack against our scheme would require to find a sheet of paper that gives the same identifier to perform a copy/paste attack. This is supposed to be hard enough in view of the 96 bits of entropy that the physical features assumably provide.

3.5 Conclusion

In this chapter, a proposal to secure documents is presented that combines robust cryptographic mechanisms and information extractors with a source of physical randomness. It has the interesting feature to provide two levels of verification, trading speed for trust. The scheme is quite generic and could be easily tuned for different application needs. Our case study was developed for low-cost standard desktop equipment. But the robustness, randomness and (mainly) unclonability of our proposal could be improved at the cost of a more expensive infrastructure. We also provide a detailed and motivated statistical analysis of the information extraction scheme. In the studied case, embedded ultra-violet fibers allows extracting 128-bit strings that correspond to an entropy of approximately 96 bits while providing 72-bit identifiers when applying an error correcting code. The resulting identifiers can be extracted with high robustness. This is considered as providing a sufficient security since an adversary would have to scan a prohibitive number of secure paper sheets to find a collision.

Bibliography

- [AW] Arjo Wiggins: Security Division. <http://www.security.arjowiggins.com/>.
- [BCJ⁺05] J. D. R. Buchanan, R. P. Cowburn, A.-V. Jausovec, D. Petit, P. Seem, G. Xiong, D. Atkinson, K. Fenton, D. A. Allwood, and M. T. Bryan. Fingerprinting Documents and Packages. *Nature*, 436:475, 2005.
- [CN98] J.-S. Coron and D. Naccache. An Accurate Evaluation of Maurer’s Universal Test. *Selected Areas in Cryptography — SAC’98*, Lecture Notes in Computer Science 1556:57–71, 1998.
- [Cor99] J.-S. Coron. On the Security of Random Sources. *Public Key Cryptography — PKC’99*, Lecture Notes in Computer Science 1560:29–42, 1999.
- [DORS08] Y. Dodis, R. Ostrovsky, L. Reyzin, and A. Smith. Fuzzy Extractors: How to Generate Strong Keys from Biometrics and Other Noisy Data. *SIAM Journal on Computing*, 38(1):97–139, 2008.
- [GCvDD02] B. Gassend, D. Clarke, M. van Dijk, and S. Devadas. Controlled physical random functions. *Proceedings of the 18th Annual Computer Security Conference*, 2002.
- [Ing] Ingenia Technology Ltd. <http://www.ingeniatechnology.com>.
- [ISO16022] International Organization for Standardization – ISO. Information technology — International symbology representation — Datamatrix. ISO/IEC 16022:2000(E), 2000.
- [JW99] A. Juels and M. Wattenberg. A Fuzzy Commitment Scheme. *Conference on Computer and Communications Security — CCS’99*, Proceedings of the 6th ACM conference on Computer and communications security:28–36, 1999.
- [MvOV01] A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 2001.
- [Pap] R. Pappu. Physical One-Way Functions. <http://alumni.media.mit.edu/~pappu/pdfs/Pappu-PhD-POWF-2001.pdf>. Ph. D. dissertation, MIT, 2001.

- [PRTG02] R. Pappu, B. Recht, J. Taylor, and N. Gershenfeld. Physical One-Way Functions. *Science*, 297:2026–2030, 2002.
- [TSK07] P. Tuyls, B. Škorić, and T. Kevenaar. *Security with Noisy Data: Private Biometrics, Secure Key Storage and Anti-Counterfeiting*. Springer-Verlag New York, Inc., 2007.
- [TSS⁺06] P. Tuyls, G. J. Schrijen, B. Škorić, J. van Geloven, N. Verhaegh, and R. Wolters. Read-Proof Hardware from Protective Coatings. *Workshop on Cryptographic Hardware and Embedded Systems — CHES’06*, Lecture Notes in Computer Science 4249:369–383, 2006.
- [Uni] Unibar Inc. Bar Code Page. <http://www.adams1.com/stack.html>.
- [Wal] J. Walker. ENT: A Pseudorandom Number Sequence Test Program. <http://www.fourmilab.ch/random/>.
- [Wil94] F. Willems. The Context-Tree Weighting Method: Extensions. *IEEE Transactions on Information Theory*, 44:792–798, 1994.
- [WST] F. Willems, Y. Shtarkov, and T. Tjalkens. Reflections on “The Context-Tree Weighting Method: Basic Properties”.
- [WST95] F. Willems, Y. Shtarkov, and T. Tjalkens. The Context-Tree Weighting Method: Basic properties. *IEEE Transactions on Information Theory*, 41:653–664, 1995.
- [X962] American National Standards Institute – ANSI. Public key cryptography for the financial services industry, the Elliptic Curve Digital Signature Algorithm (ECDSA). ANSI X9.62:2005, 2005.
- [ZWK03] B. Zhu, J. Wu, and M. S. Kankanhalli. Print Signatures for Document Authentication. *Conference on Computer and Communications Security*, Proceedings of the 10th ACM conference on Computer and communications security:145–153, 2003.

Conclusion

The article of Arjen Lenstra and Eric Verheul, *Selecting Cryptographic Key Sizes* [LV99], is at the basis of the work presented in this thesis. They presented a model to derive key length adapted to a desired security margin. This model is (mainly) based on current data points for various cryptosystems and on current trends in the evolution of computing power and cryptanalysis progress. We made it available online [GB] in order to help when dealing with the important choice of the appropriate length of the key. In this context, we have studied three different aspects related to key length.

First, we added a hardware data point. An attack to solve elliptic curve discrete logarithms over $\text{GF}(2^m)$ was mounted using a cluster of FPGAs and the best known attack, Pollard's ρ method. After a careful study of different architectures, the most promising one was explained in depth. It was shown that the smallest sizes, 113-bit recommended by [CRb], can no longer be considered as safe. On the other hand, larger parameter sizes seem out of reach of today's computing power. It is important to point out that the proposed attack is much more efficient than software versions. Lenstra and Verheul's work is based on the software data points scattered throughout the literature. In their paper, they suggest a way to adapt their results to hardware attacks, but without great confidence in its reliability. This work shows they were right since, once the data of their estimation updated, it exhibits an improvement factor of about 20. This also emphasizes that attack evaluations are of great importance as they are the best tool to evaluate our own security, and more of them should be conducted in hardware. In this respect, those data points may help tune this software-hardware conversion, but more likely they will help us getting a more precise idea of what is achievable in practice. Therefore, public challenges seem a reasonable way to motivate people, and that is what companies tend to do or at least tended to do. While Certicom still maintain their challenges available to the public, the RSA factoring challenge is no longer active. In a way, I understand their point of view. If solving those challenges is just a matter of waiting for new computers and then wait enough time, it does not sound real fun. But on the other way, we definitely need those challenges, so maybe the way to go is to restore their challenges after an appropriate mutation.

In the second place, we considered physical attacks that were intentionally left aside in Lenstra and Verheul's work. While they did not consider implementation defects, we cannot deny the importance of side channel and fault attacks. Protecting against those attacks is not an easy task. In the case of fault attacks, protection usually involves redundancy during computation or checking its outcome. A newly introduced aspect of faulty computation is to alter the public elements. This has already been done for RSA and elliptic curves. In this thesis, we have introduced an adaptation of Seifert's attack to discrete logarithm based scheme. To our knowledge, these are the first attacks on DSA and ElGamal signature schemes by a corruption of only public key elements. The secret key can – theoretically – be fully recovered at the cost of a rather large amount of successful fault injections. We expect improvements will be made in order to reduce that amount of faults. At a much higher level, I hope that someday we will have tools that may automatically handle these hurdles. Programs that automatically generate HDL code from C sources already exists, like ImpulseC [Imp]. Specifically designed for the cryptography field, Cryptol [Gal] is a specification language that can generate software implementation, HDL implementation and formal model for verification. Given what these tools can achieve at this time, I am pretty confident they will be able to manage side channel and fault attacks as additional constraints once those attacks are properly understood. Research in these areas still needs to progress in order to reduce the gap.

Finally, we were interested in the generation of keys. Indeed, once you know its length, it is worth specifying its value. In order to do so, we usually rely on random number generators, and in particular on their cryptographically secure representatives. For block ciphers, the output of those CSPRNGs can directly be used as a key. In the case of asymmetric systems, the answer of the generator is at the basis of a defined procedure to generate all required parameters. However, those keys are usually accessed with some flavour of password or pin code. A novel approach to secure those keys is to conceal them using intrinsic randomness occurring in the fabrication process of physical objects. This is achieved thanks to fuzzy extractors that allow turning error prone measurement into keys that are usable in cryptography. We applied this technique to extract fingerprint from sheets of paper secured by embedded ultra-violet fibers and use them to recover a randomly generated unique identifier. Moreover, this identifier can be coupled with the information being printed on the paper to form a message that is signed and then printed on the paper sheet. In this way, the authenticity of the document as a whole can be checked and some attacks like unauthorized copy and modification of the payload can be prevented. Any fields where paper is used as the medium to transport information can benefit from the proposed modular method. Moreover, we believe our proposal can be widely extended to almost any physical source that either embeds its own intrinsic randomness or is modified to include one (maybe like Pappu suggested for his authentication system [Pap]). More generally, I believe many applications may benefit from fuzzy extractors.

Bibliography

- [Gal] Galois Inc. Cryptol: a language for writing specifications for cryptographic algorithms. http://www.galois.com/technology/communications_security/cryptol.
- [GB] Damien Giry and Philippe Bulens. Cryptographic Key Length Recommendation. <http://www.keylength.com>.
- [Imp] Impulse Accelerated Technologies. ImpulseC: a software-to-hardware compiler. <http://www.impulseaccelerated.com/>.
- [LV99] Arjen K. Lenstra and Eric R. Verheul. Selecting Cryptographic Key Sizes. *Journal of Cryptology*, 14:255–293, 1999.
- [Pap] R. Pappu. Physical One-Way Functions. <http://alumni.media.mit.edu/~pappu/pdfs/Pappu-PhD-POWF-2001.pdf>. Ph. D. dissertation, MIT, 2001.
- [CRb] Certicom Research. SEC 2: Recommended Elliptic Curve Domain Parameters, v1.0. <http://www.secg.org>.
- [Sho94] P. W. Shor. Algorithms for Quantum Computation: Discrete Logarithms and Factoring. *Proc. 35th Annual Symposium on Foundations of Computer Science*, pages 124–134, IEEE Computer Society Press 1994.

Publication list

- ✎ Philippe Bulens, François-Xavier Standaert and Jean-Jacques Quisquater, *How to Strongly Link Data and its Medium: – the Paper Case –*, Submitted to IET Information Security on March 2009.
- ✎ François-Xavier Standaert, Philippe Bulens, Giacomo de Meulenaer and Nicolas Veyrat-Charvillon, *Improving the Rules of the DPA Contest*, IACR eprint archive: <http://eprint.iacr.org/2008/517>.
- ✎ Chong Hee Kim, Philippe Bulens, Christophe Petit and Jean-Jacques Quisquater, *Fault Attacks on Public Key Elements: Application to DLP based Schemes*, Fifth European PKI Workshop — EUROPKI'08, Lecture Notes in Computer Science 5057:182–195, Springer, June 2008.
- ✎ Philippe Bulens, François-Xavier Standaert, Jean-Jacques Quisquater, Pascal Pellegrin and Gaël Rouvroy, *Implementation of the AES-128 on Virtex-5 FPGAs*, Progress in Cryptology — AfricaCrypt'08, Lecture Notes in Computer Science 5023:16–26, Springer, June 2008.
- ✎ Gueric Meurice de Dormale, Philippe Bulens and Jean-Jacques Quisquater, *Collision Search for Elliptic Curve Discrete Logarithm over $GF(2^m)$ with FPGA*, Workshop on Cryptographic Hardware and Embedded Systems — CHES'07, Lecture Notes in Computer Science 4727:378–393, Springer, September 2007.
- ✎ Philippe Bulens, Kassem Kalach, François-Xavier Standaert and Jean-Jacques Quisquater, *FPGA Implementations of eSTREAM Phase-2 Focus Candidates with Hardware Profile*, The State of the Art of Stream Ciphers — SASC'07, January 2007, <http://sasc.crypto.rub.de/>.
- ✎ Philippe Bulens, Gueric Meurice de Dormale and Jean-Jacques Quisquater, *Hardware for Collision Search on Elliptic Curve over $GF(2^m)$* , Special-purpose Hardware for Attacking Cryptographic Systems — SHARCS'06, April 2006.

- ✎ Guerric Meurice de Dormale, Philippe Bulens and Jean-Jacques Quisquater, *An Improved Montgomery Modular Inversion Targeted for Efficient Implementation on FPGA*, International Conference on Field-Programmable Technology — FPT'04, pp. 441–444, December 2004.
- ✎ Guerric Meurice de Dormale, Philippe Bulens and Jean-Jacques Quisquater, *Efficient Modular Division Implementation –ECC over $GF(p)$ Affine Coordinates Application–*, The 14th International Conference on Field Programmable Logic and Applications — FPL'04, Lecture Notes in Computer Science 3203:231–240, Springer, September 2004.

Modular Division

Elliptic Curve Public Key Cryptosystems (ECPKC) are becoming increasingly popular for use in mobile appliances where bandwidth and chip area are strongly constrained. For the same level of security, ECPKC use much smaller key length than the commonly used RSA. The underlying operation of affine coordinates elliptic curve point multiplication requires modular multiplication, division/inversion and addition/subtraction. To avoid the critical division/inversion operation, other coordinate systems may be chosen, but this implies more operations and a strong increase in memory requirements. So, in area and memory constrained devices, affine coordinates should be preferred, especially over $GF(p)$.

This chapter presents a powerful reconfigurable hardware implementation of the Takagi modular divider algorithm. The resulting 256-bit circuits achieved a ratio throughput/area improved by at least 900 % of the only known design in Xilinx Virtex-E technology. Comparison with typical modular multiplication performance is carried out to suggest the use of affine coordinates also for speed reason.

A.1 Introduction

Modular arithmetic plays an important role in cryptographic systems. In mobile appliances, very efficient implementations are needed to meet the cost constraints while preserving good computing performances. In this field, modular multiplication has attracted great attention through different proposals: Montgomery multiplication, Quisquater algorithm, Brickell method and some others. The modular inversion problem has also been extensively studied. It can be performed using the well-known Euclid algorithm (or any of the binary variants like the Montgomery inverse [SK00, Kal95]), Fermat little theorem or the recently GCD-free method [JP03].

The modular division is believed to be slow and has not attracted a lot of attention because it can be replaced by a modular inversion followed by a modular multiplication. Despite their slowness, these operations are needed in several cases: when creating public-private key pairs for RSA and when deciphering in an ElGamal cryptosystem. Although, the main bottleneck arises when we talk about Elliptic Curve Cryptosystems (ECC).

This chapter is structured as follows: section A.2 reminds the theoretical bases of ECC over $GF(p)$ and the different coordinate systems. We introduce the rewritten algorithm

and the reason why it has been chosen in section A.3. The main contribution lies in section A.4 where we present our implementation. The opportunity of special adders is discussed and a pipelined architecture is described. Practical results and comparisons with the only known published design are in section A.5. A typical modular division design is also introduced, suggesting the use of affine coordinates for their memory and area requirements as much as their computational time. Finally, section A.6 concludes the article.

A.2 Elliptic Curve Operations over $\text{GF}(p)$

An elliptic curve E over $\text{GF}(p)$, with p a prime number, is defined as the set of points (x, y) verifying the reduced Weierstraß equation:

$$E : f(X, Y) \triangleq Y^2 - X^3 - aX - b \equiv 0 \pmod{p}$$

for $a, b \in \text{GF}(p)$, each choice of these parameters leading to a different curve. Such a curve is called “non-singular” if its discriminant is different from 0 (this corresponds to three distinct roots). The condition is then rewritten as $4a^3 + 27b^2 \not\equiv 0 \pmod{p}$.

In ECC, the data to be encrypted is represented by a point P on a chosen curve. The encipherment by the key k is performed by computing $Q = P + P + \dots + P = kP$. This operation, called scalar multiplication, is usually achieved through the “double and add” method (the adaptation of the well-known “square and multiply” to elliptic curves).

A.2.1 Point Addition and Doubling in Affine Coordinate

In most cases, the addition of points $P = (x_P, y_P)$ and $Q = (x_Q, y_Q)$ leads to the resulting point $R = (x_R, y_R)$ through the following computation:

$$\begin{cases} x_R = \lambda^2 - x_P - x_Q \pmod{p} \\ y_R = -y_P + \lambda(x_P - x_R) \pmod{p} \end{cases}, \lambda = \begin{cases} \frac{y_P - y_Q}{x_P - x_Q} \pmod{p} & \text{if } P \neq Q \\ \frac{3x_P^2 + a}{2y_P} \pmod{p} & \text{if } P = Q \end{cases}$$

In some cases, exceptions may arise. When we try to add P to its inverse $-P = (x_P, -y_P \pmod{p})$, there is an obvious problem in the computation of λ . This is theoretically tackled by the definition of the point at infinity $\mathcal{O}$. The result is defined as $P + (-P) = \mathcal{O} \Leftrightarrow P + \mathcal{O} = P$. This means that $\mathcal{O}$ is the identity element. Another issue is the doubling of P when it lies on the x -axis, i. e. when its y -coordinate is 0. As might be expected, the result is the point at infinity.

A.2.2 Using other Coordinate Systems

In the computation of $Q = kP$, the “double and add” method requires on average $(m - 1)$ doubling steps and $(m - 1)/2$ addition steps according to the Hamming weight of k , where m is the bit length of k . For each of these steps, we need to know the value λ . It can be computed by a modular division or a modular inversion followed by a modular

multiplication. As those operations are believed to be slow, one may prefer representing points in another coordinate system [CMO98].

Those kinds of systems allow point addition and point doubling with only multiplications and additions. Inversion is no more intertwined in the elliptic curve operations, but we still need it to convert the final result back to affine representation.

Table A.1 contains the timings in modified jacobian¹ and in affine coordinates, where M denotes the time for a modular multiplication, S for a squaring and I for the inversion. $I + M$ can be replaced by a modular division D .

	Doubling	Addition
modified jacobian	$t(2\mathcal{J}^m) = 4M + 4S$	$t(\mathcal{J}^m + \mathcal{J}^m) = 13M + 6S$
affine	$t(2\mathcal{A}) = 2M + 2S + I$	$t(\mathcal{A} + \mathcal{A}) = 2M + S + I$

Table A.1: Timings of different coordinate systems

It is now possible to issue the upper bound for the ratio I/M under which the affine coordinates are theoretically more useful. We extract it by asking the average timings for scalar multiplication in affine to be quicker than in jacobian coordinates.

$$\begin{aligned}
 (m-1) \cdot t(2\mathcal{J}^m) + \frac{m-1}{2} \cdot t(\mathcal{J}^m + \mathcal{J}^m) &\geq (m-1) \cdot t(2\mathcal{A}) + \frac{m-1}{2} \cdot t(\mathcal{A} + \mathcal{A}) \\
 2 \cdot (4M + 4S) + 13M + 6S &\geq 2 \cdot (2M + 2S + I) + 2M + S + I \\
 8 &\geq I/M
 \end{aligned}$$

The last result is obtained assuming a square is performed using the same circuitry as a multiplication. The same kind of criteria may be adapted to the ratio D/M . In this case, we reach $9 \geq D/M$. Of course, those timing considerations must be weighted by their area and memory requirements.

A.3 Algorithm

All the algorithms practically used for modular division are based on the Extended Binary GCD. The main problem is that a comparison is generally needed in order to determine the next operations to compute. For this reason, some efforts have been made to speed up the operation [CS01], but we believe that the most clever idea is due to the work of Takagi [Tak98], based on the plus-minus algorithm of Brent and Kung [BK83]. He reaches the goal of avoiding comparison by replacing it with the inspection of Least Significant Bit (LSB) from shift registers and variables.

Takagi's algorithm is rewritten in algorithm 2. For a m -bit modulus M , it takes between $m + 4$ and $2m + 4$ clock cycles to perform the loading of the operands, the modular division and the last correction step. The operations between brackets are performed in parallel.

The presented algorithm is slightly different from the original. The initialization step has been duplicated (STEP 1BIS) and a reset step (STEP 3) has been added before the

¹We choose this system since it is the fastest at point doubling.

Algorithm 2 Algorithm for modular division computation.

Inputs: $2^{n-1} < M < 2^n$; $-M < X, Y < M$

Output: $Z = X/Y \bmod M$

STEP 1:

$A \leftarrow 0, B \leftarrow Y, U \leftarrow 0, V \leftarrow X, P \leftarrow n, D \leftarrow 0$

STEP 1BIS:

$A \leftarrow A + B, B \leftarrow M, U \leftarrow U + V, V \leftarrow 0$

STEP 2:

while $P \geq 0$ **do**

if $[a_1 a_0] = 0$ **then** $A \leftarrow A/4, U \leftarrow MQRTR(U, M)$

if $D < 2$ **then**

if $D = 1$ **then** $P \leftarrow P - 1$

else $P \leftarrow P - 2$

$D \leftarrow D - 2$

elseif $a_0 = 0$ **then** $A \leftarrow A/2, U \leftarrow MHLV(U, M)$

if $D < 1$ **then** $P \leftarrow P - 1$

$D \leftarrow D - 1$

else

if $([a_1 a_0] + [b_1 b_0]) \bmod 4 = 0$ **then** $q \leftarrow 1$ **else** $q \leftarrow -1$

if $D \geq 0$ **then** $A \leftarrow (A + qB)/4, U \leftarrow MQRTR(U + qV, M)$

if $D = 0$ **then** $P \leftarrow P - 1$

$D \leftarrow D - 1$

else $D \leftarrow -D - 1, \{A \leftarrow (A + qB)/4, B \leftarrow A\}$

$\{U \leftarrow MQRTR(U + qV, M), V \leftarrow U\}$

STEP 3:

$U \leftarrow 0$

STEP 4:

if $[b_1 b_0] \bmod 4 = 3$ **then**

if $V \geq 0$ **then** $Z \leftarrow U - V + M$ **else** $Z \leftarrow U - V$

else

if $V \geq 0$ **then** $Z \leftarrow U + V$ **else** $Z \leftarrow U + V + M$

correction step (STEP 4) to spare resources within the targeted devices. The extra clock cycles added are negligible with respect to the global executing time.

Instead of using redundant binary representation like Takagi, we decided to use classical binary representation. Indeed, we want to focus on the smallest area requirements and the redundancy roughly twice the amount of hardware needed in the FPGA. With this choice, the timings of the P and D shift registers used in the control part are less critical. So, we decided to replace those registers by small counters, like suggested in [Tak98]. Their sizes are in $\log_2(\log_2(M))$, so the propagation delay is very short. Another fact in aid of counters is that shift registers with different directions and steps consume a lot of resources on FPGA. Finally, counters also enable a slight reduction of the algorithm complexity, leading to a more efficient control structure.

The main feature of this algorithm is the use of P and D counters. The comparison between A and B in commonly used algorithms is replaced by $D = \alpha - \beta$, where α and β are values such that 2^α and 2^β represent the minimums of the upper bounds of $|A|$ and $|B|$ respectively. This substitution reduces the comparison between A and B in the “while” loop to “ $|A| > 0$ ”. Instead of investigating all the bits of A , it is replaced by a counter P with sign detection, indicating the minimum of the upper bounds of $|A|$ and $|B|$.

The halving operation $MHLV(T, M)$ of $T \bmod M$ is performed either by $T/2$ or $(T+M)/2$ regarding the parity of T (the LSB). The quartering operation $MQRTR(T, M)$ of $T \bmod M$ depends on the value of $M \bmod 4$. If it equals 1, then the operations to be carried out are $T/4$, $(T - M)/4$, $(T + 2M)/4$ or $(T + M)/4$, accordingly as $T \bmod M$ is 0, 1, 2 or 3. If $M \bmod 4$ equals 3, the operations are $T/4$, $(T + M)/4$, $(T + 2M)/4$ or $(T - M)/4$ with respect to the value of $T \bmod M$, respectively 0, 1, 2 or 3.

It should finally be noticed that, when using binary GCD, a division does not slow down the computation compared to the inversion, since U or V are not used in the control part².

A.4 Implementation

In this section, we present two different kinds of implementation. We first introduce the basic sequential architecture and the opportunity of flags precomputation. After, we present an improvement of this architecture by tackling the critical path: the carry chain of the adders. We will show that the best compromise lies in the use of pipelining instead of carry conditional and select adder.

A.4.1 Basic Division Architecture

We present here the basic sequential architecture. It is naturally broken up into different distinct parts: the *ControlStage*, the *ABstage*, the *UVstage* and the P, D counters.

² U is set to 1 when performing an inversion, to X otherwise.

The ControlStage:

The main advantage of the algorithm we use lies in the absence of comparison, serially wired with the main operative part. Fortunately, this improvement implies only a small complexity increase with the use of P and D counters. In the main loop, all the flags only depend on parity bits of variables and on sign of small counters. So, all the flags can be efficiently precomputed, leading to high working frequency.

The ABstage:

The ABstage operative part is shown in Fig. A.1. It consumes mainly 3 Logic Elements (LE, half of a slice) on FPGA for each bit of the modulus. One is used for the two's complement adder/subtractor, another is used for the shift right selection and the last one is used for the loading and the swap of registers.

Since the loading step has been duplicated, we can spare one multiplexor, implemented by one LE per bit length, between the logical shifter and the register.

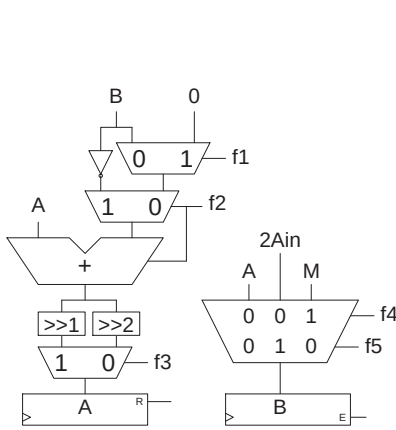


Figure A.1: stage AB

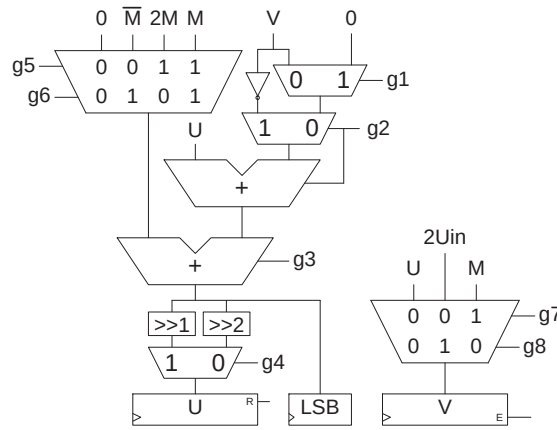


Figure A.2: stage UV

The UVstage:

The UVstage operative part is shown in Fig. A.2. This stage mainly consumes 4 Logic Elements per bit length. Two are used for two's complement adder/subtractor (M is considered constant), one is used for the shift right selection and the last one is used for the loading and the swap of registers.

We also spare a multiplexor thanks to the duplication of the loading step. The reset step (STEP 3) of the U signal enable resource sharing for the last correction step (STEP 4). However, we always right shift the input of the U register. The shifted bit must be saved in one register to provide the final result.

We should notice that the $g3$, $g5$ and $g6$ flags depend on the 2 LSB computed by the first adder. An efficient solution is to replicate this 2-bit adder to allow flags precomputation in the control part.

The P and D counters:

The P counter is based on an adder and is negative allowed (Fig. A.3). The end criteria can be checked with only the sign bit. The D counter is also adder based (Fig. A.4). We introduced two additional adders which always compute $D - 1$ and $D - 2$ to compute all the tests needed in the algorithm.

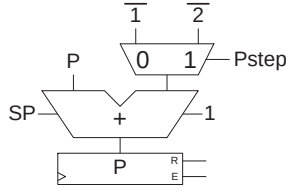


Figure A.3: counter P

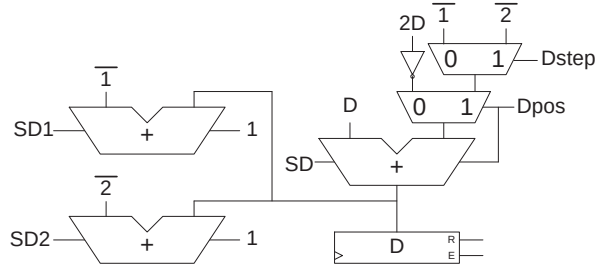


Figure A.4: counter D

A.4.2 Special Adder

ECC need computation over important length modulus (typically 160 bits). Area constrained modular division designs are adder based, so it is obvious that the critical path resides in the carry chain of the simple carry-propagate adders.

Modern FPGA are built to efficiently implement the addition operation. High speed performance is achieved by the use of a dedicated optimized carry chain. While FPGAs are structured as an array of programmable logic and routing resources, the carry chain is physically wired and therefore exhibits very low delays. It can be used without constraining the routing as long as the carry chain size does not exceed the column height. For this purpose, adders must be adapted to keep satisfactory speed performance.

This optimized carry chain explains why well known methods like carry-look-ahead, carry-bypass, carry conditional, carry-select and carry-save are practically not really attractive on area constrained reconfigurable devices. Of course, some of them can speed up work frequency, but it is at the cost of a lot of chip area.

Another solution is the pipelining of the addition. It is simply achieved by inserting register in the carry chain and by properly handling the operands. This way, almost no additional hardware is consumed. The drawback is the number of clock cycles needed to fill the pipeline. Nevertheless, this overhead can be negligible if the number of repeated additions is great with respect to the number of pipelined stages. These advantages lead us to choose this method for our improved design.

A.4.3 Pipelined Division Architecture

In order to speed up the computation, we choose the pipelined architecture. As said in the previous section, this method requires small area increase and can lead to interesting results. It can be used because the flags of the control part are only LSB for the main loop.

MSB are only required for the last correction step. So, the total overhead will be twice the number of pipeline stages added. This is negligible compared to the number of clock cycles required for the whole computation.

Number of pipeline stages:

The number of pipeline stages must still be determined. Putting the overhead aside, we cannot cut the adders into a lot of small parts. This is due to the behaviour of the algorithm: the two right shifts impose that the LSB of the next stage must be available for the MSB of the current stage. Because the computation is achieved in one clock cycle, we must bring back the LSB asynchronously. This increases the number of Logic Elements through the path leading to the computation of the carry-out.

This new design will be quite place & route dependent. So, it is not easy to theoretically determine the best number of pipeline stages. Nevertheless, approximation can be made to decide after how many bits the carry chain must be cut. After inspecting Fig. A.5 and Fig. A.6, the carry chain propagation time must be shorter than the delay of the U and V operands arrival plus the carry-in arrival and three serial Logic Element (two for the adders and one for the conditional shifter).

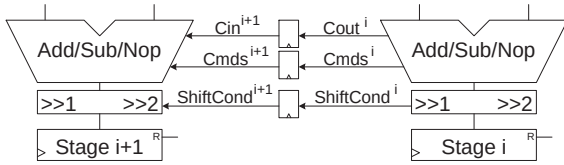


Figure A.5: Typical pipeline organisation

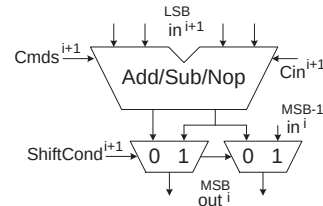


Figure A.6: Modification of the 2 MSB input of each stage

Architecture:

To improve readability, all stage features are not reproduced in Fig. A.5. The main concern is the conveyance of the shifted bits between pipelined stages. Indeed, the 2 MSB input of each stage (except the last) must be modified with the circuit shown in Fig. A.6. The two LSB adders and the conditional shifter have been duplicated to avoid the two serial delays of carry-in arrival and sum return. These two delays appear when we simply take the output from the next pipeline stage adder. With this negligible increase of logic element, we expect to only suffer from the operand arrival delay. All these modifications ensure a faster working design, with only a small increase of hardware and clock cycles.

A.5 Results

Speed and area comparison of 64, 128, 160 and 256-bit basic and pipelined designs are presented in table A.2. The VHDL synthesis and place & route have been achieved on

Xilinx ISE 6.2.02i. The first device selected is a Xilinx Virtex-E XCV2000e-6bg560 FPGA (V-E) to guarantee fair comparison with the design presented in [DMKP03]. The second is a Xilinx Spartan3 XC3S200-4pq208 (S3) to exhibit performances over a small and low cost FPGA. The two operands are loaded by two 32-bit clocked interface and stored in 32-bit shift registers. The result is registered and then moved in 32-bit shift registers to access the 32-bit clocked output interface.

Design	Area V-E (Slices)	Freq. V-E (MHz)	Thr./Area (kbit/s. Slices)	Area S3 (Slices)	Freq. S3 (MHz)	Thr./Area (kbit/s. Slices)
64-bit	420 (2 %)	77	88.8	424 (22 %)	77	88
32-bit $\times 2$	460 (2 %)	83	86.1	461 (23 %)	83	86
128-bit	778 (4 %)	55	34.8	873 (45 %)	48	27
32-bit $\times 4$	842 (4 %)	75	42.8	927 (48 %)	79	41
160-bit	951 (4 %)	45	23.3	1112 (57 %)	44	19.5
32-bit $\times 5$	1022 (5 %)	77	36.3	1180 (61 %)	78	31.8
256-bit	1457 (7 %)	29	9.8	1728 (90 %)	29	7.5
32-bit $\times 8$	1612 (8 %)	77	23	1847 (96 %)	80	20.9

Table A.2: FPGA implementation results

As explained in section A.4.3, the pipelined stages must have a minimal length to be interesting. We decided to pipeline 32-bit adders. The pipelining leads to great improvements, especially for the 256-bit basic design, where the carry chain length is too big for the selected FPGA column height.

Throughput/area ratio comparison between our best design and the best design of [DMKP03] is given in table A.3. To compute the throughput, we consider the worst case of $(2m - 1)$ clock cycles for their design and $(2m + 4) + 2 ps$ for our design, where ps is the number of pipeline stages. Unfortunately, they do not give the 160-bit design results.

Design	Freq (MHz)	Throughput (Mbit/s)	Area (Slices)	Thr./area (kbit/s. Slices)	Improvement
64-bit	45	22.7	1212	18.7	
Our 64-bit	83	39.6	460	86.1	360 %
128-bit	31	15.6	2215	7	
Our 128-bit	75	36	842	42.8	511 %
Our 160-bit	77	37	1022	36.3	
256-bit	27	13.53	5846	2.3	
Our 256-bit	77	37	1612	23	900 %

Table A.3: Performance comparisons

As previously said, it is interesting to compare our modular division implementation with modular multiplication in terms of design performances. We take the 120-bit architecture presented in [DM02] for its reasonable area requirements.

To enable fair comparison, we implemented our design over the same Virtex1000. Our 128-bit design exhibits a frequency of 74 Mhz and an area of 852 Slices but requires, in the worst case, twice as many clock cycles. So, we can roughly say that, with a multiplier work frequency of 88.5 Mhz and an area of 603 Slices, we reached a ratio D/M of 2.5. This result is more than three times as good as the threshold ratio of section A.2.2.

A.6 Conclusion

A powerful modular division implementation has been presented. It has minimal hardware requirements and high work frequency. In addition to the accurate description of our architecture, we discussed the opportunity to use special adder structures (e.g. carry-save, carry-look-ahead) and exhibit that a pipelined carry-propagate adder seems to be the best choice for an area constrained FPGA implementation of the modular division.

We achieved a throughput/area ratio of 23 kbit/(s.Slices) for a 256-bit design. It represents an improvement by at least 900 % of the only known design in Xilinx Virtex-E technology. Using our implementation, affine coordinates for ECC over GF(p) seem to be attractive for their memory and area requirements but also for speed reason. This suggests the use of our design in Elliptic Curve cryptoprocessor of embedded devices.

Bibliography

- [BK83] R. P. Brent and H. T. Kung. Systolic VLSI arrays for linear time GCD computation. *VLSI*, 83:145–154, 1983.
- [CMO98] H. Cohen, A. Miyaji, and T. Ono. Efficient Elliptic Curve Exponentiation using Mixed Coordinates. *Advances in Cryptology — ASIACRYPT’98*, LNCS 1423:51–65, 1998.
- [CS01] S. Chang-Shantz. From Euclid’s GCD to Montgomery Multiplication to the Great Divide. *Technical report*, Sun Microsystems Laboratories TR-2001-95, June 2001.
- [DM02] A. Daly and W. Marnane. Efficient architectures for implementing montgomery modular multiplication and RSA modular exponentiation on reconfigurable logic. *Proceedings of the 2002 ACM/SIGDA tenth international symposium on Field-programmable gate arrays*, 2002.
- [DMKP03] A. Daly, W. Marnane, T. Kerins, and E. Popovici. Fast Modular Division for Application in ECC on Reconfigurable Logic. *The 13th International Conference on Field Programmable Logic and Applications — FPL’03*, September 2003.
- [DMP03] A. Daly, L. Marnane, and E. Popovici. Fast Modular Inversion in the Montgomery Domain on Reconfigurable Logic. *Irish Signals and Systems Conference — ISSC03*, July 2003.
- [GTK02] A. A.-A. Gutub, A. F. Tenca, and Ç. K. Koç. Scalable VLSI Architecture for $GF(p)$ Montgomery Modular Inverse Computation. *IEEE Computer Society Annual Symposium on VLSI*, pages 53–58, April 2002.
- [GTSK02] A. A.-A. Gutub, A. F. Tenca, E. Savaş, and Ç. K. Koç. Scalable and Unified Hardware to Compute Montgomery Inverse in $GF(p)$ and $GF(2^n)$. *Workshop on Cryptographic Hardware and Embedded Systems — CHES02*, Lecture Notes in Computer Science 2523:484–499, Augustus 2002.
- [Has01] M. A. Hassan. Efficient Computation of Multiplicative Inverses for Cryptographic Applications. *Technical Report CORR 2001–03*, Center for Applied Cryptographic Research, 2001.

- [JP03] M. Joye and P. Paillier. GCD-free Algorithms for Computing Modular Inverses. *Workshop on Cryptographic Hardware and Embedded Systems — CHES03*, Lecture Notes in Computer Science 2779:243–253, 2003.
- [Kal95] B. S. Kaliski Jr. The Montgomery Inverse and its Applications. *IEEE Transactions on Computers*, 44(8):1064–1065, August 1995.
- [Knu81] D. E. Knuth. *The Art of Computer Programming, vol. 2 : Seminumerical Algorithms*, 2nd ed. Addison-Wesley, 1981.
- [KT03] M. E. Kahaira and N. Takagi. A VLSI Algorithm for Modular Multiplication/Division. *The 16th IEEE Symposium on Computer Arithmetic — ARITH 16*, pages 15–18, June 2003.
- [MR00] M. Mascagni and S. Rahimi. Parallel Inversive Congruential Generators: Software and Field-Programmable Gate Array Implementations. *International Conference on Monte Carlo Simulation*, June 2000.
- [NS03] M. Naseer and E. Savaş. Hardware Implementation of a Novel Inversion Algorithm. *The 46th IEEE Midwest Symposium on Circuits and Systems*, pages 27–31, December 2003.
- [Rab96] J. M. Rabaey. *Digital integrated circuits: a design perspective*. Prentice-Hall, 1996.
- [SK00] E. Savaş and Ç. K. Koç. The Montgomery Modular Inverse - Revisited. *IEEE Transactions on Computers*, 49(7):763–766, July 2000.
- [SK02] E. Savaş and Ç. K. Koç. Architectures for Unified Field Inversion with Applications in Elliptic Curve Cryptography. *The 9th IEEE International Conference on Electronics, Circuits and Systems — ICECS02*, pages 1155–1158, September 2002.
- [SOOS95] R. Schroepel, H. Orman, S. O’Malley, and O. Spatscheck. Fast Exchange with Elliptic Curve Systems. *Advances in Cryptology — CRYPTO’95*, LNCS 973:43–56, 1995.
- [Tak98] N. Takagi. A VLSI Algorithm for Modular Division Based on the Binary GCD Algorithm. *IEICE Trans. Fundamentals of Electronics, Communications and Computer Sciences*, E81-A(5):724–728, May 1998.

Montgomery Modular Inversion

Modular multiplication and inversion/division are the most common primitives in today's public key cryptography. Elliptic Curve Public Key Cryptosystems (ECPKC) are becoming increasingly popular for use in mobile appliances where bandwidth and chip area are strongly constrained. For the same level of security, ECPKC use much smaller key length than the commonly used RSA but need modular inversion/division.

This chapter presents an improved algorithm for prime field Montgomery modular inversion. The first important contribution lies in the reduction of the number of operations needed. Resource sharing is also used to lighten the control part of the algorithm. The second contribution is the minimization of the set of different instructions to enable powerful FPGA implementations. Resulting 256-bit circuit achieves a ratio throughput/area improved by at least 70% compared to the only known Montgomery inverse design in FPGA technology.

Though the implementations are first oriented towards FPGA, some improvements are generic. So, they could prove to be also efficient for ASIC designs in terms of area and power consumption.

B.1 Introduction

Modular arithmetic plays an important role in public key cryptographic systems. In mobile appliances, very efficient implementations are needed to meet the cost constraint while preserving good computing performance. In this field, modular multiplication has received great attention and numerous papers have been published. The modular inversion problem has also been extensively studied. It can be performed using Fermat little theorem or the well-known extended Euclidian algorithm (or any of the binary variants like the Montgomery inverse [Kal95, SK00]). The modular division is believed to be slow and can be replaced by a modular inversion followed by a modular multiplication.

Despite their slowness and their cost, inversion or division are needed in several cases: when creating public-private key pairs for RSA or when deciphering in an ElGamal cryptosystem. Although, the main bottleneck arises in the context of Elliptic Curve Cryptosystems (ECC). In this chapter, we will focus on this topic and more precisely over $\text{GF}(p)$.

Despite the fact that modular inversion has been extensively studied, not many FPGA implementations have been published. For the Montgomery modular inverse, we found only one paper [DMP03], based on the binary extended Euclidian algorithm.

The interest for the developed algorithmic modifications are twofold. First, we reduce the number of operations needed and second, we reduce the set of possible operations in order to meet efficiency in selected hardware devices.

This chapter is structured as follows: section B.2 reminds the theoretical bases of ECC over $GF(p)$. Section B.3 describes some constraints of the selected hardware devices needed to understand the algorithmic modification choices. We introduce the Montgomery inverse algorithm, the modified algorithm and its implementation in section B.4. Practical results and comparisons with the only known published design lie in section B.5. Finally, section B.6 concludes the chapter.

B.2 Elliptic Curve Operations over $GF(p)$

An elliptic curve E over $GF(p)$, with p a prime number, is defined as the set of points (x, y) verifying the reduced Weierstraß equation:

$$E : f(X, Y) \triangleq Y^2 - X^3 - aX - b \equiv 0 \pmod{p}.$$

In ECC, the data to be encrypted is represented by a point P on a chosen curve. The encipherment by the key k is performed by computing $Q = P + P + \dots + P = kP$. This operation, called scalar multiplication, is usually achieved through the *double and add* method (the adaptation of the well-known *square and multiply* to elliptic curves).

When the resulting point is not the point at infinity $\mathcal{O}$, the addition of points $P = (x_P, y_P)$ and $Q = (x_Q, y_Q)$ leads to the resulting point $R = (x_R, y_R)$ through the following computation:

$$\begin{cases} x_R &= \lambda^2 - x_P - x_Q \pmod{p} \\ y_R &= -y_P + \lambda(x_P - x_R) \pmod{p} \end{cases}$$

where

$$\lambda = \begin{cases} \frac{y_P - y_Q}{x_P - x_Q} \pmod{p} & \text{if } P \neq Q \\ \frac{3x_P^2 + a}{2y_P} \pmod{p} & \text{if } P = Q \end{cases}$$

It should be noticed that it is also possible to represent points in other coordinate systems. The reader can find an interesting overview in [CMO98].

B.3 FPGA Constraints

ECC need computation over important bit-length modulus (typically 160 bits). The inversion algorithm used in this chapter is based on adder, so it is obvious that the critical path resides in the carry chain of the simple carry-propagate adders.

Modern FPGAs are built to efficiently implement the addition operation. High speed performance is achieved by the use of a dedicated optimized carry chain. While FPGAs are structured as an array of programmable logic and routing resources, the carry chain is

physically wired and therefore exhibits very low delays. It can be used without constraining the routing as long as the carry chain size does not exceed the column height. For this purpose, adders must be adapted to keep satisfactory speed performance.

In the algorithms we deal with, the control logic uses the result of a magnitude comparison, preventing techniques like pipelining. Experience shows that intricate adder structure on FPGA leads to poor speed/area results. So, we decided to choose a simple carry conditional adder [Rab96] to break the carry chain exceeding the column height.

We decided to move towards a Xilinx Virtex or Spartan FPGA for our design. For this kind of device, a simplified Logical Element (LE), equivalent to half a slice, is illustrated in figure B.1. For this kind of architecture, another constraint is that 4-input Look-Up Table (LUT) in Adder mode can not use combinatorial circuit with one of the adder input. Indeed, with this kind of carry chain, we must have almost one input directly connected to the serial carry chain multiplexor to enable the full adder function. In figure B.1, for example, we can only apply a combinatorial process with the LUT on the A input. So, if we want to spare as much FPGA resources as possible, we must take care in the algorithmic choices of always keeping an input identical to the adders.

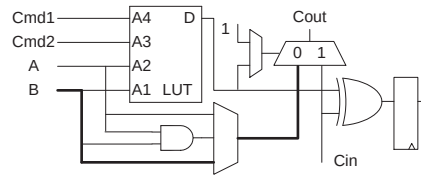


Figure B.1: A simplified Xilinx Virtex Logical Element

The first constraint is implementation specific while the last constraint has been taken into account all along the algorithmic modifications. It will explicitly come to light in the modified algorithms presented below.

B.4 The Montgomery Modular Inverse

In this section, we will discuss about integer Montgomery modular inverse. First, we introduce the original algorithm and the Montgomery notation. Then, we present our modified algorithm with motivations about the choices made. The implementation details end this section.

B.4.1 Montgomery Modular Inverse Algorithm

The Montgomery modular inverse was first introduced by Kaliski in 1995 [Kal95]. It is defined as the Montgomery representation of the modular inverse, i.e. $X^{-1}2^n \bmod M$, where n is the bit-length of M . It is based on the extended binary GCD algorithm. This algorithm takes place in two phases: the first one, presented in algorithm 3, outputs a partial result $X^{-1}2^k \bmod M$ with $n \leq k \leq 2n$, called the *almost Montgomery inverse* (introduced in [SK00]). The second part may differ according to the desired output do-

main: Montgomery domain or real integer. This result can be reached either by iterative halving modulo M or multiplication modulo M .

Algorithm 3 Kaliski Montgomery inverse algorithm, Phase 1

Input: $X \in [1, M - 1]$ and M

Output: $R \in [1, M - 1]$ and k , where $R = X^{-1}2^k \bmod M$ and $n \leq k \leq 2n$

```

 $U \leftarrow M, V \leftarrow X, R \leftarrow 0, S \leftarrow 1, k \leftarrow 0$ 
while  $V > 0$  do
  if  $U$  even then  $U \leftarrow U/2, S \leftarrow 2S$ 
  elsif  $V$  even then  $V \leftarrow V/2, R \leftarrow 2R$ 
  elsif  $U > V$  then  $U \leftarrow (U - V)/2, R \leftarrow R + S, S \leftarrow 2S$ 
  elsif  $V \geq U$  then  $V \leftarrow (V - U)/2, R \leftarrow S + R, R \leftarrow 2R$ 
   $k \leftarrow k + 1$ 
if  $R \geq M$  then  $R \leftarrow R - M$ 
return  $R \leftarrow M - R$  and  $k$ 

```

During Phase 1, the following invariants can be verified by induction:

$$\begin{aligned} XR &\equiv -U 2^k \bmod M \\ M &= US + VR \end{aligned}$$

At the end of the while loop, the values of $U = 1$ and $V = 0$ allow checking that $R = -X^{-1}2^k \bmod M$, which is then corrected to bring the result back in the range $[1, M - 1]$. Moreover, the register containing S will end the first phase with the value of the prime modulus M . Also, it has been proved [Kal95] that the number of operations needed to complete algorithm's Phase 1 is at least n and at most $2n$ cycles if X and M are relatively prime.

B.4.2 Modified Montgomery Modular Inverse Algorithm

The modified Montgomery modular inverse is presented in algorithm 4. It is clearly hardware oriented to enable an easy transposition towards the FPGA implementation. The UV variables are allowed to be negative and the two's complement representation has been chosen. In the following, $\text{sign}(T)$ stands for the sign of T , which is defined as the bit sign in two's complement representation. The absolute value of T will be expressed as $\text{abs}(T)$.

The initialization phase has been extended to meet area efficiency with the FPGA. In fact, the original load of the X operand in the register Luv means that we need a multiplexor between the adder and the memory element, wasting one LE per input bit. The added clock cycle is negligible with respect to the global running time of the algorithm.

The first major improvement relies on the removal of the $U - V$ or $V - U$ operation. Indeed, the subtraction needed for the magnitude comparison is reused for the $U = (U - V)/2$ or $V = (V - U)/2$ operation. We simply allow results to be negative and we take care of their signs to perform the appropriate next operation. The output sign of the

Algorithm 4 Modified Montgomery inverse algorithm, Phase 1 and 2

Input: $X \in [1, M - 1]$ and M **Output:** $Lrs \in [1, M - 1]$, where $Lrs = X^{-1}2^n \bmod M$

PHASE 1 $k \leftarrow -\text{bitsize}(M)$, $Luv \leftarrow 0$, $Ruv \leftarrow 2X$, $Lrs \leftarrow 0$, $Rrs \leftarrow 1$ $Luv \leftarrow Luv/2 + Ruv$, $Ruv \leftarrow M$, $Lrs \leftarrow Lrs + Rrs$, $Rrs \leftarrow 0$ **while true do** $SLuv \leftarrow \text{sign}(Luv)$, $SRuv \leftarrow \text{sign}(Ruv)$ **if** $Luv/2$ even **then** **if** $SLuv = \text{sign}(-Luv)$ **then** break while loop $Luv \leftarrow Luv/2$, $Rrs \leftarrow 2Rrs$, $k \leftarrow k + 1$ **else** $tmpuv \leftarrow Luv/2$, $tmprs \leftarrow Lrs$, $Lrs \leftarrow Lrs + Rrs$ **if** $(SLuv \text{ xor } SRuv) = 1$ **then** $Luv \leftarrow Luv/2 + Ruv$ **else** $Luv \leftarrow Luv/2 - Ruv$ $k \leftarrow k + 1$, $ctrl \leftarrow (\overline{SLuv} \text{ and } \overline{SRuv}) \text{ or } (\overline{SLuv} \text{ and } SRuv)$ **if** $\text{sign}(Luv) = ctrl$ **then** $Ruv \leftarrow tmpuv$, $Rrs \leftarrow 2tmprs$ **else** $Rrs \leftarrow 2Rrs$ /* thanks to the invariant, we have $Lrs = M *$ */ $Lrs \leftarrow Lrs - Rrs$, $Rrs \leftarrow M$ **if** $\text{sign}(Lrs) = 1$ **then** $Lrs \leftarrow Lrs + Rrs$ PHASE 2**while** $k \neq 0$ **do** $k \leftarrow k - 1$ **if** Lrs even **then** $Lrs \leftarrow Lrs/2$ **else** $Lrs \leftarrow (Lrs + Rrs)/2$ **return** Lrs

current $Luv + Ruv$ or $Luv - Ruv$ operation also tells which register must be updated (if a switch of the variables must be achieved or not).

The second major modification is based on the reusing of the Luv adder at the time of *simple Luv bitshift* to verify the $Luv = 0$ end criterion. Instead of using a *not so negligible* zero comparison, we check whether the stored Luv sign is equal to the $-Luv$ sign computed with the right adder. With two's complement representation, this equality means that $Luv = 0$.

For the last operation of the Phase 1, $U = 1$ and $V = 0$. Fortunately, the algorithm invariant tells us that S is equal to the prime modulus M . We can then directly execute the $M - R$ operation needed to output the correct result.

The bitshift of the Luv variable is performed after its register to enable the use of the memory element directly available after the adder in the chosen FPGA.

We decided to include in the modified algorithm a phase 2 based on *iterative halving modulo M* to convert the almost inverse to the Montgomery domain. The algorithm can be trivially updated to eliminate this conversion or transform the almost inverse to the real inverse.

It must be noticed that a different but related *sign technique* has been used in the recent paper [NS03].

We will show in the next subsection that all these modifications lead to an efficient FPGA implementation.

B.4.3 Montgomery Modular Inverse Implementation

We begin with general observations of the implementation before inspecting specific features of the design.

At first glance, the implementation of the UV stage should be like the one shown in figure B.2. This solution, more related to ASIC, exhibits a real algorithm improvement with the removal of one adder, but it is not optimal for FPGA implementation: the multiplexors between the adder and the memory element impose the use of extra LE. A better solution, presented in figure B.3, can be achieved if we load the input variable in the right register and allow the *add 0* operation in the left register. The left adder can no longer be used to check the end criterion, so we convert the right multiplexor towards an adder for the purpose of computing $-Luv$. This implies no additional cost for selected FPGA when we use a simple carry ripple adder.

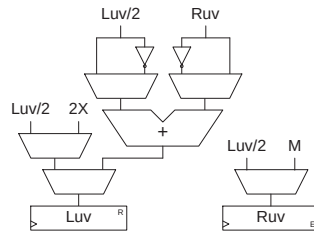


Figure B.2: Early implementation for UV stage, more ASIC related

The asynchronous reset of the Luv register is used to enable its load of the X variable on the next clock cycle. This variable must be left-loaded because the modulus is always odd and the *left add right* operation can only be performed in the left part.

The RS stage execution is delayed by one clock cycle with respect to the UV stage execution. This is motivated by the fact that the switch determined by the equality $\text{sign}(\text{Luv}) = \text{ctrl}$ is serially wired with a carry chain and constitutes the critical path. Moreover, this control signal has a large fanout, which increases the delay even more. These considerations also apply for the end criterion $\text{SLuv} = \text{sign}(-\text{Luv})$.

LE used for the multiplexor between the Lrs adder and its memory element only serves as a bitshifter to perform the Phase 2 of the algorithm. If only the almost Montgomery inverse is needed, we can simply remove it and then spare more area and control logic.

As explained above, no extra load of the constant M into the Lrs register is needed to perform the last Phase 1 operation.

Basic Adder Implementation

Figures B.3 and B.4 illustrate the main two stages of the basic adder. The small k counter is not shown. The two stages consume 5 LE per bit, but 1 LE can be removed if only Phase 1 must be performed.

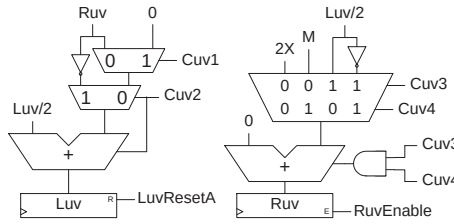


Figure B.3: Montgomery inverse operative part of the basic UV stage

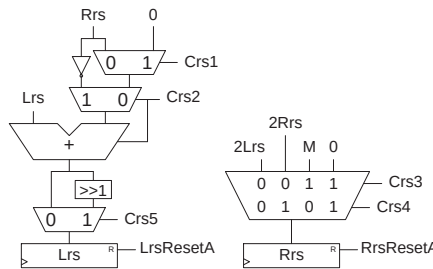


Figure B.4: Montgomery inverse operative part of the basic RS stage

Carry Conditional Implementation

Because carry chain can be too high for the FPGA, it can be necessary to use other type of adders to improve performances.

This modified implementation exhibits no difference with the basic adder design except that we replace carry ripple adders with carry conditional adders. However, the Ruv adder does not need all the traditional features of a carry conditional adder. As explained before, the only time it is used for the adder purpose, we only need the sign bit. So, it is more useful to replace this most significant part by a zero comparator. This can be efficiently achieved using LE as 4-input AND gates and the carry chain to process 4 Luv bits by LE.

B.5 Implementation Results

The modified algorithm has been implemented to provide speed and area comparison of 64, 128, 160 and 256-bit basic and carry conditional adder designs. The VHDL synthesis and place & route have been achieved on Xilinx ISE 6.2.02i. The first device selected is a Xilinx Virtex-E XCV2000e-6bg560 FPGA to guarantee fair comparison with the Daly, Marnane and Popovici design [DMP03]. The second is a Xilinx Spartan3 XC3S200-5tq144 to exhibit performances over a small and low cost FPGA. The operand is loaded by a 32-bit clocked interface and stored in 32-bit shift registers. The result is also sent to 32-bit shift registers to access the 32-bit clocked output interface.

The tables B.1 show implementation results for the two selected devices.

VirtexE Spartan3 Design	Frequency (MHz)		Area (Slices)		Thr./Area (kbit/s.Slices)	
64-bit	67	70	302	295	72.5	77.0
32-bit $\times 2$	69	77	394	425	56.9	58.9
128-bit	46	49	558	562	27.2	28.7
64-bit $\times 2$	55	63	723	806	25.0	25.7
160-bit	38	42	687	684	18.2	20.3
80-bit $\times 2$	50	59	890	971	18.5	20.0
256-bit	25	29	1057	1063	7.8	9.0
128-bit $\times 2$	41	41	1390	1490	9.8	9.1

Table B.1: Virtex-E and Spartan3 implementation results

It can be seen that carry conditional adder designs exhibit a better throughput/area only when the carry chain exceeds the height of the selected FPGA. Indeed, the area increase cannot be compensated by small frequency increase. The column height of the selected Virtex-E is 80 CLB per column (and 2 Slices per CLB) while the selected Spartan3 is 20 CLB per column (and 4 Slices per CLB). So, when we reached the 160-bit limit, carry conditional design begin to be attractive.

The throughput/area ratio comparison between our best implementation and the Daly, Marnane and Popovici best implementation is given in the table B.2. To compute the throughput, we consider the worst case of $(3n + 3)$ clock cycles (Phase 1 and 2) for their

design and $(3n + 5)$ for our design. Unfortunately, they do not give the 160-bit design results.

VirtexE Design	Freq. (Mhz)	Area (Slices)	Thr./area (kbit/s.Slices)	Improv.
64-bit	59.93	515	38.3	
Our 64-bit	67	302	72.5	89 %
128-bit	46.80	1023	15.2	
Our 128-bit	46	558	27.2	79 %
Our 160-bit	50	890	18.5	
256-bit	34.73	2022	5.7	
Our 256-bit	41	1390	9.8	72 %

Table B.2: Performance comparisons

B.6 Conclusion

We presented an improved algorithm for prime field Montgomery modular inversion. Those improvements took place in two main phases: first we reduced the number of operations needed in the algorithm and second, we rewrote the operations to take into account the targeted device constraints. To validate the approach, we implemented them in two different FPGA devices.

We achieved a throughput/area ratio of 9.8 kbit/(s.Slices) for a 256-bit Virtex-E design. It represents an improvement by at least 70% of the only known Montgomery inverse design on FPGA technology. We also produced results for Spartan3 FPGA to exhibit performance over a small and low cost FPGA device. Results can be even more improved if only the almost Montgomery inverse is needed.

Though the modified algorithms are first oriented towards FPGA, some improvements are generic. Further work could use the proposed algorithms to issue efficient designs for ASIC in terms of area and power consumption.

Bibliography

- [BK83] R. P. Brent and H. T. Kung. Systolic VLSI arrays for linear time GCD computation. *VLSI*, 83:145–154, 1983.
- [CMO98] H. Cohen, A. Miyaji, and T. Ono. Efficient Elliptic Curve Exponentiation using Mixed Coordinates. *Advances in Cryptology — ASIACRYPT’98*, LNCS 1423:51–65, 1998.
- [CS01] S. Chang-Shantz. From Euclid’s GCD to Montgomery Multiplication to the Great Divide. *Technical report*, Sun Microsystems Laboratories TR-2001-95, June 2001.
- [DM02] A. Daly and W. Marnane. Efficient architectures for implementing montgomery modular multiplication and RSA modular exponentiation on reconfigurable logic. *Proceedings of the 2002 ACM/SIGDA tenth international symposium on Field-programmable gate arrays*, 2002.
- [DMKP03] A. Daly, W. Marnane, T. Kerins, and E. Popovici. Fast Modular Division for Application in ECC on Reconfigurable Logic. *The 13th International Conference on Field Programmable Logic and Applications — FPL’03*, September 2003.
- [DMP03] A. Daly, L. Marnane, and E. Popovici. Fast Modular Inversion in the Montgomery Domain on Reconfigurable Logic. *Irish Signals and Systems Conference — ISSC03*, July 2003.
- [GTK02] A. A.-A. Gutub, A. F. Tenca, and Ç. K. Koç. Scalable VLSI Architecture for $GF(p)$ Montgomery Modular Inverse Computation. *IEEE Computer Society Annual Symposium on VLSI*, pages 53–58, April 2002.
- [GTSK02] A. A.-A. Gutub, A. F. Tenca, E. Savaş, and Ç. K. Koç. Scalable and Unified Hardware to Compute Montgomery Inverse in $GF(p)$ and $GF(2^n)$. *Workshop on Cryptographic Hardware and Embedded Systems — CHES02*, Lecture Notes in Computer Science 2523:484–499, Augustus 2002.
- [Has01] M. A. Hassan. Efficient Computation of Multiplicative Inverses for Cryptographic Applications. *Technical Report CORR 2001–03*, Center for Applied Cryptographic Research, 2001.

- [JP03] M. Joye and P. Paillier. GCD-free Algorithms for Computing Modular Inverses. *Workshop on Cryptographic Hardware and Embedded Systems — CHES03*, Lecture Notes in Computer Science 2779:243–253, 2003.
- [Kal95] B. S. Kaliski Jr. The Montgomery Inverse and its Applications. *IEEE Transactions on Computers*, 44(8):1064–1065, August 1995.
- [Knu81] D. E. Knuth. *The Art of Computer Programming, vol. 2 : Seminumerical Algorithms*, 2nd ed. Addison-Wesley, 1981.
- [KT03] M. E. Kahaira and N. Takagi. A VLSI Algorithm for Modular Multiplication/Division. *The 16th IEEE Symposium on Computer Arithmetic — ARITH 16*, pages 15–18, June 2003.
- [MR00] M. Mascagni and S. Rahimi. Parallel Inversive Congruential Generators: Software and Field-Programmable Gate Array Implementations. *International Conference on Monte Carlo Simulation*, June 2000.
- [NS03] M. Naseer and E. Savaş. Hardware Implementation of a Novel Inversion Algorithm. *The 46th IEEE Midwest Symposium on Circuits and Systems*, pages 27–31, December 2003.
- [Rab96] J. M. Rabaey. *Digital integrated circuits: a design perspective*. Prentice-Hall, 1996.
- [SK00] E. Savaş and Ç. K. Koç. The Montgomery Modular Inverse - Revisited. *IEEE Transactions on Computers*, 49(7):763–766, July 2000.
- [SK02] E. Savaş and Ç. K. Koç. Architectures for Unified Field Inversion with Applications in Elliptic Curve Cryptography. *The 9th IEEE International Conference on Electronics, Circuits and Systems — ICECS02*, pages 1155–1158, September 2002.
- [SOOS95] R. Schroepel, H. Orman, S. O’Malley, and O. Spatscheck. Fast Exchange with Elliptic Curve Systems. *Advances in Cryptology — CRYPTO’95*, LNCS 973:43–56, 1995.
- [Tak98] N. Takagi. A VLSI Algorithm for Modular Division Based on the Binary GCD Algorithm. *IEICE Trans. Fundamentals of Electronics, Communications and Computer Sciences*, E81-A(5):724–728, May 1998.

Stream Ciphers on FPGA

Efficient cryptographic implementations are a fundamental factor in the achievement and dissemination of new computerized applications. In some recent environments with (very) limited resources such as smart cards, sensor networks or RFID tags, standard algorithms may not be completely adapted. Consequently, the design of new solutions for low-cost cryptography is sometimes necessary and is at least an interesting research direction. In this context, stream ciphers are usually believed to be an efficient alternative to block ciphers, because of a lower hardware (or even software) implementation cost. The eSTREAM research initiative was consequently established in order to investigate the possibility of building secure and efficient stream ciphers. Among a large number of submitted candidates, the project recently (September 2006) narrowed the “hardware profiled” stream ciphers to four focused candidates, namely Trivium, Grain-128, MICKEY-128 2.0 and Phelix. In this chapter, we evaluate the hardware performance of these algorithms in the reconfigurable hardware Xilinx Virtex-II devices. Based on our implementations (that mainly confirm previous results), we discuss the respective interest of the focused candidates and suggest certain guidelines for their comparison.

C.1 Introduction

Stream ciphers are an important class of symmetric encryption algorithms. their internal states vary with time and generate pseudo-random bit sequences, usually denoted as the keystream. The keystream is then bitwise XORed with a message in order to emulate the one-time-pad. By contrast, block ciphers tend to simultaneously encrypt/decrypt blocks of bits and act as a pseudo-random permutation of a fixed size, parametrized by a secret key. Although they received less attention than standard block ciphers in the recent years, stream ciphers are more appropriate in certain contexts, *e.g.* where buffering is limited or when characters must be individually processed. Another interesting property is that the keystream can be generated prior to encryption and decryption when it is independent of the plaintext/ciphertext. In addition, it is also generally assumed that stream ciphers have a lower implementation cost and therefore are a sound solution for very constrained environments. Due to the proliferation of applications requiring low-cost cryptographic algorithms, *e.g.* smart cards, sensor networks or RFID tags, they appear as an interesting

research direction¹.

In this context, the eSTREAM initiative [ESCP] was launched as an effort to identify new stream ciphers that might be considered for widespread adoption. Since most known (old) stream ciphers have been the target of (more or less realistic) cryptanalytic attacks, a most important objective of the project was the design of a secure cipher and the derivation of sound (easily checkable) security criteria. As a reply to the eSTREAM call-for-papers, numerous algorithms have been submitted. They have been divided into three categories, namely software profiled, hardware profiled or suitable for both. Although the security evaluation of the different candidates is still running, a selection of focused candidates was posted on the ECRYPT page in September 2006. As for the Advanced Encryption Standard process, an important criterion for the discrimination of the remaining candidates is the performance evaluation.

In this chapter, we consequently investigated the performances of the four ciphers selected as *Phase 2 Focus Candidate for Hardware Profile*, namely Grain-128 [HJMM06], Trivium [CP06], Mickey-128 2.0 [BD06] and Phelix [WSLM05]. The different ciphers will be shortly described in Section C.2. Section C.3 details our design methodology, our target platform for the evaluation as well as our architectures for the different ciphers. The implementation results are summarized in Section C.4, together with a comparison/discussion between different stream and block ciphers from a hardware performance point of view and some guidelines for the discrimination of the four eSTREAM candidates. Finally, our conclusions are in Section C.5.

C.2 Ciphers Description

In general, stream ciphers consist of two phases: an initialization phase in which the initial state of the cipher is derived from the key and initial vector (IV) and a generation phase in which the state is repeatedly updated and used to generate the keystream (*e.g.* via some non-linear function). In the next section a brief functional description of the different eSTREAM ciphers we consider in this chapter will be given. For a more insight view of those ciphers, please refer to the original papers [BD05, BD06, CP05, CP06, HJM05, HJMM06, WSLM05].

C.2.1 Trivium

Trivium is a synchronous stream cipher designed to provide a flexible trade-off between speed and gate count. It has one of the simplest architecture amongst the eSTREAM candidates and is consequently particularly easy to implement. Trivium supports an 80-bit secret key and an 80-bit IV. It generates up to 2^{64} bits of keystream. Trivium's 288-bit internal state may be seen as a concatenation of three shift registers of different lengths. The state is updated using an iterative process which extracts the values of 15 specific state bits and uses them to both update 3 bits of the state and compute 1 bit of keystream. The state is then updated and the process is repeated until the requested $N \leq 2^{64}$ bits of

¹Note that the division stream cipher *vs.* block cipher may appear somewhat arbitrary since a block cipher can be turned into a stream cipher if used in Output Feedback Mode or Cipher Feedback Mode.

keystream are generated. To initialize the cipher, the key and IV are written into two of the shift registers and the remaining bits are set to a fixed pattern of zeros and ones. The state is then updated over 4 full cycles (meaning 4×288 updates), in the same way as explained above, but without generating keystream bits. This initial step guarantees that every bit of the internal state depends on every bit of the key and of the IV in a complex nonlinear way.

C.2.2 Grain-128

Grain is a synchronous stream cipher targeted for hardware environments where gate count, power consumption and memory are limited. It supports a key size of 128 bits and IV size of 96 bits. Grain's 256-bit internal state consists of a 128-bit linear feedback shift register (LFSR) and a 128-bit non-linear feedback shift register (NFSR). The state is updated thanks to a nonlinear filter function that takes two inputs from the NFSR and seven from the LFSR. The NFSR is updated with a nonlinear 19-to-1 boolean function and a 1-bit linear input selected from the LFSR. The LFSR is updated with a 6-to-1 linear function. Before keystream is generated, the cipher must be initialized as follows: the 128-bit key is loaded into the 128-bit NFSR, the 96-bit IV is loaded into the low 96-bits of the LFSR and the remaining 32 high bits of the LFSR are filled with ones. The cipher is then clocked 256 times without producing any keystream. Instead, the output function is fed back and XORed with the input, both to the NFSR and LFSR update functions.

C.2.3 MICKEY-128 2.0

Just like Grain-128 and Trivium, MICKEY-128 2.0 is based on shift registers. The content of the two 160-bit registers defines the state of the cipher. It uses a fixed length 128-bit key and a variable length initialization vector (ranging from 0 to 128 bits). At startup, the two shift registers are set to "0". Then, the IV is used to fill the registers, followed by the key. The cipher is then clocked 160 times before producing any keystream bit. According to the authors, up to 2^{64} bits may be generated using a single (IV,key) pair. The main difference with Grain-128 and Trivium is that each single bit of the state of MICKEY-128 2.0 is computed, not just shifted (see Sec. C.3.4). This allows replacing the use of a filter function by a simple XOR of two state bits to compute the keystream.

C.2.4 Phelix

Phelix is an enhanced version of Helix. Although it is presented as a stream cipher, it looks very much like a block cipher. It operates on a set of nine 32-bit words at a time, five of which being "active", that cycle through the Phelix structure. The basic operations involved are XORs, additions (mod 2^{32}) and rotations, allowing a fast execution on any kind of platform. Phelix works differently than the previously mentioned ciphers. For example, the first step is to derive the working key from the initial key (256-bit) and expand the nonce (128-bit) to its full size (256-bit). Those expanded words are then used to initialize and pre-clock the cipher before the keystream generation begins. Another difference is that the plaintext is incorporated during the computation. This gives the

cipher the additional feature of a message authentication code (MAC) that can be derived at the cost of a fixed number of additional loops in the structure.

C.3 Hardware Implementation

C.3.1 Methodology

Since the selected designs are purposed for low-cost devices, we adopted a bit-serial style of interface for the inputs and outputs (plaintext, key, IV, ciphertext). Similarly, throughput was generally sacrificed in favour of area requirements. For example, we did not take advantage of the possible parallelization tricks of Trivium and Grain-128 that would increase the throughput over area ratio. Consequently, our implementations are not necessarily the most efficient ones and we mainly aimed to evaluate the possibility to obtain low-cost architectures.

C.3.2 Targeted Platform

The chosen platform for implementation is a Xilinx Virtex-II FPGA. Such devices embed programmable logic blocks, RAMs and multipliers. In this description, the focus will be set on the logic elements as other resources will not be used. The slice is the logic unit that is used to evaluate a design's area requirements. Such a slice, depicted in Fig. C.1 is made up of 2 Look-Up Tables (LUTs), 2 flip flops and a few additional gates. According to the user's choice, any of these LUTs can be configured in one out of three possible ways: RAM16 that acts like a 16-bit RAM storage, SRL16 that implements a 16-bit linear shift register and LUT that is capable of computing any 4-to-1 boolean function. A more detailed view of half a slice is given in Fig. C.2. An interesting thing to notice is the fast carry chain that crosses the slice. As its designation suggests, it allows the efficient implementation of carry-propagate adders and, as will be shown, can sometimes be used to simplify the combinatorial cost of some designs.

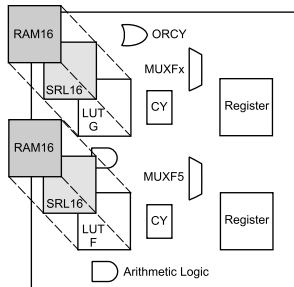


Figure C.1: Available slice configurations.

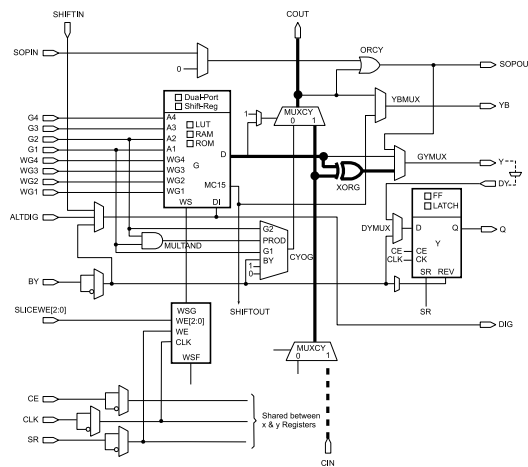


Figure C.2: Top half slice of a Xilinx Virtex-II.

C.3.3 Trivium and Grain-128

A straightforward approach was adopted to implement Grain and Trivium. We only used the platform-specific primitives SRL16 for the shift registers that provide an advantageous area optimization over a platform-independent design. It is worth noticing that the efficient exploitation of the SRL16 primitives highly depends on the required tap positions in the stream ciphers. Any time an intermediate value is extracted from the SRL16 primitives, a new LUT has to be used. As a consequence, a k bit register with $k < 16$ may occupy a complete SRL16 cell. Note also that this approach has the disadvantage of being less portable. The rest of our code is directly derived from the algorithm's specifications. Due to the simplicity of the operative parts in our designs, the cost of the control parts (*i.e.* state machines and counters) is not negligible.

C.3.4 MICKEY-128 2.0

The description of MICKEY-128 2.0 is somewhat software oriented. We consequently re-wrote the update functions of the R and S registers as follows:

- **CLOCK_R :**

- if $i == 0$: $r_0 \leftarrow (r_0 \cdot \mathbf{CR}) \oplus r_{159} \oplus \mathbf{IR}$
- if $i \in [1..159]$:

$$r_i \leftarrow \begin{cases} r_{i-1} \oplus (r_i \cdot \mathbf{CR}) \oplus r_{159} \oplus \mathbf{IR} & \text{if } i \in \text{RTAPS} \\ r_{i-1} \oplus (r_i \cdot \mathbf{CR}) & \text{if } i \notin \text{RTAPS} \end{cases}$$

- **CLOCK_S :**

- if $i == 0$: $s_0 \leftarrow s_{159} \oplus \mathbf{IS}$
- if $i \in [1..158]$:

$$s_i \leftarrow \begin{cases} s_{i-1} \oplus (\overline{s_i'} \cdot \overline{s_{i+1}}') & \text{if } \mathbf{FB0}_i \parallel \mathbf{FB1}_i == 00 \\ s_{i-1} \oplus (\overline{s_i'} \cdot \overline{s_{i+1}}') \oplus ((s_{159} \oplus \mathbf{IS}) \cdot \mathbf{CS}) & \text{if } \mathbf{FB0}_i \parallel \mathbf{FB1}_i == 01 \\ s_{i-1} \oplus (\overline{s_i'} \cdot \overline{s_{i+1}}') \oplus ((s_{159} \oplus \mathbf{IS}) \cdot \overline{\mathbf{CS}}) & \text{if } \mathbf{FB0}_i \parallel \mathbf{FB1}_i == 10 \\ s_{i-1} \oplus (\overline{s_i'} \cdot \overline{s_{i+1}}') \oplus s_{159} \oplus \mathbf{IS} & \text{if } \mathbf{FB0}_i \parallel \mathbf{FB1}_i == 11 \end{cases}$$

where $(\overline{s_i'} \cdot \overline{s_{i+1}}')$ is a shorthand for $((s_i \oplus \mathbf{COMP0}_i) \cdot (s_{i+1} \oplus \mathbf{COMP1}_i))$. The tabulated values of $\mathbf{COMP0}_i$ and $\mathbf{COMP1}_i$ allow to determine whether the $\overline{}$ operator has to apply².

- if $i == 159$: $s_{159} \leftarrow s_{158} \oplus ((s_{159} \oplus \mathbf{IS}) \cdot \overline{\mathbf{CS}})$

As stated in Sec. C.3.2, each LUT is capable of implementing any 4-to-1 boolean function. Fortunately, the update functions of the R and S registers of MICKEY-128 2.0 mainly use XOR operations, *i.e.* one of the additional gate available for each LUT. By

²For example: let $i == 2$, then $s_i \leftarrow s_{i-1} \oplus (\overline{s_i'} \cdot s_{i+1}) \oplus ((s_{159} \oplus \mathbf{IS}) \cdot \overline{\mathbf{CS}})$.

setting either r_{159} or s_{159} on the carry-in pad of the slice, elements r_i and s_i requiring 5 inputs to be computed will fit in half a slice (see emphasized path in Fig. C.2). For the remaining cases of s_i parameterized by $FB0_i \neq FB1_i$ ³, the same trick can be used in order to fill a single slice. In extenso, the first LUT computes $((s_{159} \oplus IS) \cdot \overline{CS})'$; the second, $s_{i-1} \oplus (\overline{s_i'} \cdot \overline{s_{i+1}}')$ and an additional XOR is used to complete the evaluation. Each of these bits consumes 2 LUTs.

This implementation can slightly be improved: if a group of 3 s_i with same $FB0_i \parallel FB1_i$ is stacked within 2 slices, the computation of $((s_{159} \oplus IS) \cdot \overline{CS})'$ – taking place in one LUT – can be broadcasted to the upper 3 LUTs through the fast carry-chain. The increase in area for this part is now limited to a third of the 80 LUTs required to store this part of S . It is tempting to extend this idea to a group of 7 s_i stacked within 4 slices. However, this is likely to decrease the performance by increasing the complexity of the routing as the local availability of the data is reduced. Note that, as opposed to Trivium and Grain-128, each single bit of the state has to be computed which prevents benefiting from the SRL16 configured slices of the device. As a result, MICKEY-128 2.0 is expected to consume more area in our target FPGAs.

C.3.5 Phelix

The implementation of Phelix is quite straightforward. The designer is just left with the choice of the tradeoff best suiting his needs: a “big” design with a high throughput by implementing the whole Phelix block structure or something smaller, down to the “tiny” single round. From the previous implemented designs in [GCB06], the best trade-off seems to be the one based on the H function (half the block structure) which exhibits the highest throughput over area. Therefore, we selected this configuration. In order to implement the buffer storing Z_4^{i-1} to Z_4^{i-4} , the use of SRL16-configured LUTs again allows reducing the required space from 64 slices (128 registers) to 16 slices. In addition, the 512 memory points required to remember the expanded key and nonce words can be efficiently stored in LUTs configured as RAM (RAM16). This consumes 32 LUTs, that is 16 slices. Despite these efforts, the implementation seems to stay a little more expensive in area than presented in [GCB06], which may be caused by a different interface.

C.4 Implementation Results

The four selected ciphers were implemented in VHDL. Synthesis and Place & Route were achieved on Xilinx ISE 8.2i. The selected device is a Xilinx Virtex-II XC2V6000-4ff1152. Table C.1 summarizes our results whereas table C.2 reminds some results of previous selected works.

Compared to the previous implementations of the same ciphers, the situation has not changed much. The specifications of Trivium and Phelix are the same as for eSTREAM *Phase 1* and consequently, they exhibit quite similar figures compared to the results in Table C.2. In the case of Grain-128, the revision that, amongst others, has grown the state

³About half the length of the register S ; that is around 80 bits.

Cipher	Device	Slices	Frequency (MHz)	Throughput (Mbps)	Thr./Area (Mbps/slice)
Trivium	Virtex-II	41	207	207	5.05
Grain-128	Virtex-II	48	181	181	3.77
MICKEY-128 2.0	Virtex-II	190	200	200	1.05
Phelix	Virtex-II	1213	62.5	1000	0.82

Table C.1: Implementation results.

from 2×80 -bit to 2×128 -bit goes unnoticed in terms of FPGA hardware cost. Regarding MICKEY-128 2.0, the updated version has slightly increased its cost but keeps it reasonable. It should be noticed that the gap between MICKEY-128 2.0 on the one hand and Trivium and Grain-128 on the other hand could be reduced if ASIC implementations were considered. Indeed, MICKEY-128 2.0 does not benefit from SRL16-configured slices that Trivium and Grain-128 use abundantly.

Cipher	Device	Slices	Frequency (MHz)	Throughput (Mbps)	Thr./Area (Mbps/slice)
Stream Ciphers					
A5/1 [GKKK04]	Virtex-II	32	–	188.3	5.88
Trivium [GCB06]	Spartan-II	40	–	102	2.55
Grain [GCB06]	Spartan-II	48	–	105	2.19
RC4 [GKKK04]*	Virtex-II	140	–	120.8	0.86*
MICKEY-128 [Kit]	Virtex-E	167	170	170	1.02
Phelix [GCB06]	Spartan-II	1077	–	750	0.7
Block Ciphers					
AES [GB05]*	Spartan-II	124	–	2.2	0.02*
AES [RSQL04]*	Virtex-II	146	123	358	2.45*
AES [ZNC04]	Virtex-II	1780	77.91	1000	0.56

Table C.2: Previous block and stream ciphers implementations.

Compared to the implementations of other stream ciphers, a number of additional observations can be mentioned. For illustration purposes, we also reported some implementation results for the AES Rijndael block cipher. Most interestingly, these performance evaluations highlight that Trivium and Grain-128 have extremely reduced implementation complexities that could be compared, *e.g.* to those of A5/1. MICKEY-128 2.0 has a slightly higher hardware cost in our target Xilinx FPGA. As already mentioned, this trend would probably be tempered if ASIC designs were considered. Finally, it seems that the performances of Phelix are better compared to those of a block cipher, as its structure also suggests. For example, the AES architecture in [ZNC04] iterates over a 128-bit width loop structure while the one in [RSQL04] uses a 32-bit datapath. Phelix ranges somewhere close to these results. Note that these observations have to be consid-

ered as general intuitions rather than fair comparisons since they consider different FPGA technologies: more recent FPGAs have higher work frequencies and thus throughput. In addition, the hardware efficiency (*i.e.* throughput/area ratio) of the *-marked implementations is not meaningful since they consume FPGA RAM blocks. Finally, the hardware cost can only be compared if the respective implementation efficiencies (*e.g.* measured with the throughput/area ratio) are somewhat comparable. As a matter of fact, it is always possible to reduce the implementation cost, by considering smaller datapaths (*e.g.* [GB05] uses an 8-bit datapath for the AES) at the cost of a reduced throughput.

To strengthen these observations, let us finally mention that Trivium and Grain-128 offer the best opportunities to improve their throughput by parallelization techniques. In the case of Trivium (*resp.* Grain-128), all state bits are left unused for at least 64 (*resp.* 16) iterations after they have been modified. This way, up to 64 (*resp.* 16) iterations can be computed at once, at some additional combinatorial cost. For example, [GLB⁺06] shows that a 64-bit Trivium is 54% larger and has a maximum clock frequency that is 10% lower than a 1-bit implementation.

C.5 Conclusion

This chapter reports updated results on the FPGA implementations of the eSTREAM Phase-2 Focus Candidates with Hardware Profile. Compared to Phase 1, only the specifications of Grain-128 and MICKEY-128 2.0 have been slightly modified. We re-implemented Trivium and Phelix for comparison purposes. These results suggest that Trivium and Grain-128 have the smallest complexities for a hardware implementation. Since these ciphers also provide the best opportunities to improve their throughput by exploiting parallelization techniques, the strongest cryptanalytic efforts should concentrate on these ciphers. Additional efforts could also be devoted to an ASIC implementation of MICKEY-128 2.0 since its structure is not perfectly suitable for FPGA implementations. Finally, Phelix appears as an interesting alternative among the implemented ciphers, due to its significantly different design principles.

Bibliography

- [Avo05] G. Avoine. *Cryptography in Radio Frequency Identification and Fair Exchange Protocols*. PhD thesis, EPFL, Lausanne, Switzerland, December 2005.
- [BD05] S. Babbage and M. Dodd. The stream cipher MICKEY-128 (version 1). eSTREAM, ECRYPT Stream Cipher Project, Report 2005/016, 2005. <http://www.ecrypt.eu.org/stream>.
- [BD06] S. Babbage and M. Dodd. The stream cipher MICKEY-128 2.0. eSTREAM, ECRYPT Stream Cipher Project, 2006. <http://www.ecrypt.eu.org/stream>.
- [CP05] C. De Cannière and B. Preneel. Trivium - A Stream Cipher Construction Inspired by Block Cipher Design Principles. eSTREAM, ECRYPT Stream Cipher Project, Report 2005/030, 2005. <http://www.ecrypt.eu.org/stream>.
- [CP06] C. De Cannière and B. Preneel. Trivium Specifications. eSTREAM, ECRYPT Stream Cipher Project, 2006. <http://www.ecrypt.eu.org/stream>.
- [ESCP] The eSTREAM web site. eSTREAM, ECRYPT Stream Cipher Project. <http://www.ecrypt.eu.org/stream>.
- [FDW04] M. Feldhofer, S. Dominikus, and J. Wolkerstorfer. Strong Authentication for RFID Systems Using the AES Algorithms. *Workshop on Cryptographic Hardware and Embedded Systems — CHES'04*, Lecture Notes in Computer Science 3156:357 – 370, 2004.
- [Fin03] K. Finkenzeller. *RFID Handbook: Fundamentals and Applications in Contactless Smart Cards and Identification, Second Edition*. John Wiley & Sons, Ltd., 2003.
- [GB05] M. Good and M. Benaissa. AES on FPGA from the Fastest to the Smallest. *Workshop on Cryptographic Hardware and Embedded Systems — CHES'05*, Lecture Notes in Computer Science 3659:427 – 440, 2005.

- [GCB06] T. Good, W. Chelton, and M. Benaissa. Review of stream cipher candidates from a low resource hardware perspective. eSTREAM, ECRYPT Stream Cipher Project, Report 2006/016, 2006. <http://www.ecrypt.eu.org/stream>.
- [GKKK04] M. D. Galanis, P. Kitsos, G. Kostopoulos, and O. Koufopavlou. Comparison of the Performance of Stream Ciphers for Wireless Communications. In *International Conference on Computing, Communications and Control Technologies — CCCT'04*, Austin, Texas, USA, August 14 – 17 2004.
- [GL06] F. K. Gürkaynak and P. Luethi. Recommendations for Hardware Evaluation of Cryptographic Algorithms. eSTREAM, ECRYPT Stream Cipher Project, Report 2006/028, 2006. <http://www.ecrypt.eu.org/stream>.
- [GLB⁺06] F. K. Gürkaynak, P. Luethi, N. Bernold, R. Blattmann, V. Goode, M. Marghitola, H. Kaeslin, N. Felber, and W. Fichtner. Hardware Evaluation of eSTREAM Candidates: Achterbahn, Grain, MICKEY, MOSQUITO, SFINKS, Trivium, VEST, ZK-Crypt. eSTREAM, ECRYPT Stream Cipher Project, Report 2006/015, 2006. <http://www.ecrypt.eu.org/stream>.
- [HJM05] M. Hell, T. Johansson, and W. Meier. Grain - A Stream Cipher for Constrained Environments. eSTREAM, ECRYPT Stream Cipher Project, Report 2005/010, 2005. <http://www.ecrypt.eu.org/stream>.
- [HJMM06] M. Hell, T. Johansson, W. Meier, and A. Maximov. A Stream Cipher Proposal: Grain-128. eSTREAM, ECRYPT Stream Cipher Project, 2006. <http://www.ecrypt.eu.org/stream>.
- [IPC] IP Cores. <http://http://ipcores.com/index.htm>.
- [JTS05] K. Järvinen, M. Tommiska, and J. Skyttä. Comparative survey of high-performance cryptographic algorithm implementations on FPGAs. *IEE Proceedings on Information Security*, 152(1):3–12, 2005.
- [Kit] P. Kitsos. On the Hardware Implementation of the MICKEY-128 Stream Cipher. eSTREAM, ECRYPT Stream Cipher Project, Report 2006/059. <http://www.ecrypt.eu.org/stream>.
- [LSRQ05] P. Léglise, F.-X. Standaert, G. Rouvroy, and J.-J. Quisquater. Efficient Implementation of Recent Stream Ciphers on Reconfigurable Hardware Devices. *26th Symposium on Information Theory in the Benelux*, pages 261–268, 2005.
- [RSQL04] G. Rouvroy, F.-X. Standaert, J.-J. Quisquater, and J.-D. Legat. Compact and Efficient Encryption/Decryption Module for FPGA Implementation of the AES Rijndael Very Well Suited for Small Embedded Applications. *International Symposium on Information Technology: Coding and Computing — ITCC'04*, pages 583 – 587, 2004.

-
- [WSLM05] D. Whiting, B. Schneier, S. Lucks, and F. Muller. Phelix - Fast Encryption and Authentication in a Single Cryptographic Primitive. eSTREAM, ECRYPT Stream Cipher Project, Report 2005/020, 2005. <http://www.ecrypt.eu.org/stream>.
- [ZNC04] J. Zambreno, D. Nguyen, and A. Choudary. Exploring Area/Delay Tradeoffs in an AES FPGA Implementation. *The 14th International Conference on Field Programmable Logic and Applications — FPL'04*, Lecture Notes in Computer Science 3203:575 – 585, 2004.

AES on Virtex5

This chapter presents an updated implementation of the Advanced Encryption Standard (AES) on the recent Xilinx Virtex-5 FPGAs. We show how a modified slice structure in these reconfigurable hardware devices results in significant improvement of the design efficiency. In particular, a single substitution box of the AES can fit in 8 FPGA slices. We combine these technological changes with a sound intertwining of the round and key round functionalities in order to produce encryption and decryption architectures that perfectly fit the Digital Cinema Initiative specifications. More generally, our implementations are convenient for any application requiring Gbps-range throughput.

D.1 Introduction

Reprogrammable hardware devices are highly attractive options for the implementation of encryption algorithms. During the selection process of the AES [FIPS197], an important criterion was the efficiency of the cipher in different platforms, including FPGAs. Consequently, since 2001, various implementations have been proposed, exploring the different possible design tradeoffs ranging from the highest throughput to the smallest area [JTS05]. Each of those implementations usually focuses on a particular understanding of “efficiency”. Furthermore, every time a new hardware platform is introduced, a new implementation is to be made in order to comply with and take advantage of its specificities.

Therefore, this chapter aims to provide an update on the performances of the AES, taking the new Xilinx’s Virtex-5 FPGAs as evaluation devices. Our results show how the modified slice structure (*i.e.* the 6-input Look-Up-Tables combined with multiplexors) allows an efficient implementation of the AES substitution box (S-box). We include these technological advances in a state-of-the art architecture for an encryption module. The resulting IP core typically complies with the Digital Cinema Initiative specifications [DCI]: the presented encryption and decryption designs can be used to decrypt the incoming compressed data stream in a digital cinema server and re-encrypt the uncompressed data between the server and the projector. More generally, it is convenient for any application requiring Gbps-range throughput and compares positively with most recently published FPGA implementations of the AES. Although the improvements described in this chapter straightforwardly derive from technological advances, we believe they are of interest for the cryptographic community in order to keep state-of-the-art implementation results

available and detailed in the public literature.

The rest of the chapter is structured as follows. Section D.2 briefly reminds how the AES cipher processes the data. Then, we discuss the specificities of our target platform before fully describe the architecture developed in Section D.4. The implementation results are summarized in Section D.5, together with some selected results from the literature. Finally, our conclusions are in Section D.6.

D.2 Cipher Description

The AES is a substitution permutation network (SPN) allowing the encryption/decryption of data by blocks of 128-bits and supporting key lengths of 128, 192 and 256 bits. In the following, we focus on the 128-bit key version. Its internal state, usually represented as a 4×4 matrix of bytes, is updated by iterating through the round structure (10, 12 or 14 times according to the key size). The round is described as four different byte-oriented transformations.

First, **SubBytes** introduces the non-linearity by taking, for each byte, the modular inverse in $GF(2^8)$ and then applying an affine transformation. Instead of computing distinctly these two steps, the full transformation is achieved by passing each byte through an S-box (Figure D.1). Then **ShiftRows** modifies the state. It simply consists of a circular left shift of the state's rows by 0, 1, 2 and 3 bytes respectively (Figure D.2). Third, **MixColumns** applies a linear transformation to the state's columns (Figure D.3). Each of them is regarded as a polynomial and is multiplied by a fixed polynomial $c(x) = 3 \cdot x^3 + x^2 + x + 2 \pmod{x^4 + 1}$. Finally, the **AddRoundKey** transform mixes the key with the state. As each subkey has the same size as the state, the combination is performed by a simple bitwise XOR between subkey bytes and their corresponding state bytes (Figure D.4). A first key addition is performed before entering the first round, and the last round omits the **MixColumns** transformation.

Prior to the en/de-cryption process, the subkeys have to be generated.

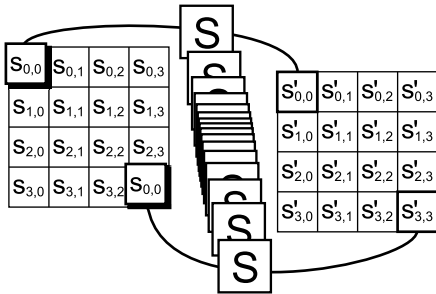


Figure D.1: **SubBytes** transformation.

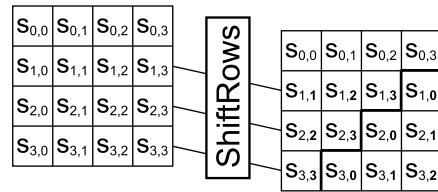
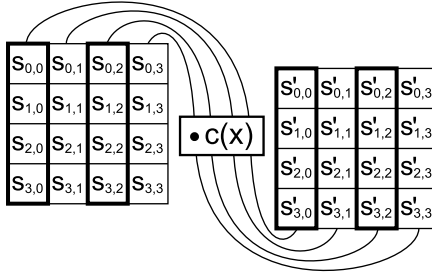
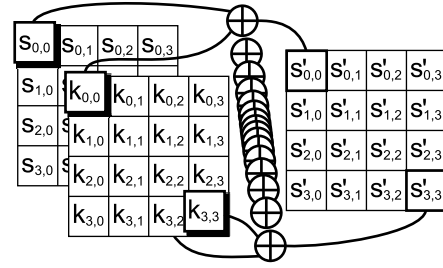


Figure D.2: **ShiftRows** transformation.

The key schedule takes the main key K_0 and expands it as shown in Fig. D.5 for the case of a 128-bit key, where **SubWord** applies the S-box to the 32-bit input word, **RotWord** rotates the word one byte to the left and $RC(i)$ is an 8-bit constant associated to

Figure D.3: **MixColumns** transformation.Figure D.4: **AddRoundKey** transform

each round i ¹.

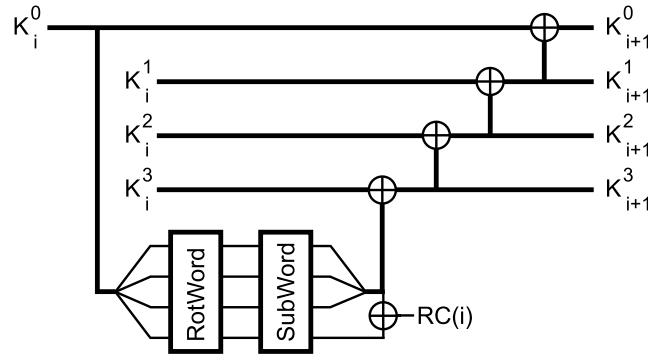


Figure D.5: AES 128-bit key expansion round.

Let us finally mention that the decryption process slightly differs from the encryption one. In order to decipher data blocks, the inverse transformations have to be applied to the state. These operations are respectively called **Inv-Transform**, except for the **AddRoundKey** as it is its own inverse, and are applied in the same order as described above. This may result in a different performance of the AES in encryption and decryption. Regarding the key schedule, the operations remain the same as for encryption but the subkeys have to be introduced in reverse order. For more details, we refer to [FIPS197].

D.3 Target Platform

The chosen platform for implementation is a Xilinx Virtex-5 FPGA. Nowadays, such devices embed programmable logic blocks, RAM memories and multipliers². Compared to

¹In encryption mode, this can easily be performed “on-the-fly”, *i.e.* in parallel to the rounds execution in order to get the subkey at the exact time it is needed. In decryption mode, the round keys generally have to be derived prior to the decipher. Solutions allowing “on-the-fly” derivation of the decryption subkeys require specific features (*i.e.* knowledge of a decryption key that corresponds to the last encryption subkeys) and hardware overhead. Therefore, these are not considered in this chapter.

²Additionally, certain devices also embed microprocessors (PowerPC).

the early reconfigurable devices, these features allow recent FPGAs to provide interesting solutions for a wide range of applications. In this description, the focus will be set on logic elements as other resources will not be used. In Xilinx FPGAs, the slice is the logic unit that is used to evaluate the area requirement of a particular design. The Virtex-5 platform exhibits two kinds of slices. Each of these contains 4 Look-Up Tables (LUTs), 4 flip-flops and some additional gates. These elements, defining the “basic” slice (sliceL), provide the user with logic, arithmetic and ROM functions. In addition, another version of the slice (sliceM) adds the capability to be configured as a distributed RAM or as a shift register. Those enhanced slices represent about 25% of the total amount of available slices. Figure D.6 shows the difference between sliceLs and sliceMs.

D.4 Architecture

A lot of architectures for the AES have been presented in the open literature. Each of those target a specific application and accordingly, various tradeoffs of pipelining, unrolling, datapath width and S-boxes designs have been presented. These contributions do generally agree that the most critical part in an AES design is the S-box. To our knowledge, three different methods have been exploited in order to achieve efficient implementations:

Logic. In this first proposal, one 256×8 -bit S-box is required for each byte of the state. Implementing this as a large multiplexor on platforms where LUTs provide 4-to-1 computation and by taking advantage of special FPGA configurations (*i.e.* MuxF5s and MuxF6s), led the authors of [SRQL03b] to consume 144 LUTs for one S-box. In the case of a full length datapath (128-bit), 2304 LUTs (144×16) are required to perform the whole substitution. This method results in a logic depth of 2 slices. Those two levels can be advantageously pipelined to prevent frequency reduction.

Algorithmic. Another approach to compute **SubBytes** is to directly implement the multiplicative inverse and affine transforms. In order to make it more efficient, Rijmen suggested in [Rij] to move computations from $GF(2^8)$ to the composite field $GF((2^4)^2)$. The main advantage relies on the reduced size of the inversion table: $2^4 \times 4$ instead of $2^8 \times 8$. However, some logic is required to implement transformations to and from such composite field. This type of implementation has been exploited in [RDJ⁺01, HV04] for example.

RAM. Embedded BlockRAMs on FPGA platforms can also be used to store the S-boxes. Such an approach achieves high-throughput performances in [SMMS03]. If enough of these memories are available, another idea is to combine **SubByte** and **Mix-Columns** in a single memory table, as sometimes proposed in software implementations. Examples of such designs are in [MM01b, RSQL04]. Depending on the size and availability of RAM blocks, these solutions may be convenient in practice for recent FPGA devices.

In our context, the choice is straightforward. Due to the technology evolution, a 256×8 -bit S-box fits in 32 LUTs. Using a similar approach as in [SRQL03b], four LUTs make

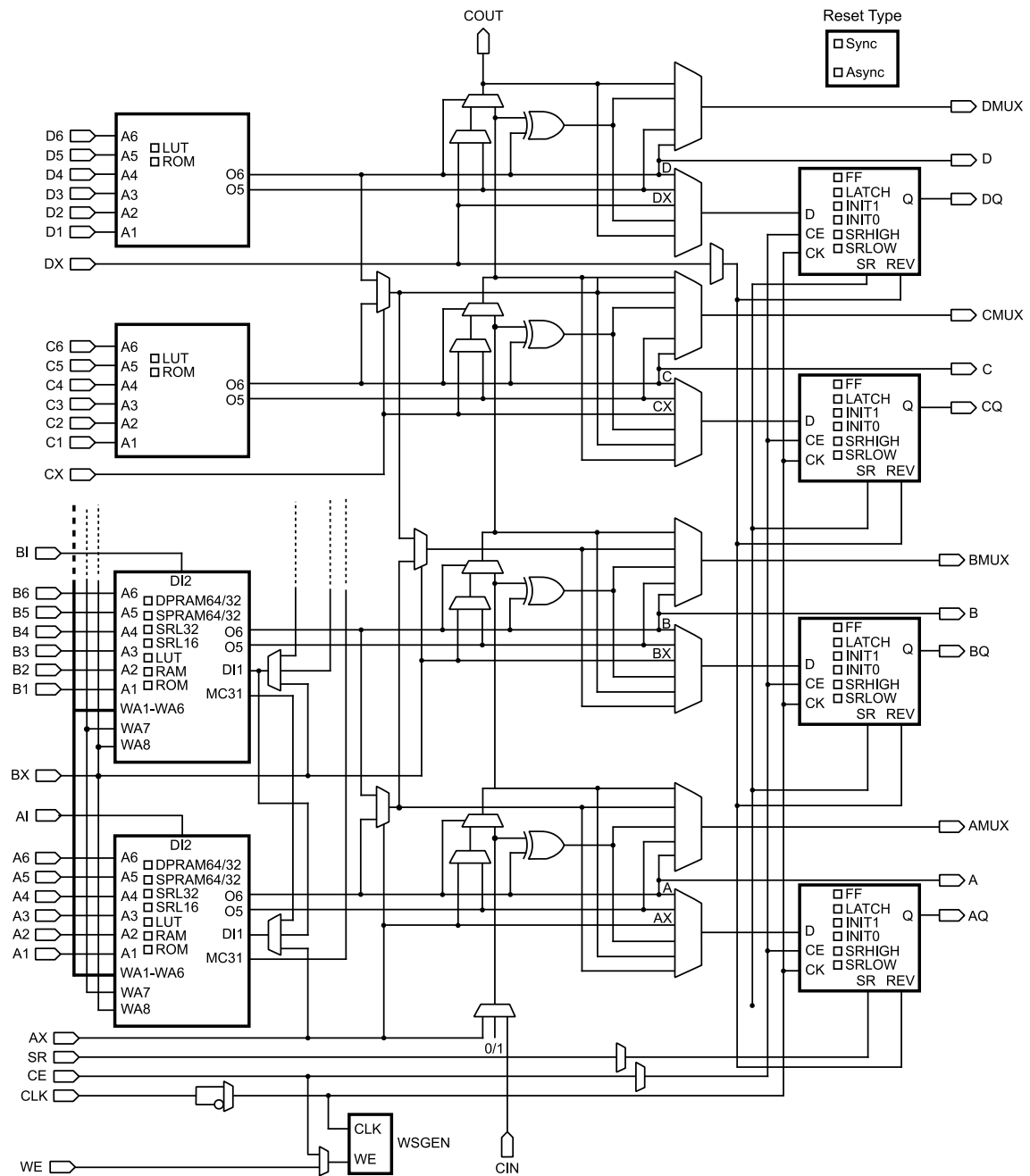


Figure D.6: Virtex-5 mixed view of top-half sliceL and bottom-half sliceM.

As far as the key expansion is concerned and when dealing with large amount of data, like in the Digital Cinema context, computing all subkeys prior to en/de-cryption seems a better alternative than an “on-the-fly” key schedule. Indeed, the overhead due to a master key change quickly vanishes as the number of messages using this same key increases. It also allows us to reduce the area requirements of the complete design. As the key schedule and the encryption module use the same S-boxes, these are shared in our architecture. Multiplexors allow the S-box inputs to change between the state and the subkey. These multiplexors do not increase the implementation cost as they are packed in LUTs together with the key addition. The remaining of the subkeys’ computation proceeds as explained in Section D.2. Each subkey is written in a RAM configured sliceM. They are brought back when needed for en/de-cryption. We note that the key schedule must be performed before decryption takes place anyway, as the subkeys have to be applied in reverse order. Also, the S-Boxes sharing does not hold for the decryption architecture. Indeed, the S-boxes implementing **SubBytes** for the key schedule can not be reused for deciphering. This is because decryption involves the **InvSubBytes** operation that do not yield the same S-boxes.

This architecture perfectly suits the needs of ECB and Counter modes of operations. It could also be tuned to handle the CBC mode, at the cost of a reduced efficiency. Indeed, as the plaintext block is to be XORed with the previous ciphertext before being encrypted, the four pipeline stages of the round do not allow encryption of four plaintext blocks at the same time. Note that in the case of CBC decryption, this restriction does not hold as this additional XOR is performed after looping through the round.

Although the focus of this chapter is the implementation of an AES en/decryption module on a Virtex-5, we also investigated this IP core in Virtex-4 and Spartan-3 FPGAs, for illustration/comparison purposes. The architecture remains the same as the one presented here for the Virtex-5. The only difference relies on the way S-boxes are implemented. In the case of a Virtex-4, the S-boxes are made up of BlockRAMs. Each of the blockRAM has a datapath width of 32-bit that allows the output of the S-box to be stored times 0, 1, 2 and 3. That is, a part of the **MixColumns** computation is made while passing through the RAMs. To make things clear, let us consider the combination of SubBytes and MixColumns in the AES. An output column of this transform equals:

$$\begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \times \begin{bmatrix} SB(a_0) \\ SB(a_1) \\ SB(a_2) \\ SB(a_3) \end{bmatrix},$$

where the b_i ’s represent the transform output bytes and the a_i ’s its input bytes. The b_i vector is equivalent to:

$$\begin{bmatrix} 02 \\ 01 \\ 01 \\ 03 \end{bmatrix} \times [SB(a_0)] \oplus \begin{bmatrix} 03 \\ 02 \\ 01 \\ 01 \end{bmatrix} \times [SB(a_1)] \oplus \begin{bmatrix} 01 \\ 03 \\ 02 \\ 01 \end{bmatrix} \times [SB(a_2)] \oplus \begin{bmatrix} 01 \\ 01 \\ 03 \\ 02 \end{bmatrix} \times [SB(a_3)]$$

Therefore, if we define four tables as:

$$T_0(a) = \begin{bmatrix} 02 \times SB(a) \\ SB(a) \\ SB(a) \\ 03 \times SB(a) \end{bmatrix}, \quad T_1(a) = \begin{bmatrix} 03 \times SB(a) \\ 02 \times SB(a) \\ SB(a) \\ SB(a) \end{bmatrix},$$

$$T_2(a) = \begin{bmatrix} SB(a) \\ 03 \times SB(a) \\ 02 \times SB(a) \\ SB(a) \end{bmatrix}, \quad T_3(a) = \begin{bmatrix} SB(a) \\ SB(a) \\ 03 \times SB(a) \\ 02 \times SB(a) \end{bmatrix},$$

the combination of SubBytes and MixColumns equals:

$$\begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix} = T_0(a) \oplus T_1(a) \oplus T_2(a) \oplus T_3(a)$$

In our Virtex-4 implementation, these T tables are stored in RAM and all what is left to complete the **MixColumns** transform is a single level of logic handling the XOR of the four bytes. On Spartan-3 devices, the situation is different since the BlockRAMs do not provide a dedicated latch at their output. Reproducing the behaviour of Virtex-4 requires 24-bits to be stored using the slice flip-flops which consumes much more area. Since XORing the table outputs without using the slice flip flops causes a reduction of the work frequency, the most efficient solution is to implement **MixColumns** and **SubBytes** independently.

D.5 Results

The AES designs were described using VHDL. Synthesis and Place & Route were achieved on Xilinx ISE 9.1i. The selected devices are Xilinx's Virtex-5, Virtex-4 and Spartan-3. Table D.1 summarizes the results achieved for both the encryption and decryption (Enc/Dec) modules. Moreover, some previous results are summarized in Table D.2. As it is generally true for any comparison of hardware performances, those results have to be taken with care since they relate to different FPGA devices. In the Virtex-5 FPGAs, a slice is made up of 4 LUTs instead of 2 for previous Xilinx devices. In order to allow fair(er) comparison, it then makes sense to double the figures as if a slice was 2 LUTs. This is taken into account into the parenthesis of Table D.1. Compared to previous devices, the benefit of Virtex-5 is easily underlined. It corresponds either to the removal of the block-RAMs from the design on the Virtex-4 or a 50% slice reduction from a full logic design on Spartan-3 FPGAs. This strongly emphasized the advantage of technology evolution shifting from 4 to 6 input bits LUTs.

Additional insights on our implementation results can be obtained by looking at Table D.2. Namely, the proposed architectures range among the efficient ones found out

Device	Slices	BRAM	Freq. (MHz)	Thr. (Gbps)	Thr. / Area (Mbps/slice)
Virtex-5	400 / 550 (800 / 1100)	0	350	4.1	10.2 / 7.4
Virtex-4*	700 / 1220	8	250	2.9	4.1 / 2.3 *
Spartan-3	1800 / 2150	0	150	1.7	0.9 / 0.8

Table D.1: Implementation Results: encryption/decryption designs.

Device	Datapath	Slices	BRAM	Freq. (MHz)	Thr. (Gbps)	Thr./Area (Mbps/slice)
Spartan-2 [GB05]*	8	124	2	—	0.002	0.02*
Virtex-2 [RSQL04]*	32	146	3	123	0.358	2.45*
Virtex-E [SRQL03a]*	128	542	10	119	1.45	2.67*
Virtex-E [SRQL03b]	128	2257	0	169	2.0	0.88
Virtex-2 [ZNC04]*	128	387	10	110.16	1.41	3.64*
Virtex-2 [ZNC04]	128	1780	0	77.91	1.0	0.56
Virtex-4 [LWFB07]	128	18400	0	140	17.9	0.97
Virtex-5 [HT]	128	349	0	350	4.1	11.67

Table D.2: Previous implementations.

in the literature. Again, these observations have to be considered as general intuitions rather than fair comparisons since they consider different FPGA technologies: more recent FPGAs have higher work frequencies and thus throughput. In addition, the hardware efficiency (*i.e.* throughput/area ratio) of the *-marked implementations is not meaningful since they consume FPGA RAM blocks. Finally, the hardware cost can only be compared if the respective implementation efficiencies (*e.g.* measured with the throughput/area ratio) are somewhat comparable. As a matter of fact, it is always possible to reduce the implementation cost, by considering smaller datapaths (*e.g.* [GB05] uses an 8-bit datapath for the AES, [RSQL04] uses a 32-bit datapath, all the others use 128-bit architectures) at the cost of a reduced throughput.

In the case of Helion Technology’s implementation [HT], the comparison is more interesting since it relates to the same Virtex-5 platform as ours. At first sight, their Fast AES Encryption core seems to consume less area than the proposed architecture. However, the gap can be reduced if we assume that their core uses an “on-the-fly” key schedule. In such a case, the distributed RAM used to store subkeys is to be removed from our presented design (along with its control logic) which allows earning at least 32 slices. This makes both designs very close. As a matter of fact, the differences between these cores mainly relate to different optimization goal. Our was to design encryption and decryption IPs exploiting a very similar architecture with a key scheduling algorithm executed prior to the encryption/decryption process. We note that not using the “on-the-fly” key scheduling for encryption makes sense for power consumption reasons. If the implementation context does not require frequent key changes, there is no need to re-

compute these keys for every plaintext.

D.6 Conclusion

This chapter reports implementation results of the AES algorithm on the new Virtex-5 devices. It exhibits the (straightforward but significant) benefits that can be drawn from the technology evolution within recent FPGAs. In particular it is shown how the AES substitution box perfectly suits the new Virtex-5 slice structure using 6-bit LUTs. This enables reducing the cost of a single S-box from 144 down to 32 LUTs ! Compared to 4 input bit LUTs-based designs, this advantage roughly corresponds to either the removal of blocks of embedded RAM memory or a slice count reduction of 50%, depending on the design choices. The proposed architectures range among the most efficient ones published in the open literature. Their reasonable implementation cost makes them a suitable solution for a wide range of applications requiring Gbps-range throughput, including digital cinema and network encryption.

Bibliography

- [Can05] D. Canright. A Very Compact S-box for AES. *Workshop on Cryptographic Hardware and Embedded Systems — CHES'05*, Lecture Notes in Computer Science 3659:441–456, 2005.
- [CG03] P. Chodowiec and K. Gaj. Very Compact FPGA Implementation of the AES Algorithm. *Workshop on Cryptographic Hardware and Embedded Systems — CHES'03*, Lecture Notes in Computer Science 2779:319–333, 2003.
- [DCI] Digital Cinema Initiative. DCI System Specification, V1.0, July 20 2005.
- [FDW04] M. Feldhofer, S. Dominikus, and J. Wolkerstorfer. Strong Authentication for RFID Systems Using the AES Algorithms. *Workshop on Cryptographic Hardware and Embedded Systems — CHES'04*, Lecture Notes in Computer Science 3156:357 – 370, 2004.
- [FIPS197] National Institute of Standards and Technology. Advanced Encryption Standard (AES). Federal Information Processing Standards Publications – FIPS 197, <http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf>, November 2001.
- [GB05] M. Good and M. Benaissa. AES on FPGA from the Fastest to the Smallest. *Workshop on Cryptographic Hardware and Embedded Systems — CHES'05*, Lecture Notes in Computer Science 3659:427 – 440, 2005.
- [HT] Helion Technology. <http://www.heliontech.com>.
- [HV04] A. Hodjat and I. Verbauwhede. A 21.54 Gbits/s Fully Pipelined AES coprocessor on FPGA. *12th. Annual IEEE Symposium on Field Programmable Custom Computing Machine — FCCM'04*, pages 308–309, 2004.
- [IPC] IP Cores. <http://ipcores.com/index.htm>.
- [JTS05] K. Järvinen, M. Tommiska, and J. Skyttä. Comparative survey of high-performance cryptographic algorithm implementations on FPGAs. *IEE Proceedings on Information Security*, 152(1):3–12, 2005.

- [LWFB07] S. Lemsitzer, J. Wolkerstorfer, N. Felbert, and M. Braendli. Multi-gigabit GCM-AES Architecture Optimized for FPGAs. *Workshop on Cryptographic Hardware and Embedded Systems — CHES'07*, Lecture Notes in Computer Science 4727:227–238, 2007.
- [MM01a] M. McLoone and J. V. McCanny. High Performance Single-Chip FPGA Rijndael Algorithm Implementation. *Workshop on Cryptographic Hardware and Embedded Systems — CHES'01*, Lecture Notes in Computer Science 2162:65–76, 2001.
- [MM01b] M. McLoone and J. V. McCanny. Rijndael FPGA Implementation Utilizing Look-Ups Tables. *IEEE Workshop on Signal Processing Systems — SIPS'01*, pages 349–360, 2001.
- [MM01c] M. McLoone and J. V. McCanny. Single-Chip FPGA Implementation of the Advanced Encryption Standard Algorithm. *11th. International Conference on Field-Programmable Logic and Applications — FPL'01*, pages 152–161, 2001.
- [RDJ⁺01] A. Rudra, P. K. Dubey, C. S. Jutla, V. Kumar, J. R. Rao, and P. Rohatgi. Efficient Rijndael Encryption Implementation with Composite Field Arithmetic. *Workshop on Cryptographic Hardware and Embedded Systems — CHES'01*, Lecture Notes in Computer Science 2162:171–184, 2001.
- [Rij] Vincent Rijmen. Efficient implementation of the Rijndael S-box. www.citeseer.ist.psu.edu/rijmen00efficient.html.
- [RSQL04] G. Rouvroy, F.-X. Standaert, J.-J. Quisquater, and J.-D. Legat. Compact and Efficient Encryption/Decryption Module for FPGA Implementation of the AES Rijndael Very Well Suited for Small Embedded Applications. *International Symposium on Information Technology: Coding and Computing — ITCC'04*, pages 583 – 587, 2004.
- [SM03] A. Satoh and S. Morioka. Unified Hardware Architecture for the 128-bit Block Ciphers AES and Camellia. *Workshop on Cryptographic Hardware and Embedded Systems — CHES'03*, Lecture Notes in Computer Science 2779:304–318, 2003.
- [SMMS03] G. P. Saggese, A. Mazzeo, N. Mazzocca, and A. G. M. Strollo. An FPGA-based Performance Analysis of the Unrolling, Tiling and Pipelining of the AES Algorithm. *13th. International Conference on Field Programmable Logic and Applications — FPL'03*, pages 292–302, 2003.
- [SRHDP03] N. A. Saqib, F. Rodríguez-Hendríquez, and A. Díaz-Pérez. AES Algorithm Implementation – An Efficient Approach for Sequential and Pipeline Architectures. *4th. Mexican International Computer Science — ENC'03*, pages 126–130, 2003.

- [SRQL03a] F.-X. Standaert, G. Rouvroy, J.-J. Quisquater, and J.-D. Legat. A Methodology to Implement Block Ciphers in Reconfigurable Hardware and its Application to Fast and Compact AES RIJNDAEL. *11th. ACM International Symposium on Field-Programmable Gate Arrays — FPGA'03*, pages 216–224, 2003.
- [SRQL03b] F.-X. Standaert, G. Rouvroy, J.-J. Quisquater, and J.-D. Legat. Efficient Implementation of Rijndael in Reconfigurable Hardware: Improvements and Design Tradeoffs. *Workshop on Cryptographic Hardware and Embedded Systems — CHES'03*, Lecture Notes in Computer Science 2779:334–350, 2003.
- [ZNC04] J. Zambreno, D. Nguyen, and A. Choudary. Exploring Area/Delay Tradeoffs in an AES FPGA Implementation. *The 14th International Conference on Field Programmable Logic and Applications — FPL'04*, Lecture Notes in Computer Science 3203:575 – 585, 2004.

