KATHOLIEKE UNIVERSITEIT LEUVEN
FACULTEIT TOEGEPASTE WETENSCHAPPEN
DEPARTEMENT ELEKTROTECHNIEK
Kasteelpark Arenberg 10, 3001 Leuven (Heverlee)

HARDWARE DESIGN OF ELLIPTIC CURVE CRYPTOSYSTEMS AND SIDE-CHANNEL ATTACKS

Promotors:
Prof. dr. ir. B. Preneel
Prof. dr. ir. I. Verbauwhede

Proefschrift voorgedragen tot
het behalen van het doctoraat
in de toegepaste wetenschappen

door

Sıddıka Berna Örs Yalçın

Februari 2005

KATHOLIEKE UNIVERSITEIT LEUVEN
FACULTEIT TOEGEPASTE WETENSCHAPPEN
DEPARTEMENT ELEKTROTECHNIEK
Kasteelpark Arenberg 10, 3001 Leuven (Heverlee)

HARDWARE DESIGN OF ELLIPTIC CURVE CRYPTOSYSTEMS AND SIDE-CHANNEL ATTACKS

Jury:
Prof. dr. ir. H. Van Brussel, voorzitter
Prof. dr. ir. B. Preneel, promotor
Prof. dr. ir. I. Verbauwhede, promotor
Prof. dr. ir. L. Claesen
Prof. dr. ir. W. Dehaene
Prof. dr. ir. H. De Man
Prof. dr. ir. J.-J. Quisquater (Université Catholique de Louvain)
Prof. dr. ir. J. Vandewalle

Proefschrift voorgedragen tot
het behalen van het doctoraat
in de toegepaste wetenschappen

door

Sıddıka Berna Örs Yalçın

U.D.C. 681.3*D46 Februari 2005

D/2005/7515/18

ISBN 90-5682-584-4

Acknowledgements

At the end of my doctoral work it is a pleasure to thank those who have helped me along the way.

First of all I would like to express my appreciation to Prof. Bart Preneel for giving me the opportunity to work at COSIC. I am always amazed with his energy and enthusiasm for his work. It was always very instructive and exciting when he came back from a conference, a visit, etc. full of new ideas to try. I also appreciate his understanding of ordinary life problems and the solutions he always provided to me.

I would like to thank Prof. Ingrid Verbauwhede for her ideas, teaching and comments during the short period of our working time together.

I would like to thank Prof. Joos Vandewalle for the day that he gave me the time to talk about a place at ESAT. In fact this was the first step to COSIC.

I am thankful to Prof. Luc Claesen for my first year as a pre-doctoral student in Leuven. If he would not accept to meet me after a summer school at IMEC and give the opportunity to study there for one year this long way would not start.

I would like to thank Prof. Hugo De Man for reading my thesis and for his very useful comments to improve it. I would like to thank Prof. Jean-Jacques Quisquater and Prof. Wim Dehaene for being a jury member for my defense.

Prof. Hendrik Van Brussel is gratefully acknowledged for chairing the jury.

I would like to thank to my M.Sc thesis promotor from Istanbul Prof. Dr. Ahmet Devişoğlu for introducing me to the wonderful world of the university. I thank Prof. Dr. Fuat Anday for allowing me to be on a leave from Istanbul Technical University during this thesis study.

I thank Elisabeth Oswald as a colleague, a co-author and a friend. It was and will be a pleasure to discuss future works with the company of a nice “mousse au chocolat”.

I would like to thank Kağan Gürkaynak first for his long friendship and second

for our work together. I will always remember the nice dinner in Amsterdam with a big discussion which ended up as a nice work.

I would like to acknowledge the collaboration with François-Xavier Standaert. I have learned many things from him and working with him was a great pleasure.

I would like to thank Lejla Batina, Nele Mentens and Elke De Mulder for their friendship and the working experience we had all together. I had delightful days as a member of the COSIC girls hardware group.

I feel myself very lucky that Nele Mentens, Pieter Rommens and Marian Verhelst, Pieter Buysschaert and Elke De Mulder decided to do their thesis study with me. We worked hard and had lots of fun eating ice-cream, making jokes even dancing together.

I would like to thank Péla Noë for solving my problems even before I know that they existed. I will always miss the chats we had together.

I would like to thank all my colleagues for the excellent working atmosphere. Special thanks to Svetla Nikova, Michael Quisquater, Karel Wouters, Jurgen Truyen, Claudia Diaz, Danny De Cock, Jasper Scholten, Frederik Vercauteren, Christopher Wolf and Elvira Wouters.

The financial support of the Research Council Scholarships programme of the Katholieke Universiteit Leuven and Fonds voor Wetenschappelijk Onderzoek -Vlaanderen are greatly appreciated.

I would like to thank Bilge Bayrakçı, Ebru Şahin, Kanat Çamlıbel, Eralp and Deniz Tezcan who have always been there for the moral support.

I want to thank İffet and Yıldırım Yalçın. They always provided all their help to support me during the hard study periods of this thesis.

I would like to thank my mother, Hacer, my father, Hüseyin, and my brother, Ahmet. Over the years they have continued to support and encourage me in my studies and my doctoral work. It is a great feeling to know that they are always with me where and when I need them. *Anneme, babama ve kardeşime sonsuz teşekkürler.*

I would like to thank our daughter Setenay. She was a surprise in the middle of the thesis study. She was with me for nine months also hearing the computer voices and feeling the excitement. She is giving us the happiness that we could not dream about.

I want to thank my husband Müştak for love and happiness he is bringing in my life. He had to listen to me and see the tears in my eyes during the stressful days and nights. This thesis would not be possible without his support.

Sıddıka Berna Örs Yalçın

Leuven, February 2005

Abstract

In this thesis, we present FPGA implementations of the Montgomery modular multiplication over $GF\,(2^m)$ and $GF(p)$. The designs have systolic array architectures to allow pipelining and to make the clock frequency independent of m and the bit-length of p. In this way, the clock frequency does not change when the bit-length is enlarged for security reasons. We describe efficient implementations of elliptic curve processors over $GF(p)$ and $GF\,(2^m)$. The processors can be programmed to execute a modular multiplication, addition/subtraction, modular multiplicative inversion, elliptic curve point addition/doubling and multiplication. It is desirable to use the resulting FPGA implementations also for an evaluation of the designed circuits against power analysis attacks. In order to configure the FPGA, to communicate with it, and to conduct measurements on it during the execution of the cryptographic algorithms, we have designed our own FPGA board. Then we were free to design it such that gives us the best information about the dynamic power consumption of the FPGA. We conduct timing, power and electromagnetic analysis attacks on our FPGA implementations of elliptic curve cryptosystems over $GF(p)$. We conclude that our initial design was vulnerable to simple attacks: by using only timing information it is possible to find the Hamming weight of the key. More drastically, by using simple power and electromagnetic analysis attacks it was possible to find all the key bits. Next we present an improved design such that they become immune to timing and simple power and electromagnetic analysis attacks. We conduct differential power and electromagnetic analysis attacks on our improved FPGA implementation of elliptic curve processor. To our knowledge, there exists no previous publication on practical implementations of power analysis attacks on dedicated hardware implementations of the Advanced Encryption Standard (AES). In this thesis we demonstrate the feasibility of power analysis attacks against hardware implementations of the AES. Our attacks target an ASIC implementation of the AES developed by the ETH Zürich and an FPGA implementation of the AES developed by the UCL Crypto group. In this thesis we describe a power analysis attack against an FPGA implementation of the Data Encryption Standard (DES) developed by the UCL Crypto group.

Contents

List of Abbreviations

AES	Advanced Encryption Standard
ALU	Arithmetic Logic Unit
AM	Amplitude Modulation
AOP	All One Polynomial
ASC	Addition/Subtraction Circuit
ASIC	Application Specific Integrated Circuit
ASM	Algorithmic State Machine
AtoP	Affine to Projective coordinates converter
AU	Arithmetic Unit
AUC	Arithmetic Unit Controller
AWP	Asynchronous Wave Pipeline
CDB	Circular Dual Basis
CLB	Configurable Logic Block
CMOS	Complementary Metal Oxide Semiconductor
CPU	cCentral Processing Unit
CRT	Chinese Remainder Theorem
DB	Dual Basis
DC	Direct Current
DEMA	Differential ElectroMagnetic Analysis
DES	Data Encryption Standard
DH	Diffie-Hellman
DHP	Diffie-Hellman Problem
DL	Discrete Logarithm
DLP	Discrete Logarithm Problem
DPA	Differential Power Analysis
DSA	Digital Signature Algorithm
DSO	Digital Storage Oscilloscope
DSRCP	Domain-Specific Reconfigurable Cryptographic Processor
DSS	Digital Signature Standard
EC	Elliptic Curve
ECC	Elliptic Curves Cryptosystem
ECDH	Elliptic Curve Diffie-Helman
ECDLP	Elliptic Curve Discrete Logarithm Problem

ECDSA	Elliptic Curve Digital Signature Algorithm
ECIES	Elliptic Curve Integrated Encryption Scheme
ECP	Elliptic Curve Processor
ECPAC	Elliptic Curve Point Addition Circuit
ECPDC	Elliptic Curve Point Doubling Circuit
ECPM	Elliptic Curve Point Multiplier
EEA	Extended Euclidean Algorithm
EM	ElectroMagnetic
EMA	Electromagnetic Analysis
ESP	Equally Spaced Polynomial
FA	Full Adder
FF	Flip-Flops
FG	Function Generator
FIPS	Federal Information Processing Standard
FM	Frequency Modulation
FPGA	Field Programmable Gate Array
FPSLIC	Field Programmable System Level Integration Circuit
FSM	Finite State Machine
GBB	Ghost Bit Basis
GCD	Greatest Common Divisor
GPS	Girault Poupard Stern
HA	Half Adder
HCMOS	High speed CMOS
HDL	Hardware Description Language
IDEA	International Data Encryption Algorithm
I/O	Input/Output
IOB	Input/Output Block
IP	Internet Protocol
IPA	Inferential Power Analysis
IPSEC	IP Security Protocol
ISA	Instruction Set Architecture
LC	Logic Cell
LSB	Least Significant Bit
LSD	Least Significant Digit
LSR	Left-Shift Register
LSW	Least Significant Word
LUT	Look-Up Table
MA	Modular Addition
MC	Main Controller
MMI	Modular Multiplicative Inverter
MMM	Montgomery Modular Multiplication
MMMC	Montgomery Modular Multiplication Circuit
MO	Massey-Omura
MSB	Most Significant Bit
MtoN	Montgomery to Normal representation converter

NAF	Non-Adjacent Form
NB	Normal Basis
NIST	National Institute of Standards and Technology
NMOS	Negative-channel Metal-Oxide Semiconductor
NOP	No-OPerations
NtoM	Normal to Montgomery representation converter
ONB	Optimal Normal Basis
OTP	One-Time Programmable
P	Permutation
PA	Power Analysis
PB	Polynomial Basis
PC	Personal Computer
PKC	Public-Key Cryptography
PRR	Polynomial Ring Representation
PtoA	Projective to Affine coordinates converter
RAM	Random Access Memory
RBR	Redundant Bases Representation
RC	Rivest Cipher
RNG	Random Number Generator
ROM	Read Only Memory
RSA	Rivest-Shamir-Adelman
RSR	Right-Shift Register
RST	Reset Signal
SCA	Side-Channel Analysis
SEMA	Simple ElectroMagnetic Analysis
SHA	Secure Hash Algorithm
SPA	Simple Power Analysis
SQUIDS	Superconducting QUantum Interference DeviceS
SRAM	Static Random Access Memory
SSH	Secure Shell
SSL	Secure Sockets Layer
SSR	Serial Shift Register
TLS	Transport Layer Security
TA	Timing Analysis
VHDL	Very High Speed Integrated Circuit Hardware Description Language
VLSI	Very Large Scale Integration

List of Notations

$x \in Y$	x is an element of group Y
$\log_x y$	logarithm of y to the base x
x^y	exponentiation of x to y
$\mathbb{Z}_p^*$	ring of p-adic integers
$x^y \bmod z$	modular z exponentiation of x to y
$\mathcal{O}$	point at infinity
E	elliptic curve
P	point on an elliptic curve E
$\mathbb{R}$	field of real numbers
$GF(q)$	Galois field with q elements
p	prime, mostly characteristic of field
$[x]P$	elliptic curve point multiplication x and P
P^2	projective plane
(x, y)	affine point
(X, Y, Z)	projective point
a	parameter of elliptic curve
(X, Y, Z, aZ^4)	point in modified Jacobian coordinates
J^m	Jacobian coordinate system
$E(GF(q))$	an elliptic curve over $GF(q)$
L	32-bit left half register
R	32-bit right half register
L_i	the value of L at the ith round
R_i	the value of L at the ith round
K_i	48-bit subkey of round i
$X \|\| Y$	the concatenation of bit strings X and Y
$IP()$	initial permutation
$f(x, y)$	function of x and y
$E(x)$	expansion of x register

$AddRoundKey()$	round key addition to the state using an XOR operation
$InvMixColumns()$	inverse of MixColumns()
$InvShiftRows()$	inverse of ShiftRows()
$InvSubBytes()$	inverse of SubBytes()
K	cipher key
$MixColumns()$	transformation in the cipher that takes all of the columns of the state and mixes their data to produce new columns
Nb	number of columns
Nk	number of 32-bit words comprising the cipher key
Nr	number of rounds
$Rcon[]$	the round constant word array
$RotWord()$	cyclic permutation function
$ShiftRows()$	transformation in the cipher that processes the state by cyclically shifting the last three rows of the state
$SubBytes()$	transformation in the cipher that processes the state using a nonlinear byte substitution table (S-box) that operates on each of the state bytes independently
$SubWord()$	function used in the key expansion routine that takes a four-byte input word and applies an S-box to each of the four bytes to produce an output word
XOR	exclusive-OR operation
$\bigoplus$	exclusive-OR operation
$\bigotimes$	multiplication of two polynomials modulo $x^4 + 1$
$\bullet$	finite field multiplication
$C(X, Y)$	Pearson correlation coefficient between the sets X and Y
$E(X)$	expectation (average) of set X
$Var(X)$	variance of a set X
$\frac{\mathrm{d}f}{\mathrm{d}x}$	derivative of a function f with respect to x
$\vec{x}$	vector x
$\vec{x} \times \vec{y}$	vector product of $\vec{x}$ and $\vec{y}$
$\hat{\vec{r}}$	unit vector
$X * Y$the product in $GF(2^m)$. $Tr_F^K(x)$	trace of $x \in K$ relative to subfield F

$\lceil x \rceil$	smallest integer not less than x
$a^{-1}(x)$	multiplicative inverse of $a(x)$
$a_i(x)$	value of $a(x)$ at ith iteration step
$A_{i,j}(x)$	jth word of $a_i(x)$
$<< x$	left shift over x bits
$>> x$	right shift over x bits
$x\&y$	concatenation of x and y
$\mathrm{Mont}(x, y)$	Montgomery modular multiplication of x and y
$xR \bmod N$	Montgomery representation of x
$a_{i,j}$	jth bit of register A at ith iteration
$H(X)$	Hamming weight

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

Cryptography is the science of information integrity [239]. The best known use of cryptography is providing secrecy of information. Until the mid seventies this was indeed its main purpose. The cryptosystems used were so-called symmetric key systems. In these systems secure communication requires that all parties share the same key that has to remain secret to others. In 1976 Diffie and Hellman published an article [58] where a fundamental new approach to cryptography was introduced: public key cryptography. Here the key is split into two parts: a public key and a private key. This has the following consequences: many people can encrypt messages that only can be decrypted by one person and one person can sign messages that many people can verify. The first property makes key management significantly 'easier'. The second has its merits in producing digital signatures.

The security of any cryptosystem is based on a mathematical problem that is hard to solve. Since its introduction many problems have been proposed for use in public key cryptography. Until now the two most popular ones are:

- factorization of a composite number, like RSA;
- discrete logarithm problem in a group, like Diffie-Hellman, ElGamal.

The group in the second problem is typically a finite field. However, one can define a group structure on an elliptic curve. The first elliptic curve cryptosystems (ECCs) were independently proposed by Koblitz [126] and Miller [165] in 1985. Since its inception elliptic curve cryptography has been the subject of

extensive cryptanalysis. Today, elliptic curve cryptosystems are deemed secure for commercial [12, 13, 108] as well as government use [174].

Elliptic curve cryptosystems (ECCs) offer security comparable to that of traditional public-key cryptosystems (PKCs), such as those based on the RSA [220] and ElGamal [61, 62] encryption and digital signature algorithms and those based on the Diffie-Hellman key agreement algorithm [58], with shorter key-length and computationally more efficient algorithms. Because of these advantages over traditional public key algorithms elliptic curve cryptography is becoming popular for use in constrained environments such as cellular phones and smart cards.

Software implementations have the great advantage that they are portable to multiple hardware platforms. Their disadvantages are their high power consumption and lower speed when compared to specialized hardware architectures and their inability to protect private keys from disclosure with the same degree of security that is achievable in hardware. These disadvantages are some of the reasons motivating the study of efficient hardware architectures. Two of the contributions of this thesis are elliptic curve processors over $GF\,(2^m)$ and $GF(p)$ published in [159, 197, 198].

Unlike traditional very large scale integration (VLSI) hardware, field programmable gate arrays (FPGAs) do not possess fixed functionality after fabrication. FPGAs are reconfigurable hardware devices, that is, devices whose functionality is programmable. This reconfigurability allows us to try different architectures for our elliptic curve processor designs. As part of a modern design flow, FPGAs are gaining more importance. The reasons for this include their relatively low cost and the available tools. Register transfer level descriptions for a circuit can easily be ported, if not directly used, for an FPGA implementation of the circuit. These motivate our choice for an FPGA as a hardware platform. In order to verify the functionality and perform some experiments, we have designed our own FPGA board.

The performance of an ECC and of several other PKCs is mostly determined by the efficient implementation of finite field arithmetic. The most critical operation for latency is modular multiplication. In 1985 Montgomery introduced a new method for modular multiplication [168]. The approach of Montgomery avoids the time consuming trial division that is a bottleneck for most other algorithms. His method is very efficient and is the basis of many implementations of modular multiplication, both in software and hardware [19].

In this thesis, we present our FPGA implementations of the Montgomery modular multiplication (MMM) over $GF\,(2^m)$ and $GF(p)$. The designs have systolic array architectures to allow pipelining and to make the clock frequency independent of m and the bit-length of p. In this way, the clock frequency does not change when the bit-length is enlarged for security reasons. These results

were published in [196, 198] for $GF(p)$ and in [157, 158] for $GF(2^m)$. We have described efficient implementations of elliptic curve processors over $GF(p)$ and $GF(2^m)$. The processors can be programmed to execute a modular multiplication, addition/subtraction, modular multiplicative inversion, EC point addition/doubling and multiplication. These results were published in [197, 198] for $GF(p)$ and in [159] for $GF(2^m)$.

A cryptographic primitive can be viewed as an abstract mathematical object (a transformation, possibly parameterized by a key, transforming some input into some output); on the other hand, this primitive will have to be implemented in software or hardware, in a specific environment and will therefore have specific properties. It can be attacked at two levels. The first point of view is that of "classical" cryptanalysis; the second one is that of *side-channel analysis attack*. Side-channel analysis (SCA) attack takes advantage of implementation-specific characteristics to recover the secret parameters involved in the computation. It is therefore less general, but often much more powerful than classical cryptanalysis.

Side-channel analysis attacks can be divided into two groups according to the ability of the attacker; *active* and *passive* attacks . Active attacks targeting the keys in cryptographic devices are commonly referred to as *tamper attacks*; they have a long history in the field of cryptography [11]. The attacker has to reach the internal circuitry of the cryptographic device. One distinguishes between *probing attacks* [135] and *fault induction attacks* [26, 119]. Probing attacks consist inserting sensors into the device, in order to directly examine the content of memory zones or the data circulating on a bus. Fault induction attacks work by disturbing the device's behavior in order to induce errors in the computation.

Passive attacks were recognized in the cryptographic community as a major threat in 1996, when the first article about timing analysis (TA) [132] was published. In a passive attack, the adversary uses the standard functionality of the cryptographic device. The physical and/or electrical effects of the functionality on the device are then used for the attack. There are many different types of effects. If these effects unintentionally deliver information about the key which is used inside the device, then they deliver side-channel information and are called side-channels.

Passive attacks are divided into four groups according to the side-channel information that they exploit. Timing analysis attacks exploit the timing information on the cryptographic hardware. Power analysis (PA) attacks [133] use the dynamic power consumption of the cryptographic hardware during the execution of the cryptographic algorithm. Electromagnetic analysis (EMA) attacks [215, 72] use the electromagnetic radiation of the cryptographic hardware during the execution of the cryptographic algorithm. Acoustic (sound) analysis attacks [236] exploit the sound coming out of the cryptographic hardware

during the execution of the cryptographic algorithm.

All the groups of the passive attacks have two variations. In a *simple analysis*, an attacker uses the side-channel information from one measurement directly to determine (parts of) the secret key. In *differential analysis*, many measurements are used in order to filter out noise. While a simple analysis exploits the relationship between the executed operations and the side-channel information, differential analysis attack exploits the relationship between the processed data and the side-channel information.

It is desirable to use the resulting FPGA implementation also for an evaluation of the designed circuit against power-analysis attacks. In order to configure the FPGA, to communicate with it, and to conduct measurements on it during the execution of the cryptographic algorithms, we need an evaluation board. There are many commercial FPGA evaluation boards which provide the ability of configuration of the FPGA and communication with it. In order to make the power consumption measurements, we have to interrupt the power line of the FPGA with a resistor or the line has to pass through a hole in the current probe. The only power line that can be reached is the power line of the evaluation board which shows the power consumption of all the devices on the board such as the FPGA, microcontroller, leds, etc. Hence, we can not make the FPGA board ready for measurements without damaging the board. Because of this reason we have designed our own FPGA board. Then we were free to design it such that gives us the best information about the dynamic power consumption of the FPGA.

As explained in Chapter 3, side-channel analysis attacks form important threats against hardware implementations of cryptographic algorithms. Hence it is natural to study the weaknesses of our circuits, presented in Chapter 4, against these attacks. We have implemented timing, power and electromagnetic analysis attacks on our FPGA implementations of elliptic curve cryptosystems over $GF(p)$. We conclude that our initial design was vulnerable to simple attacks: by using only timing information it is possible to find the Hamming weight of the key. More drastically, by using simple power and electromagnetic analysis attacks it was possible to find all the key bits. Next we improved our circuits such that they become immune to timing and simple power and electromagnetic analysis attacks. We showed that this improved design is vulnerable to differential attacks. We conduct differential power (DPA) and electromagnetic (DEMA) analysis attacks on our improved FPGA implementation of elliptic curve processor. We use two well known techniques for DPA and DEMA; correlation analysis and distance of mean test.

During the Advanced Encryption Standard (AES) [175] selection process, the security of Rijndael implementations [48] was evaluated with respect to all types of attacks. While being resistant to the classical cryptanalytic methods, it turned out that side channel attacks are a serious threat against naive im-

plementations of the Rijndael algorithm. To our knowledge, there exists no previous publication on practical implementations of power analysis attacks on dedicated hardware implementations of the AES. In this thesis we demonstrate the feasibility of power analysis attacks against hardware implementations of the AES. Our attacks target an ASIC implementation of the AES developed by the ETH Zürich [90] and an FPGA implementation of the AES developed by the UCL Crypto group [246].

In 1977, the Data Encryption Standard (DES) Algorithm [176] was adopted as a Federal Information Processing Standard for unclassified government communication. Although a AES was selected in October 2000, triple-DES is still widely used, particularly in the financial sector. In this thesis we describe a power analysis attack against an FPGA implementation of the DES developed by the UCL Crypto group [222].

1.2 Thesis Outline and Contributions

In Chapter 2 we introduce the notation and mathematical background on elliptic curves (ECs) and elliptic curve cryptosystems (ECCs). We also briefly introduce the block ciphers Data Encryption standard (DES) [176] and Advanced Encryption Standard (AES) [175].

In Chapter 3 we introduce the side-channel analysis attacks that we conduct on the hardware implementations of an ECC, the AES and the DES. We also summarize the previous work on the following side-channel attacks; TA attacks, PA attacks and EMA attacks.

In Chapter 4, we briefly explain the arithmetic operations over $GF\,(2^m)$ and $GF(p)$, respectively. The elements of $GF\,(2^m)$ can be represented in three different ways which are called bases. The performance of the most time and area consuming operation, multiplication, depends on the representation of the elements in $GF\,(2^m)$. Hence, we give an overview of these representations. Then, we present the most used multiplication architectures for different basis. Then, we summarize the pervious work on multiplications over $GF\,(2^m)$. We present our Montgomery modular multiplier designs over $GF\,(2^m)$ and $GF(p)$. Then, we use these implementations in our EC processors over $GF\,(2^m)$ and $GF(p)$. The results of this chapter are published in [196, 157, 159, 197, 198].

In Chapter 5 we describe the first realization of power-analysis attacks on a Virtex FPGA [201]. We can prove that this FPGA leaks a significant amount of information about its internal computations through the power supply lines. We can even provide evidence that the power consumption characteristics are comparable with the power consumption characteristics of ordinary application specific integrated circuits (ASICs).

In Chapter 6 we show a timing analysis attack against an FPGA implementation of ECC over $GF(p)$. Our initial design was vulnerable and it was possible to deduce the Hamming weight of the key by using the timing information. Then we improved our circuit and present the TA attacks on it. It is not possible to find out any information about the key by a TA attack on the improved design. We give the results of the simple power analysis attack on the same initial circuit. The SPA attack was successful and we could obtain all the bits of the key by using only one measurement. After improving our initial design, our elliptic curve processor became resistant against a SPA attack. We implemented a more powerful DPA attack on the improved implementation mentioned above and showed that it is possible to find the value of the MSB of the key. By using this information of the previous key bit we can implement the DPA attack targeting the next MSB. Hence, the DPA attack presented in this chapter reveals all the bits of the key. Also we present simple and differential electromagnetic analysis attacks on the same circuits. The SEMA attack is as successful as the SPA attack. In order to succeed with the DEMA attack we need more measurements for the DEMA attack than the DPA attack, because in our measurement setup the noise level of the electromagnetic radiation measurements is higher than for the power consumption measurements.

In Chapter 7 we demonstrate the feasibility of power analysis attacks against hardware implementations of the AES. Our attacks target against an ASIC implementation of the AES developed by the ETH Zürich [90] and an FPGA implementation of the AES developed by the UCL Crypto group [246]. In this chapter we describe a power analysis attack against an FPGA implementation of the DES developed by the UCL Crypto group [222]. These results allow us to compare the alternative ways of implementing the same algorithm with respect to side-channel attacks. The results of this chapter are published in [200, 199, 244, 245].

Chapter 2

Overview of Cryptographic Primitives

In this chapter, we introduce the notation and mathematical background on elliptic curves (ECs) and elliptic curve cryptosystems (ECCs). For a detailed exposure we refer to the cited literature [165, 126, 156, 53, 24]. We also briefly introduce the block ciphers Data Encryption Standard (DES) [176] and Advanced Encryption Standard (AES) [175].

In Section 2.1 we define ECCs. In order to explain how these cryptosystems work we explain the hard problem behind it and the mathematical principles of ECs. A major contribution of this thesis is the hardware design of two elliptic curve processors presented in Chapter 4. It is important that all the mathematical background is clear before introducing the hardware architectures that compute them. Section 2.1 forms the background information for the side-channel analysis (SCA) attacks on the field programmable gate array (FPGA) implementations of the elliptic curve processors presented in Chapter 6. In Section 2.3 and 2.4 we introduce the DES and AES block ciphers, respectively. We use the FPGA implementations of these block ciphers in our power analysis (PA) attacks in Chapter 7.

2.1 Elliptic Curve Cryptosystems

Elliptic curves were first proposed for use in public-key cryptography (PKC) by Koblitz in [126] and Miller in [165]. In order to introduce a public key cryptosystem based on elliptic curves, we first describe the elliptic curve group and then define the Elliptic Curve Discrete Logarithm Problem (ECDLP) which

is similar to the Discrete Logarithm Problem (DLP) for an arbitrary cyclic group.

2.1.1 Discrete Logarithm Problem

The security of many cryptographic techniques depends on the intractability of the discrete logarithm problem. A partial list of these is Diffie-Hellman key agreement and its derivatives [58], ElGamal encryption, and the ElGamal signature scheme and its variants [61, 62].

Definition 2.1 Let G be a finite cyclic group of order n. Let α be a generator of G, and let $\beta \in G$. The *discrete logarithm* (DL) of β to the base α, denoted $\log_\alpha \beta$, is the unique integer x, $0 \leq x \leq n-1$, such that $\beta = \alpha^x$. □

Definition 2.2 The *discrete logarithm problem* (DLP) is the following problem: given a prime p, a generator α of $\mathbb{Z}_p^*$, and an element $\beta \in \mathbb{Z}_p^*$, find the integer x, $0 \leq x \leq p-2$, such that $\beta = \alpha^x \bmod p$. □

2.1.2 The Diffie-Hellman Problem

The Computational Diffie-Hellman (CDH) problem is closely related to the DLP given in Section 2.1.1. It is important for public-key cryptography because its apparent intractability forms the basis for the security of many cryptographic schemes including Diffie-Hellman key agreement and its derivatives [58], and ElGamal public-key encryption [61, 62].

Definition 2.3 The *Computational Diffie-Hellman problem* (CDHP) is the following problem: given a prime p, a generator α of $\mathbb{Z}_p^*$, and elements $\alpha^a \bmod p$ and $\alpha^b \bmod p$, find $\alpha^{ab} \bmod p$. □

2.1.3 Diffie-Hellman Key Agreement

In 1976, Diffie and Hellman described a protocol whereby two people, A and B, can derive and share a common piece of secret information over an insecure communications channel in [58]. This protocol known as *Diffie-Hellman key agreement* can be described as follows:

1. A and B publicly select a finite group G and an element $\alpha \in G$.
2. A generates a random integer a, computes α^a in G and transmits α^a to B over a public communications channel.

3. B generates a random integer b, computes α^b in G and transmits α^b to A over the same channel.
4. A receives α^b and computes $\left(\alpha^b\right)^a$.
5. B receives α^a and computes $\left(\alpha^a\right)^b$.

A and B now share the common group element α^{ab}. Suppose that the DLP in $\mathbb{Z}_p^*$ could be efficiently solved. Then given α, p, $\alpha^a \bmod p$ and $\alpha^b \bmod p$, one could first find a from α, p, and $\alpha^a \bmod p$ by solving a DLP, and then compute $\left(\alpha^b\right)^a = \alpha^{ab} \bmod p$. This means the CDHP is solved which results that the third party C knows the common element shared between A and B.

2.1.4 Principles of Elliptic Curves

Definition 2.4 An *elliptic curve* (EC) over a field K is the set of solutions of the Weierstrass equation:

$$y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6 \tag{2.1}$$

over this field and the point at infinity $\mathcal{O}$. □

This set of solutions form an additive Abelian group with point addition as a group operation [156, 128, 24].

Let P_1 and P_2 be two points on an elliptic curve E. The line which goes through P_1 and P_2 intersects the curve in a third point Q. The point $P_1 + P_2$ is obtained by intersecting the line joining Q and the point at infinity $\mathcal{O}$ with E. If $P_1 = P_2$ the line between P_1 and P_2 is replaced by the tangent on E in P_1. Fig. 2.1 and Fig. 2.2 illustrate the group operation when E is given over $\mathbb{R}$.

Let $P = (x, y)$ be a point on the curve, its inverse $-P$ is given by $-P = (x, -y - a_1x - a_3)$.

Given two points $P_1 = (x_1, y_1)$ and $P_2 = (x_2, y_2)$, $P_1 \neq P_2$, the sum $P_3 = P_1 + P_2 = (x_3, y_3)$ can be computed as

$$\lambda = \begin{cases} \frac{y_1 - y_2}{x_1 - x_2} & P_1 \neq P_2 \\ \frac{3x_1^2 + 2a_2x_1 + a_4 - a_1y_1}{2y_1 + a_1x_1 + a_3} & P_1 = P_2 \end{cases}$$

$$x_3 = \lambda^2 - a_1\lambda - a_2 - x_1 - x_2,\ y_3 = (x_1 - x_3)\,\lambda - y_1 - a_1x_3 - a_3\,.$$

Let E over K be an elliptic curve given by the Weierstrass equation 2.1. If characteristic K is not 2 ($char(K) \neq 2$), then the admissible change of variables

$$(x, y) \rightarrow \left(x, y - \frac{a_1}{2}x - \frac{a_3}{2}\right)$$

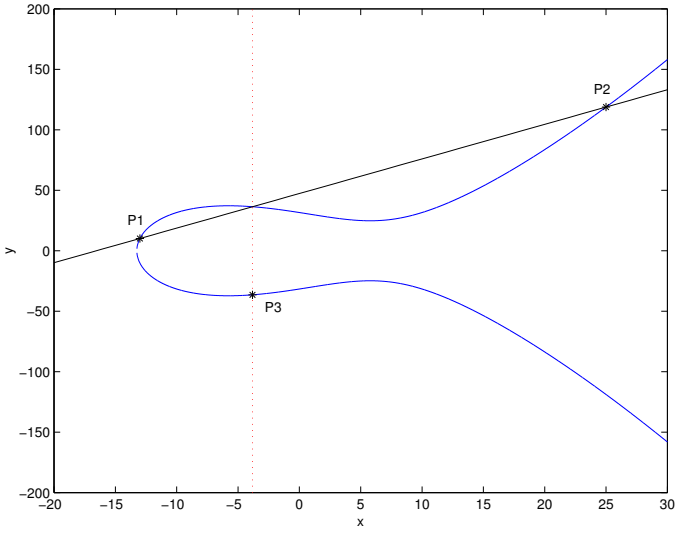


Figure 2.1: Adding two points on the elliptic curve over $\mathbb{R}$ given by the Weierstrass equation $y^2 = x^3 - 100 \cdot x + 6000$

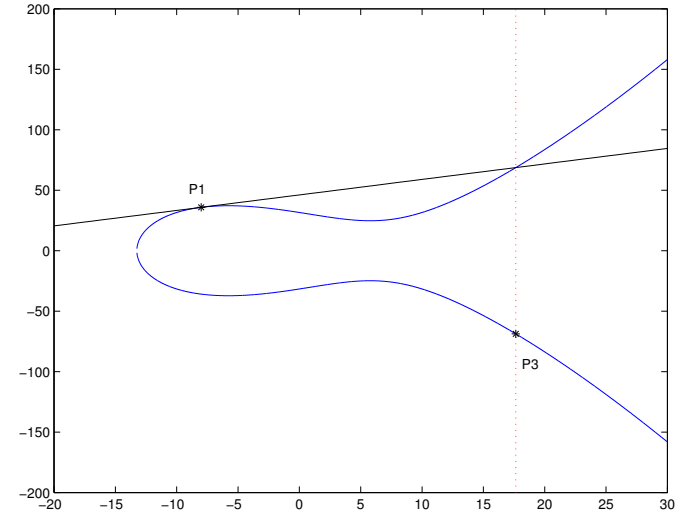


Figure 2.2: Doubling a point on the elliptic curve over $\mathbb{R}$ given by the Weierstrass equation $y^2 = x^3 - 100 \cdot x + 6000$

transforms E over K to the curve

$$E' : y^2 = x^3 + b_2 + b_4 + b_6.$$

Note that E is isomorphic to E' ($E \cong E'$) over K which applies that E and E' can be mapped onto each other, in such a way that to each point on E there is a corresponding point on E'. Formally, an *isomorphism* is a bijective map f such that both f and its inverse f^{-1} are structure-preserving mappings.

If $char(K) \neq 2, 3$, then the admissible change of variables

$$(x, y) \rightarrow \left(\frac{x - 3b_2}{36}, \frac{y}{216}\right)$$

further transforms E' to the curve

$$y^2 = x^3 + ax + b, \tag{2.2}$$

where $a, b \in K$ with $4a^3 + 27b^2 \neq 0$.

Let K be a field of characteristic 2, a quantity known as the *j-invariant* of E is defined as

$$j(E) = \frac{2^8 3^3 a_3^3}{4a_3^3 + 27a_2^2}.$$

If *j-invariant* of E ($j(E)$) is not 0, then the admissible change of variables

$$(x, y) \rightarrow \left(a_1^2 x + \frac{a_3}{a_1}, a_1^3 y + \frac{a_1^2 a_4 + a_3^2}{a_1^3}\right)$$

transforms E to the curve

$$y^2 + xy = x^3 + a_2 x^2 + a_6\,. \tag{2.3}$$

The basic operation for ECC algorithms is point or scalar multiplication, denoted as $Q = [k]P$, k is an integer, P and Q are EC points. The efficiency of point multiplication is mostly determined by the implementation of the finite field arithmetic. This operation can be calculated by using the double-and-add algorithm as shown in Algorithm 2.1.

There are many types of coordinates in which an elliptic curve can be represented. In the above equation affine coordinates are used, but so-called *projective coordinates* have some implementation advantages. The main conclusion is that point addition can be done in projective coordinates using only field multiplications, with no inversions required. Thus, inversions are deferred, and only one needs to be performed at the end of a point multiplication operation when converting back to affine coordinates [165, 126, 156, 24, 108]. Detailed information about projective coordinates can be found in Section 2.1.4.2 and 2.1.4.3.

Algorithm 2.1: Elliptic curve point multiplication

Require: EC point $P = (x, y)$, integer k, $0 < k < M$, $k = (k_{l-1}, k_{l-2}, \cdots, k_0)_2$, $k_{l-1} = 1$ and M
Ensure: $Q = [k]P = (x', y')$
1: $Q \leftarrow P$
2: **for** i from $l-2$ downto 0 **do**
3: $Q \leftarrow 2Q$
4: **if** $k_i = 1$ **then**
5: $Q \leftarrow Q + P$
6: **end if**
7: **end for**

2.1.4.1 Point Multiplication

One can visualize this operation in a hierarchical structure as follows. At the top is point multiplication. It is realized by means of repeated point additions and doublings. At the next (lower) level are these operations which are closely related to the coordinates used to represent the points. At the bottom level are finite field operations such as addition, subtraction, multiplication and inversion required to perform the group operation. Some results on implementation of elliptic curve point multiplication algorithm can be found in [166, 88, 43].

Point multiplication in elliptic curves is a special case of the general problem of exponentiation in Abelian groups. As such, it benefits from all the techniques available for the *shortest addition chain* problem [125].

Certain properties of elliptic curve can be taken into account to obtain faster algorithms. These properties are as follows:

1. Elliptic curve subtraction has the same cost as addition, hence the search space for fast algorithms can be expanded to include *shortest addition-subtraction chains* and *signed representations* (see Blake *et al.* [24]).
2. The relative complexities of general point addition and doubling have to be considered.
3. For certain families of elliptic curves, specific shortcuts are available that can significantly reduce the computational cost.

Specific work on elliptic curve point multiplication algorithms can be found in [170] by Morain and Olivos. Solinas explained how to use the non-adjacent form (NAF) for an elliptic curve point multiplication in [241].

2.1.4.2 Projective coordinates over $GF(2^m)$

A *projective plane*, denoted by P^2, is defined to be the set of equivalence classes of triples (X, Y, Z), not all zero. (X_1, Y_1, Z_1) and (X_2, Y_2, Z_2) are said to be equivalent if there exists a $\lambda \neq 0 \in GF(2^m)$ such that $X_1 = \lambda X_2$, $Y_1 = \lambda^2 Y_2$ and $Z_1 = \lambda Z_2$. Each equivalence class is called a projective point $(x, y, 1)$, where $x = X/Z$ and $y = Y/Z^2$. The *projective equation* derived from the affine equation Eq. (2.3) is given by $Y^2 + XYZ = X^3Z + aX^2Z^2 + bZ^4$.

To convert an affine point (x, y) to a projective point, one sets $X = x$, $Y = y$, $Z = 1$. To convert a projective point (X, Y, Z) to an affine point, we compute $x = X/Z$, $y = Y/Z^2$. The formulae for elliptic curve point addition and doubling are given by López and Dahab in [143].

Let $P = (X_1, Y_1, Z_1)$, $Q = (X_2, Y_2, Z_2)$ and $P + Q = R = (X_3, Y_3, Z_3)$. The addition formulas are the following ($P \neq \pm Q$):

$$
\begin{array}{lll}
A_1 = Y_2 \cdot Z_1^2\,, & A_2 = Y_1 \cdot Z_2^2\,, & B_1 = X_2 \cdot Z_1\,, \\
B_2 = X_1 \cdot Z_2\,, & C = A_1 + A_2\,, & D = B_1 + B_2\,, \\
E = Z_1 \cdot Z_2\,, & F = D \cdot E\,, & Z_3 = F^2\,, \\
G = D^2 \cdot \left(F + aE^2\right)\,, & H = C \cdot F\,, & X_3 = C^2 + H + G\,, \\
I = D^2 \cdot B_1 \cdot E + X_3\,, & J = D^2 \cdot A_1 + X_3\,, & Y_3 = H \cdot I + Z_3 \cdot J\,.
\end{array}
$$

These formulas can be improved for the special case $Z_1 = 1$ as follows:

$$
\begin{array}{lll}
A_2 = Y_1 \cdot Z_2^2\,, & B_2 = X_1 \cdot Z_2\,, & C = Y_2 + A_2\,, \\
D = X_2 + B_2\,, & F = D \cdot Z_2\,, & Z_3 = F^2\,, \\
G = D^2 \cdot \left(F + aZ_2^2\right)\,, & H = C \cdot F\,, & X_3 = C^2 + H + G\,, \\
I = D^2 \cdot X_2 \cdot Z_2 + X_3\,, & J = D^2 \cdot Y_2 + X_3\,, & Y_3 = H \cdot I + Z_3 \cdot J\,.
\end{array}
$$

The doubling formulas are the following ($R = 2P$):

$$
Z_3 = Z_1^2 \cdot X_1^2\,,\; X_3 = X_1^4 + b \cdot Z_1^4\,,\; Y_3 = bZ_1^4 \cdot Z_3 + X_3 \cdot \left(aZ_3 + Y_1^2 + bZ_1^4\right)\,.
$$

2.1.4.3 Projective Coordinates over $GF(p)$

We use the modified Jacobian coordinates as proposed by Cohen *et al.* in [43], because EC point doubling is the fastest in this representation. They represent internally the Jacobian coordinates as a quadruple $\left(X, Y, Z, aZ^4\right)$. This representation is called the modified Jacobian coordinate system and denoted by the authors as J^m. The algorithms for EC point addition and doubling are as follows [43].

Let $P = \left(X_1, Y_1, Z_1, aZ_1^4\right)$, $Q = \left(X_2, Y_2, Z_2, aZ_2^4\right)$ and $P + Q = R = (X_3, Y_3,$

$Z_3, aZ_3^4\big)$. The addition formulas in J^m are the following ($P \neq \pm Q$):

$$\begin{array}{l} U_1 = X_1 Z_2^2,\ U_2 = X_2 Z_1^2,\ S_1 = Y_1 Z_2^3,\ S_2 = Y_2 Z_1^3,\ H = U_2 - U_1, \\ r = S_2 - S_1,\ X_3 = -H^3 - 2U_1 H^2 + r^2, \\ Y_3 = -S_1 H^3 + r\left(U_1 H^2 - X_3\right),\ Z_3 = Z_1 Z_2 H,\ aZ_3^4 = aZ_3^4\,. \end{array} \tag{2.4}$$

The doubling formulas in J^m are the following ($R = 2P$):

$$\begin{array}{l} S = 4X_1 Y_1^2,\ U = 8Y_1^4,\ M = 3X_1^2 + \left(aZ_1^4\right),\ X_3 = -2S + M^2, \\ Y_3 = M(S - X_3) - U,\ Z_3 = 2Y_1 Z_1,\ aZ_3^4 = 2U\left(aZ_1^4\right)\,. \end{array}$$

2.1.5 Elliptic Curve Discrete Logarithm Problem

Now we introduce the *Elliptic Curve Discrete Logarithm Problem* (ECDLP); the ECC protocols are based on the difficulty of this problem. Let $E(GF(q))$ be an elliptic curve over $GF(q)$ and let $P \in E(GF(q))$ be a point of order n with $q = p^m$, p is prime. A point is of order n if $[n]P = \mathcal{O}$ and n is the smallest integer satisfying this equation. Let $Q \in E(GF(q))$, so that $Q = [k]P$ for some integer k, $0 \leq k < n$. The **Elliptic Curve Discrete Logarithm Problem (ECDLP)** is defined to be the problem of finding the number k for a given P and Q. This problem is believed to be hard *i.e.*, it is still unknown if there exists an algorithm to solve it in less than exponential time. Pollard's ρ method is the best general purpose algorithm known for solving the ECDLP [214]. The algorithm has an expected running time of $\sqrt{\pi n/2}$ elliptic curve operations.

State of the art suggest that EC in a subgroup of m-bit size is equivalent to $m/2$ bits symmetric key [78]. This will provide some protection against unforeseen developments in cryptanalysis of elliptic curves, without serious impact on performance. Curves over binary fields are often recommended to use slightly larger keys to mitigate certain hardware attacks [270]. Also, the generic attacks are in the binary case sometimes a bit more efficient. Curves over binary fields generally also require other forms of special attention when selecting parameters.

According to state of the art, the difficulty of solving DL in prime order fields of size 2^n is, up to constants, asymptotically equivalent to that of breaking n-bit RSA. In practice though, DL is noticeably more difficult. Moreover, DL is in most standardized algorithms performed in a smaller subgroup and it is then the size of this subgroup that matters, in which case the symmetric key equivalent is theoretically half of the bit-size of said subgroup, again according to the same generic attacks applicable also in the EC case. This would imply DL subgroups of the same size as a corresponding EC group. Note though that performing exponentiations over a finite field noticeably more expensive than on an elliptic curve of equivalent security. (The difference can be on the order 10-40 times, depending on security level.) Implementations which are

Table 2.1: NESSIE recommendations

Equivalent symmetric key size	56	64	80	112	128	160
Elliptic curve key size	112	128	160	224	256	320
Modulus length (pq)	512	768	1536	4096	6000	10000
Modulus length (p^2q)	570	800	1536	4096	6000	10000

concerned with performance could therefore, if required, reduce subgroup size by a few bits without lowering the security compared to the EC case.

Recently the NESSIE consortium, in [177], for "medium term" (5-10 years) security recommends the use of 1536-bit keys for RSA and DL based public key schemes, and 160-bit for elliptic curve discrete logarithms, suggesting a 1536/80 equivalence (depending cost model; HW/general purpose computer). This recommendation is based on an assumed equivalence between 512-bit RSA keys and 56-bit keys, and an extrapolation of that.

2.2 Elliptic Curve Digital Signature Algorithm

ElGamal first described how this DHP could be utilized in public-key encryption and digital signature schemes [62]. ElGamal's methods have been refined and incorporated into various protocols to meet a variety of applications, and one of its extensions forms the basis for the U.S. government Digital Signature Algorithm (DSA).

The DSA was proposed in August 1991 by the U.S. National Institute of Standards and Technology (NIST) and became a U.S. Federal Information Processing Standard (FIPS 186) in 1993. The FIPS 186 standard is also referred to as the Digital Signature Standard (DSS). It exploits small subgroups in $\mathbb{Z}_p^*$ in order to decrease the size of signatures. The key generation, signature generation, and signature verification procedures for DSA are given next.

DSA key generation. Each entity A does the following:

1. Select a prime q such that $2^159 < q < 2^160$.
2. Select a 1024-bit prime number p with the property that $q|p-1$. (The DSS mandates that p be a prime such that $2^{511+64t} < p < 2^{512+64t}$ where $0 \leq t \leq 8$. If $t = 8$ then p is a 1024-bit prime.)
3. Select an element $h \in \mathbb{Z}_p^*$ and compute $g = h^{(p-1)/q} \bmod p$; repeat until $g \neq 1$. (g is a generator of the unique cyclic group of order q in $\mathbb{Z}_p^*$.)
4. Select a random integer x in the interval $[1, q-1]$.

5. Compute $y = g^x \bmod p$.
6. A's public key is (p, q, g, y); A's private key is x.

DSA signature generation. To sign a message m, A does the following:

1. Select a random integer k in the interval $[1, q-1]$.
2. Compute $r = \left(g^k \bmod p\right) \bmod q$.
3. Compute $k^{-1} \bmod q$.
4. Compute $s = k^{-1}\{h(m) + xr\} \bmod q$, where h is the Secure Hash Algorithm (SHA-1).
5. If $s = 0$ then go to step 1.
6. The signature for the message m is the pair of integers (r, s).

DSA signature verification. To verify A's signature (r, s) on m, B should:

1. Obtain an authentic copy of A's public key (p, q, g, y).
2. Verify that r and s are integers in the interval $[1, q-1]$.
3. Compute $\omega = s^{-1} \bmod q$ and $h(m)$.
4. Compute $u_1 = h(m)\omega \bmod q$ and $u_2 = r\omega \bmod q$.
5. Compute $v = (g^{u_1} y^{u_2} \bmod p) \bmod q$.
6. Accept the signature if and only if $v = r$.

Since r and s are each integers less than q, DSA signatures are 320 bits in size. The security of the DSA relies on two distinct but related discrete logarithm problems.

The Elliptic Curve Digital Signature Algorithm (ECDSA) is the elliptic curve analogue of the DSA [117, 12]. That is, instead of working in a subgroup of order q in $\mathbb{Z}_p^*$, we work in an elliptic curve group $E(\mathbb{Z}_p)$. The ECDSA is currently being standardized within the ANSI X9F1, IEEE P1363, and ISO SC27 standards committees.

The key generation, signature generation, and signature verification procedures for ECDSA are given next.

ECDSA key generation. Each entity A does the following:

1. Select an elliptic curve E defined over $\mathbb{Z}_p$. The number of points in $E(\mathbb{Z}_p)$ should be divisible by a large prime n.
2. Select a point $P \in E(\mathbb{Z}_p)$ of order n.
3. Select a statistically unique and unpredictable integer d in the interval $[1, n-1]$.
4. Compute $Q = [d]P$.
5. A's public key is (E, P, n, Q); A's private key is d.

ECDSA Signature Generation. To sign a message m, an entity A with domain parameters $D = (q,\ FR,\ a,\ b,\ G,\ n,\ h)$ and associated key pair $(d,\ Q)$ does the following:

1. Select a random or pseudo-random integer k, $1 \leq k \leq n-1$.
2. Compute $kG = (x_1, y_1)$ and $r = x_1 \bmod n$. If $r = 0$ then go to step 1.
3. Compute $k^{-1} \bmod n$.
4. Compute $e = SHA-1(m)$.
5. Compute $s = k^{-1}(e + dr) \bmod n$. If s=0 then go to step 1.
6. A's signature for the message is (r, s).

ECDSA Signature Verification. To verify A's signature (r,s) on m, B obtains an authentic copy of A's domain parameters $D = (q,\ FR,\ a,\ b,\ G,\ n,\ h)$ and associated public key Q. It is recommended that B also validates D and Q. B then does the following:

1. Verify that r and s are integers in the interval $[1, n-1]$.
2. Compute $e = SHA-1(m)$.
3. Compute $w = s^{-1} \bmod n$.
4. Compute $u_1 = ew \bmod n$ and $u_2 = rw \bmod n$.
5. Compute $X = u_1 G + u_2 Q$. If $X = O$, then reject the signature. Otherwise, compute $\vartheta = x_1 \bmod n$ where $X = (x_1, y_1)$.
6. Accept the signature if and only if $\vartheta = r$

ANSI X9.62 mandates that $n > 2^160$. To obtain a security level similar to that of the DSA (with 160-bit q and 1024-bit p), the parameter n should have about 160 bits. If this is the case, then DSA and ECDSA signatures have the same size (320 bits).

2.3 Data Encryption Standard

In 1977, the Data Encryption Standard (DES) algorithm [176] was adopted as a Federal Information Processing Standard for unclassified government communication. Although a new Advanced Encryption Standard (AES, [175]) was selected in October 2000, triple-DES is still widely used, particularly in the financial sector. DES encrypts 64-bit blocks with a 56-bit key; its main operations are permutations, substitutions and XOR operations. DES is an iterated block cipher that applies 16 key-dependent transformations called rounds to the plaintext. This structure allows for very efficient hardware implementations.

The plaintext is first permuted by a fixed permutation initial permutation. Next the result is split into two 32-bit halves, denoted with L (left) and R (right) to which a round function is applied 16 times. The ciphertext is calculated by applying the inverse of the initial permutation to the result of the 16th round.

The secret key is expanded by the key schedule algorithm from 56 bits to sixteen 48-bit subkeys K_i; each round uses a different subkey K_i. The key schedule consists of bit permutations and rotations. As a consequence, if one can find any subkey, one can derive the complete key immediately (the missing 8 bits can be found by exhaustive search over 256 values).

Finally, the round function is represented in the grey part of Fig. 2.3(a); it can be described as follows:

$$\begin{aligned} L_{i+1} &= R_i \\ R_{i+1} &= L_i \oplus f(R_i, K_i), \quad i = 0, \ldots 15 \ . \end{aligned}$$

Here $L_{16} \| R_{16}$ is the ciphertext. The details of the nonlinear function f are provided in Fig. 2.3(b): the right part R_i is first expanded to 48 bits with the `Expansion`, $E(x)$, box, which duplicates some bits. Next, the 48-bit subkey K_i is added bitwise modulo 2 (XORed) to $E(R_i)$ and the result of the XOR function is sent to eight non-linear S-boxes (S). Each of them has six input bits and four output bits. The resulting 32 bits are permuted by the bit permutation P.

We have performed our experiments on a sequential DES implementation of that takes one clock cycle to perform one round [222]. It is represented in Fig. 2.3(a).

2.4 Advanced Encryption Standard

Rijndael is an iterated block cipher with a variable block length and a variable key length [48, 50, 52, 51]. The block length and the key length can be

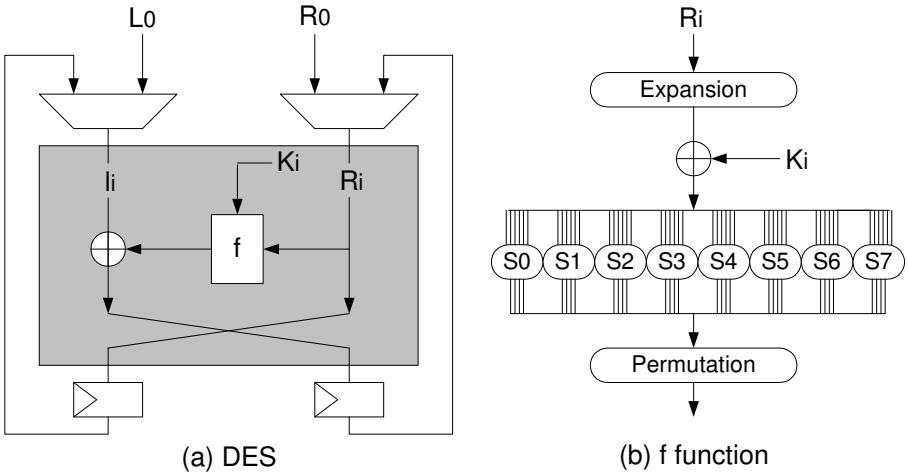


Figure 2.3: Block diagram of the Data Encryption Standard (DES)

independently specified to 128, 192 and 256 bits. Rijndael was chosen as the AES in October 2000. The AES has been published by the US government in November 2001 in [175]. Rijndael was designed to handle additional block sizes and key lengths, however the standard adopted only a 128 bit block length and key length of 128, 192 and 256 bits.

2.4.1 The State

Internally, the AES algorithm's operations are performed on a two-dimensional array of bytes called the **State**. The State consists of four rows of bytes, each containing Nb bytes, where Nb is the block length divided by 32.

At the start of the Cipher and Inverse Cipher, the input is copied into the State array. The Cipher or Inverse Cipher operations are then conducted on this State array, after which its final value is copied to the output.

2.4.2 Algorithm Specification

For both its Cipher and Inverse Cipher, the AES algorithm uses a round function that is composed of four different byte-oriented transformations: 1) byte substitution using a substitution table (S-box), 2) shifting rows of the State array by different offsets, 3) mixing the data within each column of the State array, and 4) adding a Round Key to the State.

```
Cipher(byte in[4*Nb], byte out[4*Nb], word w[Nb*(Nr+1)])
begin
    byte state[4,Nb]
    state = in
    AddRoundKey(state, w[0, Nb-1])
    for round = 1 step 1 to Nr1
        SubBytes(state)
        ShiftRows(state)
        MixColumns(state)
        AddRoundKey(state, w[round*Nb, (round+1)*Nb-1])
    end for
    SubBytes(state)
    ShiftRows(state)
    AddRoundKey(state, w[Nr*Nb, (Nr+1)*Nb-1])
    out = state
end
```

Figure 2.4: Pseudo code for the Advanced Encryption Standard (AES)

2.4.3 Cipher

At the start of the Cipher, the input is copied to the State array. After an initial Round Key addition, the State array is transformed by implementing a round function 10, 12, or 14 times (depending on the key length), with the final round differing slightly from the first $Nr - 1$ rounds. The final State is then copied to the output. The round function is parameterized using a key schedule that consists of a one-dimensional array of four-byte words derived using the Key Expansion routine. The Cipher is described in the pseudo-code in Fig. 2.4. The individual transformations – $SubBytes()$, $ShiftRows()$, $MixColumns()$, and $AddRoundKey()$ - process the State. In Fig. 2.4, the array $w[]$ contains the key schedule.

We focus on the 128-bit version. The algorithm consists of initial round which is adding the plaintext and the key and a series of 10 applications of a key-dependent round transformation to the cipher state and the round is composed of four different operations.

SubBytes is a non-linear byte substitution operating on each byte of the state independently. ShiftRows is a cyclic shift of the bytes of the state. In MixColumns, the columns (1 column = 4 bytes) of the state are considered as polynomials over $GF\left(2^8\right)$ and multiplied modulo $x^4 + 1$ with a fixed polynomial. Finally, AddRoundKey is a bitwise XOR with the bits of the key. AES's initial 128-bit key is expanded to eleven 128-bit round keys by means of a key scheduling algorithm.

2.5 Conclusions

One of the aims of this thesis is to present the hardware architectures for elliptic curve cryptosystems. In order to design a circuit which operates several operations, the mathematical background should be well understood. Hence, in this chapter we have introduced the notation and mathematical background on elliptic curves (ECs) and elliptic curve cryptosystems (ECCs). We have implemented several side channel attacks on our FPGA implementations of ECCs, that are described in Chapter 6. In order to compare the side channel resistance of our FPGA implementations of ECC and other systems, we have also implemented power analysis attacks on an ASIC and an FPGA implementations of the AES and an FPGA implementation of the DES. This chapter gives the mathematical background for these attacks.

Chapter 3

Side-Channel Analysis Attacks

Traditionally, the main task of cryptographic hardware is the acceleration of operations frequently used in cryptosystems or the acceleration of a complete cryptographic algorithm. In applications, hardware devices are also required to store secret or private keys securely. Hence, a cryptographic device must prevent the extraction of any sensitive information. A side-channel analysis (SCA) attack takes advantage of implementation specific characteristics to recover the secret parameters involved in the computation. It is therefore less general, but often more powerful than classical cryptanalysis.

Side-channel analysis attacks can be divided into two groups as *active* and *passive* attacks according to the ability of the attacker. Active attacks targeting the keys in cryptographic devices are commonly referred to as *tamper attacks*; they have a long history in the field of cryptography [11]. In these attacks the attacker has to reach the internal circuitry of the cryptographic device. There are two kinds: *probing attack* [135] and *fault induction attack* [26, 119]. A probing attack consists in inserting sensors into the device, in order to directly examine the content of memory zones or the data circulating on a bus. A fault induction attack works by disturbing the device's behavior in order to induce errors in the computation.

Passive attacks were recognized in the cryptographic community as a major threat in 1996, when the first article about timing analysis (TA) attacks [132] was published. In a passive attack, the adversary uses the standard functionality of the cryptographic device. The physical and/or electrical effects of the functionality of the device are then used for the attack. There are many different types of effects. If these effects unintentionally deliver information

about the key which is used inside the device, then they deliver side-channel information and are called side-channels.

Passive attacks are divided in four groups according to the side-channel information that they exploit. Timing analysis attacks exploit the timing information on the cryptographic hardware. Power analysis (PA) attacks use the dynamic power consumption of the cryptographic hardware during the execution of the cryptographic algorithm. Electromagnetic analysis (EMA) attacks use the electromagnetic radiation of the cryptographic hardware during the execution of the cryptographic algorithm. Acoustic (sound) analysis attacks exploit the sound coming out of the cryptographic hardware during the execution of the cryptographic algorithm [236].

All the groups of the passive attacks have two types. In a *simple analysis* attack, an attacker uses the side-channel information from one measurement directly to determine (parts of) the secret key. In *differential analysis* attack, many measurements are used in order to filter out noise. While a simple analysis attack exploits the relationship between the executed operations and the side-channel information, a differential analysis attack exploits the relationship between the processed data and the side-channel information.

In this chapter we introduce the passive attacks that we have conducted on the hardware implementations of an ECC, the AES and the DES. We also summarize the previous work on these side-channel attacks. We introduce and summarize the previous work on TA attacks in Section 3.2, on PA attacks in Section 3.3 and on EMA attacks in Section 3.4.

3.1 Theoretical Background

In differential analysis attacks, an attacker uses a so-called hypothetical model of the attacked device. The quality of this model is dependent on the knowledge of the attacker. The model is used to predict several values for the side-channel information of a device.

These predictions are compared to the real, measured side-channel information of the device. Comparisons are performed by applying statistical methods on the data. Among others, the most popular are the *distance-of-mean test* and the *correlation analysis*. We decided to use the correlation analysis in our attacks given in Chapter 6 and 7.

The frequency of the function generator in our measurement setup was changing during our measurements. In order to find the real clock frequency we have computed the discrete Fourier transform of each measurement. In the following sections we explain the basic mathematical tools used in SCA.

More theoretical considerations related to all side-channel attacks, but mainly inspired by work on EMA, are also given by Chari *et al.* in [40]. They discuss so-called template attacks in which the attacker uses a device that is identical to the target device. The only difference is that this replica is programmable which makes it possible to handle the noise in each measurement sample. Unlike previous considerations, which all try to eliminate noise, this approach attempts to model the noise in order to extract more useful information. As an example, they detail the template attack on RC4 using only one single measurement sample. Other case studies listed include DES and RSA exponentiation. At that moment, template attacks were considered to be the strongest possible SCA attacks from an information theoretical point of view, but the authors themselves came up with an even stronger approach afterwards. Namely, besides carefully exploring all available EM radiations an attacker can also focus on a combination of two or more side-channels. Agrawal *et al.* defined these so-called multi-channel attacks in which the side-channels are not necessarily of a different kind [8]. For example, they discussed combined power and EM analysis but also multi-channel DPA attacks. The former uses a CMOS leakage model and the maximum-likelihood principle for performing and analyzing. Agrawal *et al.* showed that it is even more effective than template attacks. Another example of a multi-channel attack is introduced by Walter and Thompson in [264]. They were the first to combine power and timing analysis.

3.1.1 Correlation Analysis

For the correlation analysis, the model predicts the amount of side-channel information for a certain moment of the execution. These predictions are correlated to the real side-channel information. This correlation can be measured with the Pearson correlation coefficient [42]. Let t_i denote the ith measurement data and T the set of measurements. Let p_i denote the prediction of the model for the ith measurement and P the set of such predictions. Then we calculate

$$C(T,P) = \frac{E(T \cdot P) - E(T) \cdot E(P)}{\sqrt{Var(T) \cdot Var(P)}} \quad -1 \leq C(T,P) \leq 1. \tag{3.1}$$

In Eq. (3.1), $E(T)$ denotes the expected (average) measurement data of the set of measurements T and $Var(T)$ denotes the variance of the set of measurements T. T and P are said to be uncorrelated, if $C(T,P)$ equals zero. Otherwise, they are said to be correlated. If their correlation is high, *i.e.*, if $C(T,P)$ is close to $+1$ or -1, it is usually assumed that the prediction of the model, and thus the key hypothesis, is correct.

3.1.2 Distance of Mean Test

A distance of mean test begins by running the cryptographic algorithm for N random values of input. For each of the N inputs, I_i, a discrete time side-

channel signal, $S_i[j]$, is collected and the corresponding output, O_i, may also be collected. The side-channel signal $S_i[j]$ is a sampled version of the side-channel output of the device during the portion of the algorithm that is being attacked. The index i corresponds to the I_i that produced the signal and the index j corresponds to the time of the sample. The $S_i[j]$ are split into two sets using a partitioning function, $D(\cdot)$:

$$\begin{aligned} S_0 &= \{S_i[j] \,|D(\cdot) = 0\} \\ S_1 &= \{S_i[j] \,|D(\cdot) = 1\}. \end{aligned}$$

The next step is to compute the average side-channel signal for each set:

$$\begin{aligned} A_0[j] &= \tfrac{1}{|S_0|} \textstyle\sum_{S_i[j] \in S_0} S_i[j] \\ A_1[j] &= \tfrac{1}{|S_1|} \textstyle\sum_{S_i[j] \in S_1} S_i[j], \end{aligned}$$

where $|S_0| + |S_1| = N$. By subtracting the two averages, a discrete time differential side-channel bias signal, $T[j]$, is obtained:

$$T[j] = A_0[j] - A_1[j].$$

Selecting an appropriate D function results in a differential side channel bias signal that can be used to verify guessed portions of the secret key.

3.1.3 Practical Challenges

When conducting power analysis attacks in practice, we need to deal with several technical difficulties. One of the most important issues is how to obtain *good*, *i.e.*, relatively noise free, measurements. The more noisy the obtained measurements are, the worse the statistical evaluation work and the more measurements are needed.

Another practical challenge is the complexity of the measurement setup. Such a setup typically consists of the attacked device, some monitoring tool (*i.e.*, the scope) and some tools to operate the attacked device (for example a smart card reader or a chip tester). In addition to the hardware components, we need several software tools that handle the communication between the hardware devices.

Hence, if one performs such an attack in practice, one needs to be sure that, if the analysis fails, this is because there is not enough side-channel information and not because there is a bug in one of the components of the measurement setup. Therefore, the first step in evaluating a device against side-channel attacks is to simulate attacks. As they are noise free, and they only involve some of the parts of the complete system, they allow to estimate how difficult a real attack will be.

3.1.4 Signal to Noise Ratio

In analog and digital communications, signal-to-noise ratio, often written S/N or SNR, is a measure of signal strength relative to background noise. The ratio is usually measured in decibels (dB).

If the incoming signal strength in μV is V_s, and the noise level, also in μV, is V_n, then the signal-to-noise ratio, SNR, in decibels is given by the formula

$$SNR = 20 \log_{10}(\frac{V_s}{V_n}) \tag{3.2}$$

If $V_s = V_n$, then $SNR = 0$. In this situation, the signal borders on unreadable, because the noise level severely competes with it. In digital communications, this will probably cause a reduction in data speed because of frequent errors that require the source (transmitting) computer or terminal to resend some packets of data.

Ideally, V_s is greater than V_n, so SNR is positive. $SNR = 20.0$ dB results in the signal being clearly readable. If the signal is much weaker but still above the noise for example $SNR = 2.28$ dB which is a marginal situation. There might be some reduction in data speed under these conditions.

If V_s is less than V_n, then SNR is negative. In this type of situation, reliable communication is generally not possible unless steps are taken to increase the signal level and/or decrease the noise level at the destination (receiving) computer or terminal.

3.1.5 Discrete Fourier Transform

Let x be a complex series with N samples of the form $x = x_0, x_1, \ldots, x_{N-1}$ where x_i is a complex number. The series outside the range 0, $N-1$ is extended in an N-periodic way, that is, $x_i = x_{i+N}$ for all i. Figure 3.1 shows the function $f(t) = \sin(2\pi{\cdot}2t)+2\sin(2\pi{\cdot}10t)+4\sin(2\pi{\cdot}100t)+6\sin(2\pi{\cdot}200t)+8\sin(2\pi{\cdot}300t)$, $0 \leq t \leq 0.5$ in the time domain. The function is sampled with a frequency of 1000 Hz. Hence the Nyquist frequency is 500 Hz [46]. The discrete Fourier transform (DFT) of x will be denoted as X; it will also have N samples. The forward transform is defined as

$$X_n = \frac{1}{N} \sum_{i=0}^{N-1} x_i e^{-jk2\pi n/N} \text{ for } n = 0 \cdots N-1$$

The DFT of $f(t)$ is shown in Fig. 3.2. The five different frequencies (2, 10, 100, 200 and 300 Hz) are clearly visible on this figure.

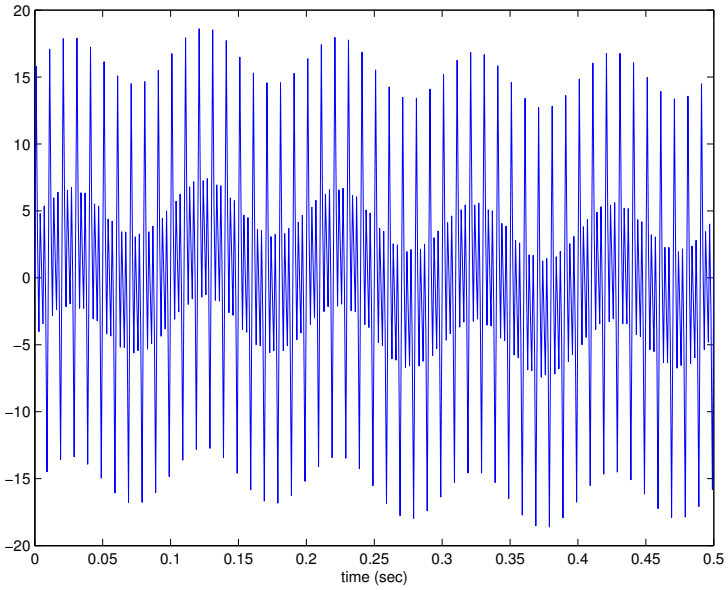


Figure 3.1: A sum of some sinus functions

3.2 Timing Analysis Attacks

It is not only the power consumption of a hardware or software system that may vary with the code sequence and processed data sets. Checking the differences in the processing time may in unsecured systems also retrieve secret information [132, 115].

An unsecured hardware or software system shows data dependencies due to differences in timing according to different operations executed. Addition and multiplication may be distinguished. Assume that we want to calculate the following operations $z = x + y$ and $z = x \times y$, with x and y which are m-bit binary numbers. The execution time of one of the implementations of the addition operation will take $T_A = m$ clock cycles. If we use this addition implementation as the basis for a multiplication implementation, then its execution time will be $T_M = \frac{3(m-1)m}{2}$. Hence, for the same bit-length operands, the one with shorter execution time will be an addition operation.

As the timing depends on the bit-length of the operands, by just using the timing information of one operation, even the big values with higher bit-length will be distinguished from the smaller ones with smaller bit-length. The same problem arises if the test of specific values and a following dependent branch in the program code is not secured.

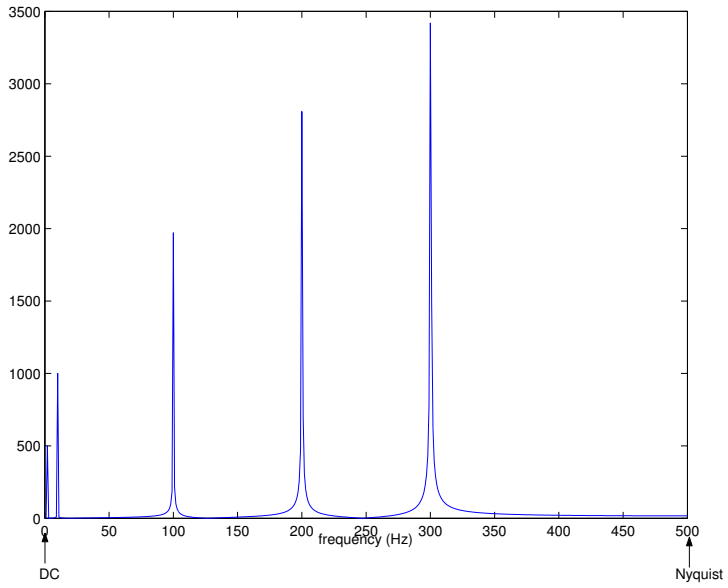


Figure 3.2: Discrete Fourier Transform (DFT) of the function in Fig. 3.1

3.2.1 Timing Analysis Attacks on Public Key Cryptosystems

TA attacks have been important significantly in the last few years. In June 1998, a TA attack could be performed on a smart card, compromising a software test code for the RSA [220] public key cryptosystem. After analyzing 300 000 timing tests, a 512-bit RSA key could be determined. The overall time for this attack has been specified to only a few minutes [56]. In this study, the individual bits of the RSA key were tested sequentially.

Schindler demonstrated a further evolution of TA attack, breaking the barriers of the RSA Chinese Remainder Theorem (CRT) [155] applications [229]. The attack can only be effectively performed if the so-called "Montgomery Algorithm" [168] is used for calculation of the RSA, and if CRT is used. This attack is improved by using an error-correction strategy in [93, 230].

Muir showed a TA attack on the function *NN-ModExp* for implementing modular exponentiation in RSAREF 2.0 [223] in [171]. He used the timing variation in the multiplications and the squares.

Cathalo *et al.* proposed a TA attack on the Girault, Poupard, Stern (GPS) signature scheme [79] in [37]. Their attack allows recovering the prover's private key provided the running time the exponentiation is dependent on the

Hamming weight of the exponent. In this scenario, the attacker impersonates the verifier and is able to measure precisely the computation time for the commitment step. Apart from this, the attacker has no knowledge of the implementation, such as the multiplication algorithm and the time needed for an individual multiplication.

3.2.2 Timing Analysis Attacks on Block Ciphers

Handschuh and Heys showed that the implementations of Rivest's RC5 [219] that take time for a rotation that is linear in the number of left shifts, are vulnerable to a TA attack [96]. The attack recovers the extended secret key table with only 2^{20} ciphertexts from the sole knowledge of the total amount of rotations carried out during the encryption.

Hevia and Kiwi studied the vulnerability of two implementations of the DES cryptosystem under a TA attack in [105]. They showed that a TA attack yields the Hamming weight of the key used by the DES implementations and that all the design characteristics of the target system could be inferred from timing measurements.

Koeune and Quisquater explained how to perform a TA attack on Rijndael in [134]. They used the fact that *MixColumn* operation can be implemented very efficiently and the execution time can be depend on the data processed.

The International Data Encryption Algorithm (IDEA) is a product block cipher designed by Lai, Massey, and Murphy [136]. IDEA can be cryptanalyzed with a piece of side-channel information: whether one of the inputs into one of the multiplications is zero. Since the multiplication is done modulo $2^{16} + 1$, a zero operand is treated as a special case. Some implementations bypass the multiplication completely and simply patch in the correct value. Kelsey *et al.* used this information and the ciphertexts for attacking the IDEA block cipher in [123].

3.2.3 Timing Analysis Attacks on Other Systems

While using Secure Shell (SSH) in interactive mode, every individual keystroke that a user types is sent to the remote machine in a separate internet protocol (IP) packet immediately after the key is pressed. Song *et al.* show that this property can enable the eavesdropper to learn the exact length of user's passwords in [242]. They have verified that the time it takes for the operating system to send out the packet after the key press is in general negligible comparing to the inter-keystroke timing. Hence an eavesdropper can learn the precise inter-keystroke timings from the arrival times of packets.

Brumley and Boneh developed a remote TA attack against open secure sockets

layer (OpenSSL) [189] in [32]. They measure the time an OpenSSL server takes to respond to decryption queries in their attack. They are able to extract the private key stored on the server.

3.2.4 Countermeasures

Most timing attacks exploit the modular reduction occurring in a Montgomery multiplication [168]. Therefore, Dhem in [55], Walter in [258, 261] and Hachez and Quisquater in [94] propose several countermeasures that typically consist of removing the time variation in this multiplication.

Kocher suggests a countermeasure consist of randomizing the exponent in RSA by adding a random multiple of $\varphi(n)$, a modification that does not effect the final result in [132].

Using square and multiply always algorithm during the exponentiation allows to hide the Hamming weight of the keys. Using double and add always algorithm proposed by Coron in [44] during the elliptic curve point multiplication allows to hide the Hamming weight of the keys. This countermeasure increases the computation time by about 30%.

As a second countermeasure against timing analysis attack on elliptic curve cryptosystems, Izu and Takagi propose the binary right to left point multiplication algorithm by executing point addition and doubling in parallel in [112, 113].

3.3 Power Analysis Attacks

Nowadays, complementary metal oxide semiconductor (CMOS) is by far the most commonly used technology to implement digital integrated circuits. The dominating factor for the power consumption of a CMOS gate is the dynamic power consumption [122]. For a single CMOS gate, we can express it as follows:

$$P_D = C_L V_{DD}^2 P_{0\to 1} f \,, \tag{3.3}$$

where C_L is the gate load capacitance, V_{DD} the supply voltage, $P_{0\to 1}$ the probability of a $0 \to 1$ output transition and f the clock frequency. Eq. (3.3) specifies that the power consumption of CMOS circuits is data-dependent. However, for the attacker, the relevant question is to know whether this data-dependent behavior is observable.

In order to show the effect of output transition on the power consumption of a gate, we have conducted experiments on some gates in a plastic package. We used a proto-board, a power supply, a function generator and an oscilloscope for these experiments. We have measured the power consumption of an inverter

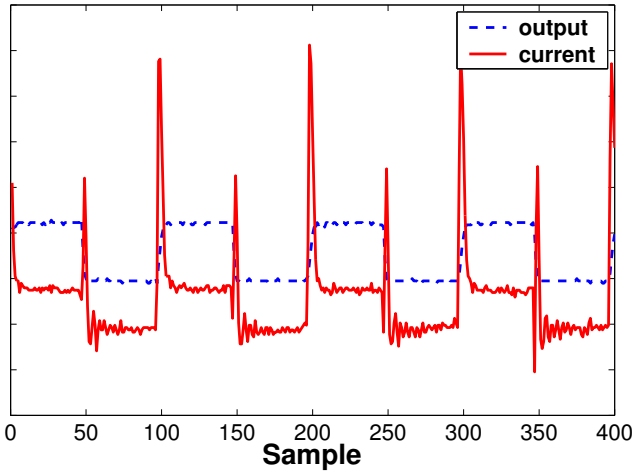


Figure 3.3: Current consumption trace of an inverter during some transitions of its output

and a D-flip flop while we changed the input from 0 to 1 and vice versa. The resulting power consumption traces of these experiments can be seen in Fig. 3.3 and Fig. 3.4. This (very partially) explains why information leaks when data bits change and why the power consumption trace is correlated with the number of transitions.

Two types of power consumption leakage that can be observed are the *transition count leakage* and the *Hamming weight leakage*. Transition count information leaks when the dominant source of the current is due to the switching of the gates. Thus, the more gates that change state, the more power that is dissipated. This effect can be seen on both falling and rising edges of the output of the inverter in Fig. 3.3. The power consumption seen by the measurement from the total power source of a hardware will depend on the total number of gates that change their states. As it can be seen from Fig. 3.3, $0 \rightarrow 1$ transitions have less effect than $1 \rightarrow 0$ transitions on the total power consumption of the hardware. Hence, this has to be taken account when a prediction for power consumption is calculated.

A situation where Hamming weight information leaks is when a pre-charged bus design is used. In this case, the number of zeros driven onto the pre-charged bus directly determines the amount of current that is being discharged. This effect can be seen on the falling edges of the output of the inverter shown in Fig. 3.3. As in the pre-charged bus, if the previous states of the outputs of some gates in the circuit are known and constant for every data, then the

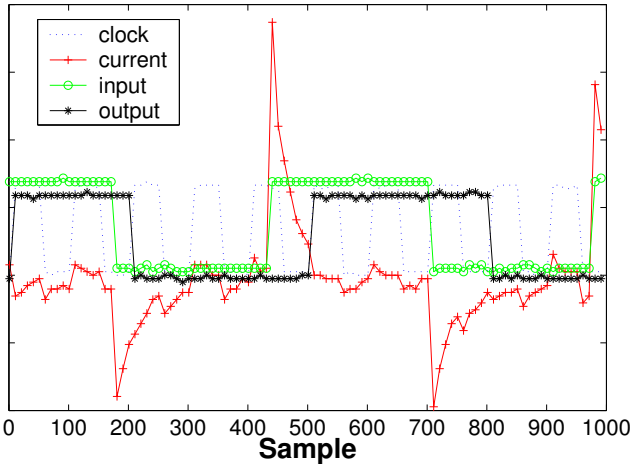


Figure 3.4: Current consumption trace of a D-flipflop during some transitions of its output

power consumption measured from the total power source will give information about the Hamming weight of the current state of these gates.

Two types of power analysis attacks are distinguished. In a *simple power analysis* (SPA) attack, an attacker uses the side-channel information from one measurement directly to determine (parts of) the secret key. In *differential power analysis* (DPA) attack, many measurements are used in order to filter out noise. While SPA exploits the relationship between the executed operations and the power leakage, DPA exploits the relationship between the processed data and the power leakage.

The first practical implementation of a power analysis attack on the DES was reported in [133] by Kocher *et al.*. Since then, several companies and universities have developed the skills to conduct these measurements in practice; these skills include knowledge about statistics, the properties of the attacked cryptographic algorithm, and the measurement setup.

3.3.1 Attacks and Countermeasures for Symmetric Key Cryptosystems

3.3.1.1 Attacks

Daemen *et al.* present power analysis results on bitslice ciphers in [47]. Mangard describes how to determine the complete secret key in Rijndael by using

Hamming weight information from a few sub-keys in [145]. Novak shows a side-channel attack on a substitution box of Rijndael, which is usually implemented as a table lookup operation in [180]. The attack is based on identifying equal intermediate results from power measurements while the actual values of these intermediates remain unknown.

Biham and Shamir present an attack which can determine the secret key of the DES uniquely by attacking several sub-keys in [22]. Fahn and Pearson introduce inferential power analysis (IPA) in [64]. An IPA attack is characterized by two stages, the first a long profiling stage and the second a simpler key extraction stage. The profiling step is typically based on comparisons of repeated parts of a selected cryptographic operation. These comparisons can be performed on a single cryptographic module, requiring many measured operations and result in a profile that can subsequently be used to extract keys from other modules using as little as a single cryptographic operation. Messerges *et al.* review and analyze the power analysis techniques used to attack DES in [163].

Megarajan proposes an attack based on the comparison of the repeated parts of an algorithm in [154]. Chari *et al.* describes an attack on SAFER++ [149] in [38]. They argue that at most half of the round keys need to be attacked in order to determine the secret key. They also propose attacks on the whitening process of Twofish [231].

3.3.1.2 Countermeasures

The goals of power analysis countermeasures are reducing the correlation between the power consumption data and the secret data and/or obscuring the power consumption measurements. There are two different types of countermeasures as software and hardware. Surveys about the countermeasures are given in [162, 27].

Software Countermeasures:

Time randomization was discussed in [39, 44, 49, 98, 99, 120, 141, 151, 206, 92, 113, 21]. In this type of countermeasures, the operations occur during random intervals of in an execution. This is done by using no-operations (NOPs), using dummy variables and instructions, data balancing (representation of the data is done in order to make the Hamming weight constant).

Permuting the execution (rearranged instructions) is proposed by Goubin and Patari in [86].

Masking techniques are studied in [86, 161, 45, 10, 85, 81, 256, 254]. Gomulkiewicz and Kutylowski show that masking is not always useful in [83]. The authors present an attack against an addition implementation, based on the observation of the Hamming weight of the sequence of carry that occur

during the bitwise addition. Apart from the efficiency of this attack, of more interest is the fact that this attack is not hindered by masking; in fact, the authors note that this could even make the attack easier.

In order to obtain DPA resistant applications, it cannot be tolerated that the software or hardware performs 'many' cryptographic operations on known inputs with the same secret information. Also, not 'too many' cryptographic operations should occur on the inputs that vary according to a known scheme with keys that vary according to a known scheme. Borst *et al.* demonstrated how to take countermeasures at the protocol level in [28, 27]. They proposed to use more key levels in a typical smart card application.

Hardware Countermeasures:

Increasing the measurement noise was the idea of Kocher *et al.* by a hardware noise generator as a random number generator (RNG) in [133]. The design of this approach may be relatively simple and it is an effective way to resist power analysis attacks. But it is expensive to implement and might be easy to disable through tampering and it is not energy efficient.

Shamir and Coron and Goubin proposed *power signal filtering* to obscure the measurements in [235] and [45], respectively. The design of this approach may be relatively simple and it is an effective way to resist attacks, but it requires a change to the hardware and might be easy to disable through tampering. Two types of filters were proposed as a *passive filter* in which physical limitations restrict the size of an on-chip capacitor and an *active filter* in which compensation techniques are likely to lag behind power supply changes. This countermeasure does not hide the electromagnetic radiation information of the device. The source of the electromagnetic radiation of the device is the internal current flow on the wires of the device and this countermeasure does not change the current flow which depends on the processed data.

There are also *novel circuit designs*. Shamir proposed detachable power supplies in [235]. Securing algorithm at the logic level was the idea of Tiri and Verbauwhede [253]. The method employs logic gates with a power consumption, which is independent of the data signals and therefore the technique removes the foundation for DPA. Asynchronous circuits are used as a countermeasure in [69, 169]. The power consumption and electromagnetic radiation are reduced, but the execution time depends on the data processed, so they are vulnerable to timing attacks. Golič used reversible logic in order to reverse computation which returns the consumed energy during the computation back to the circuit in [80].

Mangard has identified the hardware countermeasures that influence the number of samples needed in DPA attacks in [147]. Based on these properties, he proposed formulas that allow the calculation of lower bounds for the number

of samples needed in DPA attacks.

3.3.2 Attacks and Countermeasures for Public Key Cryptosystems

3.3.2.1 Attacks on Implementations of the Modular Exponentiation

After the description of power-analysis attacks by Kocher *et al.* [133], the first paper dedicated to the application of these types of attacks on public-key cryptosystems was from Messerges *et al.* [164]. They studied the application of PA attacks on software implementations of modular exponentiations in smart cards. They observed, that SPA attacks are possible due to several reasons. Firstly, when using the binary algorithm, the branching instruction itself could be observed. Secondly, in software implementations where two different routines for the square and the multiply operation are implemented, the two routines are likely to have different power traces.

Walter *et al.* observe in [264] that modular subtractions can also be used to determine the secret key. In [260] Walter describe an attack on an RSA secret key with single measurement; additionally he shows that certain implementations of sliding window techniques might be vulnerable to PA attacks.

In [179], Novak uses the power leakage of a smart card to determine the secret primes p and q for a certain implementation of the Chinese Reminder Theorem (CRT) exponentiation. In [54], a chosen plaintext attack on a CRT implementation is presented. Klima *et al.* discuss in [124] several side-channel attacks on various RSA schemes such as EME-OAEP PKCS #1 v.2.1. Three different attack variants, with different assumptions about the side-channel leakage are given, one of the variants can determine the RSA private key.

3.3.2.2 Attacks on Implementations of the Elliptic-Curve Scalar Point-Multiplication

Coron's work [44] was the first article on PA attacks dedicated solely to ECC. He presents attacks on the unprotected implementations of an Elliptic Curve Diffie-Hellman (ECDH) and an Elliptic Curve Integrated Encryption Scheme (ECIES) protocol.

Nguyen and Shparlinski pointed out that also the scalar point-multiplication in the Elliptic Curve Digital Signature Algorithm (ECDSA) must resist against PA attacks in [178]. The paper gives a rigorous treatment of this topic. Römer and Seifert treated the same topic with more heuristic arguments in [221].

Oswald developed another type of attack to attack countermeasures which are based on randomizing (variants of) the binary algorithm in [204]. This attack can defeat countermeasures that randomize the sequence of instructions in the

binary algorithm in a certain way [206] under the assumption that the power trace from an EC point-addition operation can be distinguished from the power trace of an EC point-doubling operation. Oswald also mounted another type of attack to defeat certain variants of these randomization approaches in [184]. It assumes that an attacker can monitor several executions of an EC scalar point-multiplication with the same scalar and it additionally assumes that EC point-addition and EC point-doubling can be distinguished.

3.3.2.3 Countermeasures for Modular Exponentiation

Countermeasures for Modular Exponentiation consist of either blinding the plaintext or blinding the exponent. Exponent blinding can be achieved by adding a multiple of the group order to the exponent. Efficient methods for this task have been proposed by Kocher in [132].

Messerges *et al.* also propose a countermeasure to prevent DPA attacks on modular exponentiations in [164]. They propose to randomize the starting point in the binary algorithm. From this random starting point on, the algorithm is either first computed upwards to the most significant bit (MSB) and then computed downwards to the least significant bit (LSB), or vice versa. The results of the computations of both directions are then combined.

Walter presents a randomized-window algorithm in [261]. An analysis consisting of two attack scenarios is given in [263].

3.3.2.4 Countermeasures for Elliptic Curve Scalar Point Multiplication

For elliptic curve scalar point multiplication there are two types of countermeasures against SPA and DPA. First of all, one has to randomize the expressions (*i.e.*, the coordinates) of calculated points. This can be achieve with randomized projective coordinates. Secondly, one has to obscure the multiplier. It would be optimal if there would be no correlation between the sequence of EC point-doubling (addition/subtraction) operations and the multiplier bits. Proposals for how to achieve these two goals have already been presented in [44]. Countermeasures applicable to arbitrary curves fixing the sequence of EC operations are given in [44, 167, 92, 113, 110, 255, 185, 186]. Countermeasures applicable to arbitrary curves not fixing the sequence of EC operations are given in [206, 31, 23, 68]. Countermeasures applicable to special curves fixing the sequence of EC operations are given in [98, 183, 182]. Countermeasures applicable to special curves not fixing the sequence of EC operations are given in [141, 118, 120, 41].

3.4 Electromagnetic Analysis Attacks

The sudden current pulse that occurs during the transition of the output of a CMOS gate, mentioned in Section 3.3, causes a sudden variation of the electromagnetic field surrounding the chip, that can be monitored by inductive probes which are particularly sensitive to the related impulse. The electromotive force across the sensor (Lentz' law) relates to the variation of magnetic flux as follows [234]:

$$V = -\frac{\mathrm{d}\phi}{\mathrm{d}t} \quad \text{and} \quad \phi = \iint \vec{B} \cdot d\vec{A},$$

where V is the probe's output voltage, ϕ the magnetic flux sensed by probe, t is the time, $\vec{B}$ is the magnetic field and $\vec{A}$ is the perpendicular area that it penetrates.

The Biot-Savart Law relates magnetic fields to the currents which are their sources. Finding the magnetic field resulting from a current distribution involves the vector product, and is inherently a calculus problem when the distance from the current to the field point is continuously changing.

$$\mathrm{d}\vec{B} = \frac{\mu_0 I \mathrm{d}\vec{L} \times \vec{\hat{r}}}{4\pi r^2},$$

where $\mathrm{d}\vec{L}$ is length of conductor carrying electric current I and $\vec{\hat{r}}$ is unit vector to specify the direction of the vector distance r from the current to the field point.

Electromagnetic radiation itself consists of two components, the electrical and magnetic field vectors [106]. In theory, both components can be measured individually or in their interaction. Capacitive sensors mainly capture the electrical field components, while antennas and coils are able to acquire both electrical and magnetic components, and Hall sensors and so-called "SQUIDS" (super conducting quantum interference devices) mainly detect the pure magnetic field components.

Two types of electromagnetic analysis attacks are distinguished. In a *simple electromagnetic analysis* (SEMA) attack, an attacker uses the side-channel information from one measurement directly to determine (parts of) the secret key. In a *differential electromagnetic analysis* (DEMA) attack, many measurements are used in order to filter out noise. While SEMA exploits the relationship between the executed operations and the electromagnetic radiation, DEMA exploits the relationship between the processed data and the electromagnetic radiation.

3.4.1 Attacks

It is well known that the United-States government has been aware of electromagnetic leakage since the 1950s. The resulting standards are called TEMPEST; partially declassified documents can be found in [181]. The first published papers are work of Quisquater and Samyde [215] and the Gemplus team [72]. Quisquater and Samyde showed that it is possible to measure the electromagnetic radiation from a smart card. Their measurement setup consisted of a sensor which was a simple flat coil, a spectrum analyzer or an oscilloscope and a Faraday cage. Quisquater and Samyde also introduced the terms Simple EMA (SEMA) and Differential EMA (DEMA). The work of Gemplus deals with experiments on three algorithms: DES, RSA and COMP128. They observed the feasibility of EMA attacks and compared them with PA attacks in favor of the first. Namely, EM emanation can also exploit local information and, although more noisy, the measurements can be performed from a distance. This fact broadens the spectrum of targets to which SCA attacks can be applied. They are not limited to smart cards and similar tokens but also include SSL accelerators and many other cryptographic devices.

According to Agrawal *et al.* there are two types of emanations: *intentional* and *unintentional* [7, 6]. The first type results from direct current flows. The second type is caused by various couplings, modulations (amplitude modulation (AM) and frequency modulation (FM)), etc. The two papers mentioned above deal exclusively with intentional emanations. To the contrary, the real advantage over other SCA attacks lies in exploring unintentional emanations [7, 6]. More precisely, electromagnetic (EM) leakage consists of multiple channels. Therefore, compromising information can be available even for DPA resistant devices which can be detached from the measurement equipment.

Mangard showed that near-field EM attacks can be conducted even with a simple hand-made coil in [146]. Also he showed that measuring the far-field emissions of a smart card connected to a power supply unit also suffices to determine the secret key used in the smart card.

Carlier *et al.* showed that EM side channels from an FPGA implementation of AES can be effectively used by an attacker to retrieve some secret information in [36]. They worked close to the FPGA and by this way were able to get rid of the effects of other computations made at the same time. They also introduced a new Square EM Attack.

Up to now, most papers on EMA applied similar techniques as power analysis while apparently much more information is available to be explored. It is likely that future work will also deal with combinations of EMA with other side channel attacks.

3.4.2 Countermeasures

Very few articles describe countermeasures against an EMA analysis. A complete shielding of Smart Card controllers, known from devices used in electronic data processing, is possible, but an attacker could simply remove the shield prior to analysis, making this countermeasure worthless [106].

With these presumptions in mind, EMA countermeasures have to reach much further than the commonly known PA defense systems, due to the fact that EMA attacks may provide information about small chip areas, whereas the PA measurement only yields data concerning the supply current of the complete chip.

3.5 Acoustic Analysis Attacks

Recently, Shamir and Tromer present their results using the sound of a central processing unit (CPU) as a side-channel information in [236]. The oldest eavesdropping channel, namely acoustic emanations, has received little attention. Shamir and Tromer's preliminary analysis of acoustic emanations from personal computers shows them to be a surprisingly rich source of information on CPU activity.

Several desktop and laptop computers have been tested and in all cases it was possible to distinguish an idle CPU from a busy CPU. For some computers, it was also possible to distinguish various patterns of CPU operations and memory access. This can be observed for artificial cases (e.g., loops of various CPU instructions), and also for real-life cases (e.g., RSA decryption).

A low-frequency (KHz) acoustic source can yield information on a much faster (GHz) CPU in two ways. First, when the CPU is carrying out a long operation, it may create a characteristic acoustic spectral signature. Second, temporal information about the length of each operation is learnt and this can be used to mount TA, especially when the attacker can affect the input to the operation.

One obvious countermeasure is to use sound dampening equipment, such as "sound-proof" boxes, that is designed to sufficiently attenuate all relevant frequencies. Conversely, a sufficiently strong wide-band noise source can mask the informative signals, though ergonomic concerns may render this unattractive. Careful circuit design and high-quality electronic components can probably reduce the emanations. Alternatively, one can employ known algorithmic techniques to reduce the usefulness of the emanations to attacker. These techniques ensure the rough-scale behavior of the algorithm is independent of the inputs it receives; they usually carry some performance penalty, but are often already used to thwart other side-channel attacks.

3.6 Conclusions

In this chapter we have introduced the side-channel analysis attack. We have presented four groups of the SCA attacks according to four types of side-channel information: timing, power consumption, electromagnetic radiation and sound. We have given the definition of the two different types of SCA attacks; simple and differential. The previous work and the countermeasures on the SCA analysis have been also presented in this chapter.

This chapter provides the background information for the timing, power and electromagnetic analysis attacks on our FPGA implementation of elliptic curve processor over $GF(p)$ presented in Chapter 6. We have also used the information provided in this chapter for the power analysis attacks on the hardware implementations of the AES and the DES in Chapter 7.

Chapter 4

Hardware Implementations of Elliptic Curve Cryptosystems

Elliptic Curve Cryptography (ECC) was proposed independently by Miller [165] and Koblitz [126] in the 1980s. Since then a considerable amount of research has been performed on secure and efficient ECC implementations. The benefits of ECC, when compared with classical cryptosystems such as RSA [220], include: higher speed, lower power consumption and smaller certificates, which are especially useful for wireless applications.

In this chapter we present our FPGA implementations of processors for ECC over finite field $GF(p)$, p is prime and $GF\,(2^m)$, m is prime. Our results were published in [197, 198, 159]. The performance of an elliptic curve cryptosystem and of several other public key cryptosystems is mostly determined by the efficient implementation of finite field arithmetic. The most critical operation for latency is modular multiplication. In 1985 Montgomery introduced a new method for modular multiplication [168]. The approach of Montgomery avoids the time consuming trial division that is a common bottleneck of other algorithms. His method is proven to be very efficient and is the basis of many implementations of modular multiplication, both in software and hardware [19]. In this chapter, we present our FPGA implementations of the Montgomery modular multiplication (MMM) over $GF\,(2^m)$ and $GF(p)$. The designs have systolic array architectures to allow pipelining and to make the clock frequency independent of m and the bit-length of p. In this way, the clock frequency does not change when the bit-length is enlarged for security reasons. These results were published in [196, 198] for $GF(p)$ and in [157] for $GF\,(2^m)$.

Our elliptic curve processors consist of special operational blocks for MMM, modular addition/subtraction (MAS), EC point doubling/addition, modular multiplicative inversion, EC point multiplier, projective to affine coordinates conversion and Montgomery to normal representation conversion. Hence it can be programmed by the user to execute any of these operations in any order. It is possible to use the proposed processor over $GF(p)$ not only for ECC, but also for any system for which modular arithmetic operations are essential, such as the RSA cryptosystem.

The basic building blocks of our designs are MMM and MAS. The other blocks include some finite state machines (FSMs), which control the execution of these operations. The critical path depends only on the critical path of the circuits for MMM and MAS. The architecture of these blocks is designed to ensure a short critical path to allow for high clock frequencies which are independent from the bit-length of the EC parameters. For simplicity, all blocks were designed separately with their own FSMs; this allows for independent optimization and testing of the building blocks.

In this chapter, we briefly explain the arithmetic operations over $GF\left(2^m\right)$ and $GF(p)$, respectively. It is possible to represent the elements of $GF\left(2^m\right)$ in three main different ways which are called basis. The performance of the most time and area consuming operation, multiplication, depends on the representation of the elements in $GF\left(2^m\right)$. Hence, we give an overview of these representations. Then, we present the most used multiplication architectures for different basis. Then, we summarize the pervious work on multiplications over $GF\left(2^m\right)$. We present our Montgomery modular multiplier designs over $GF\left(2^m\right)$ and $GF(p)$. Then, we use these implementations in our EC processors over $GF\left(2^m\right)$ and $GF(p)$.

4.1 Arithmetic Operations over $GF\left(2^m\right)$

In this section we introduce the arithmetic operations over $GF\left(2^m\right)$ used in ECC over $GF\left(2^m\right)$. These operations determine the performance of an ECC and of several other public key cryptosystems. Hence, the hardware design for the public key cryptosystems starts with the hardware design for finite field operations.

4.1.1 Representation of the Elements in $GF\left(2^m\right)$

The most critical operation for latency is modular multiplication and its performance depends on the representation of the elements in $GF\left(2^m\right)$. It is possible to represent the elements of $GF\left(2^m\right)$ different ways which are called basis. In the following sections, we give an overview of these representations.

4.1.1.1 Polynomial (or Standard or Canonical) Basis

Definition 4.1 The *polynomial basis (PB)* is given by the set $\left\{1, x, x^2, \cdots, x^{m-1}\right\}$ where x is a root of the prime polynomial $P(x)$ of degree m used to construct $GF(q^m)$ from $GF(q)$. □

According to this representation an element a of $GF(q^m)$ is a polynomial with length m, written as

$$a(x) = \sum_{i=0}^{m-1} a_i x^i = a_{m-1}x^{m-1} + \cdots + a_1 x + a_0 \,,$$

where the coefficients $a_i \in GF(q)$. In the word-level representation, these coefficients are grouped into s words of equal length. Let ω be the word length and $m = s \cdot \omega$. Hence, a can be written as $a(x) = \sum_{i=0}^{s-1} A_i(x)x^{i\omega}$, where each polynomial $A_i(x)$ corresponds to a word of length ω, or

$$A_i(x) = a_{i\omega+\omega-1}x^{\omega-1} + \cdots + a_{i\omega+1}x + a_{i\omega} \,. \tag{4.1}$$

4.1.1.2 Dual Basis

Definition 4.2 Let $F = GF(2)$, $K = GF(2^m)$ and $a \in K$. The *trace* of a relative to the subfield F is:

$$Tr_F^K(a) = a + a^2 + a^{2^2} + \cdots + a^{2^{m-1}} \,.$$

□

Definition 4.3 The set $\{x_0, x_1, \cdots, x_{m-1}\}$ is a *basis* for $K = GF(2^m)$ over $F = GF(2)$, such that $a^{2^i} = x_i$, $a \in K$, $0 \leq i \leq m-1$. So the elements of the set are linearly independent over F. □

Definition 4.4 The corresponding *dual basis (DB)* is: $\{y_0, y_1, \cdots, y_{m-1}\} \subseteq GF(2^m)$ such that:

$$Tr(x_i y_j) = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{if } i \neq j \end{cases} \tag{4.2}$$

□

4.1.1.3 Normal Basis

Definition 4.5 Let x be a root of the prime polynomial $P(x)$ of degree m used to construct $GF(q^m)$ from $GF(q)$. Then the set $\left\{x, x^q, \cdots, x^{q^{m-1}}\right\}$ forms a *normal basis (NB)*. □

4.1.1.4 Triangular Basis

Definition 4.6 Let $\{x_0, x_1, \cdots, x_{m-1}\}$ be a PB. A *triangular basis*, denoted as $\{z_0, z_1, \cdots, z_{m-1}\}$, is derived as follows:

$$z_i = \sum_{j=i}^{m-1} p_{j+1} x^{j-i} \quad 0 \leq i \leq m-1\,,$$

where p_is are the coefficients of an irreducible polynomial $P(x)$. □

4.1.1.5 Other Basis Representations

Parker and Benaissa presented multiplier architectures for *redundant basis representation* (RBR) in [213]. RBR was proposed by Parhami in [211]. Wu *et al.* presented multiplication, conversion and squaring architectures for finite field computation using RBR in [278]. Drolet proposed a new representation for finite field elements called *polynomial ring representation* (PRR) in [59]. He also presented architectures for a parallel and serial multiplier, exponentiation, inversion and division architectures with PRR. Silverman proposed a new representation for finite field elements in 1999 [238]. He called this method *Ghost Bit Basis* (GBB). There is no multiplier hardware structure for this representation so far.

4.1.2 Addition/Subtraction

The addition or subtraction of two elements a and b in $GF\,(2^m)$ is performed by adding the polynomials $a(x)$ and $b(x)$, where the coefficients are added in the field $GF(2)$. This is equivalent to a bit-wise XOR operation on the vectors a and b.

4.1.3 Multiplication

In order to multiply two elements a and b in $GF\,(2^m)$, we need to select an irreducible polynomial of degree m. Note that a different choice of polynomial leads to a different finite field representation, but all finite fields with the same number of elements are isomorphic. Different choices of irreducible polynomials are discussed in [155] by Menezes *et al.* Let $P(x)$ be an irreducible polynomial of degree m over the field $GF(2)$, hence $p_m = 1$ and $p_0 = 1$. The product $c = a \cdot b$ in $GF\,(2^m)$ is obtained by computing $c(x) = a(x)b(x) \bmod p(x)$, where $c(x)$ is a polynomial of length m, representing the element $c \in GF\,(2^m)$. Mastrovito's thesis from 1991 [150] serves as an extensive reference of hardware architectures for performing $GF\,(2^m)$ multiplication.

4.1.3.1 Serial Polynomial Basis Multiplier

In this section an architecture for bit-serial computation of products of two elements of $GF(2^m)$ that are represented by PB will be described. This architecture is called a *serial shift register (SSR)* multiplier.

Let $A(x) = a_0 + a_1 x + \cdots + a_{m-1}x^{m-1}$ and $B(x) = b_0 + b_1 x + \cdots + b_{m-1}x^{m-1}$ be the two field elements that will be multiplied. $C(x) = c_0 + c_1 x + \cdots + c_{m-1}x^{m-1}$ is the result of the multiplication. Then

$$\begin{aligned} C(x) &= A(x)B(x) \bmod P(x) \\ &= \left[b_0 A(x) + b_1 x A(x) + \cdots + b_{m-1}x^{m-1}A(x)\right] \bmod P(x)\,. \end{aligned} \tag{4.3}$$

Each term $b_i x^i A(x)$ in Eq. (4.3) can be computed recursively in the following way

$$\begin{aligned} &b_i x^i A(x) \bmod P(x) = \\ &\quad (\cdots((b_i x A(x) \bmod P(x))\, x \bmod P(x))\cdots)\, x \bmod P(x)\,. \end{aligned}$$

for $i = 0, 1, \cdots, m-1$. So $C(x)$ can be written as follows:

$$\begin{aligned} C(x) = \quad & (((b_{m-1}xA(x) + b_{m-2}A(x))\, x + b_{m-3}A(x))\, x + \cdots)\, x \\ & + b_0 A(x) \bmod P(x)\,. \end{aligned} \tag{4.4}$$

The block diagram of a SSR multiplier is shown in Fig. 4.1. The product $C(x)$ is found in the $C = [c_{m-1}, c_{m-2}, \cdots, c_0]$ register after m clock cycles. A schematic view of a SSR multiplier cell is shown in Fig. 4.2. The multiplier operates as follows. The C register is initially set to zero. During the first clock cycle, the polynomial $b_{m-1}A(x)$ is loaded to the C register. On the next clock cycle both C and B registers are shifted one position left. The A and P registers are unchanged. Shifting Z to left is equivalent to multiplication by x in Eq. (4.4) and since this operation can result in a term of degree greater than $m-1$, the *FB_IN* and *FB_OUT* signals in Fig. 4.1 and 4.2 indicate a reduction should be whether or not performed. Also during the second clock cycle the polynomial $b_{m-2}A(x)$ is added to the previous result $b_{m-2}A(x)x \bmod P(x)$.

Properties of SSR Multiplier: The length of the critical path is 4 (one register, one AND-gate and two XOR-gates), independently of the choice of $P(x)$. This implies that the SSR allows the same clock frequency for any m. The number of clock cycles needed to complete one multiplication is m. The hardware complexity is $O(m)$, exactly $4m$-bit register and $7m$ gates.

4.1.3.2 Parallel Polynomial Basis Multiplier

Mastrovito proposed in his thesis [150] a parallel multiplier which is named after him. The architecture can be explained as follows.

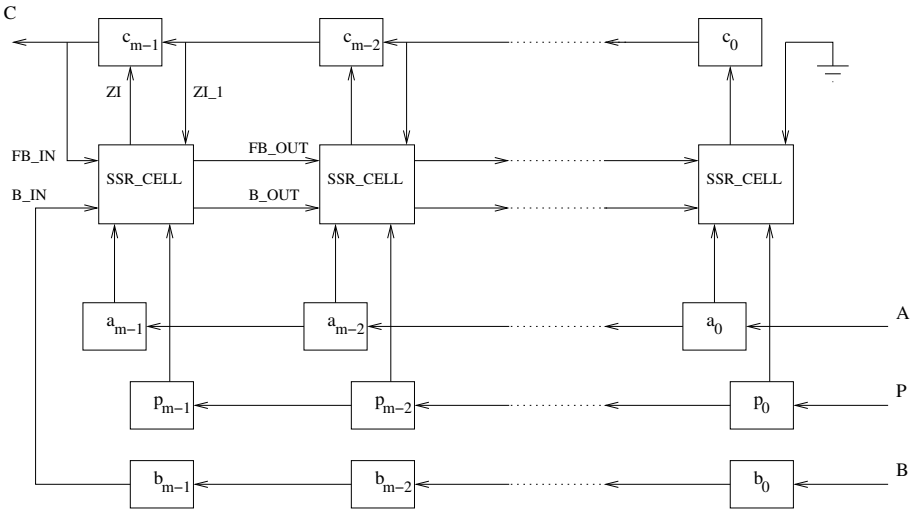


Figure 4.1: Block diagram of the serial shift register multiplier over $GF\,(2^m)$

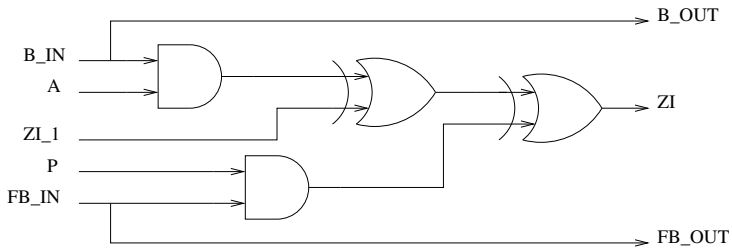


Figure 4.2: Schematic view of a cell of the serial shift register multiplier over $GF\,(2^m)$

The product $C(x) = A(x)B(x) \bmod P(x)$ can be written in matrix form as $C = ZB$, where Z is a binary m by m matrix. The entry $z_{i,j}$ of Z is denoted by $f_{i,j}(A)$ or simply $f_{i,j}$. The elements of $C(x)$ can be written as follows:

$$c_i = b_0 f_{i,0}(A) + b_1 f_{i,1}(A) + \cdots + b_{m-1} f_{i,m-1}(A)\,, \tag{4.5}$$

or in matrix notation:

$$C = \begin{pmatrix} f_{0,0} & f_{0,1} & \cdots & f_{0,m-1} \\ f_{1,0} & f_{1,1} & \cdots & f_{1,m-1} \\ \vdots & \vdots & & \vdots \\ f_{m-1,0} & f_{m-1,1} & \cdots & f_{m-1,m-1} \end{pmatrix} \begin{pmatrix} b_0 \\ b_1 \\ \vdots \\ b_{m-1} \end{pmatrix} = ZB\,.$$

It is desirable to have explicit formulas for the functions $f_{i,j}$. Next, an $m-1$ by m binary matrix Q which is called the *reduction matrix* is defined as:

$$\begin{pmatrix} x^m \\ x^{m+1} \\ \vdots \\ x^{2m-2} \end{pmatrix} = Q \begin{pmatrix} 1 \\ x \\ \vdots \\ x^{m-1} \end{pmatrix} = \begin{pmatrix} q_{0,0} & q_{0,1} & \cdots & q_{0,m-1} \\ q_{1,0} & q_{1,1} & \cdots & q_{1,m-1} \\ \vdots & \vdots & & \vdots \\ q_{m-1,0} & q_{1,m-1} & \cdots & q_{m-1,m-1} \end{pmatrix} \begin{pmatrix} 1 \\ x \\ \vdots \\ x^{m-1} \end{pmatrix} .$$

The reduction matrix is obtained from $P(x)$ by a simple shift & add-on-overflow procedure. Because,

$$\begin{aligned} x^m \bmod P(x) &= p_{m-1}x^{m-1} + \cdots + p_1 x + p_0 \\ &= q_{0,0} + q_{0,1}x + \cdots + q_{0,m-1}x^{m-1} . \end{aligned}$$

It follows that $q_{0,i} = p_i$ for $i = 0, 1, \cdots, m-1$. For $1 \le i \le m-1$ we have that

$$\begin{aligned} q_{i,0} &= q_{i-1,m-1}q_{i-1,0} \\ q_{i,1} &= q_{i-1,m-1}q_{i-1,1} + q_{i-1,0} \\ &\vdots \\ q_{i,m-1} &= q_{i-1,m-1}q_{i-1,m-1} + q_{i-1,m-2} . \end{aligned}$$

Mastrovito then shows that the elements $f_{i,j}$ in Eq. (4.5) can be written as follows:

$$f_{i,j} = \sigma(i-j)\, a_{i-j} + \sum_{t=0}^{j-1} q_{j-1-t,i} a_{m-1-t} ,$$

where $\sigma(k)$ is a step function defined by

$$\sigma(k) = \begin{cases} 1 & k \ge 0 \\ 0 & k < 0 \end{cases} .$$

The multiplier is divided into two subsystems. The first subsystem computes the functions $f_{i,j}$ and is called the f-network. The second subsystem is called the IP network and consists of m identical cells.

Properties of the Parallel PB Multiplier: the total complexity (in gates) is denoted by C and the length of the critical path through the whole multiplier is denoted by L.

$2 + \lceil log_2 m \rceil \le L \le 1 + 2\lceil log_2 m \rceil$,

$1 + m(2m-1) \le C \le (m-1)(\omega_P - 2) + m(2m-1) \le 3m(m-1) + 1$,

with ω_P with the Hamming weight of $P(x)$.

The C and L depend on the selection of the irreducible polynomial $P(x)$. One multiplication is computed in one clock cycle with a parallel PB multiplier.

4.1.3.3 Serial Dual-Basis Multiplier

Let $A, B, C \in GF(2^m)$ and $C = A * B$ denote the product in $GF(2^m)$. We assume that A is in dual basis representation and B is in PB representation, i.e., $A = \sum a_i y_i$ and $B = \sum x_i \alpha_i$ with $\{x_0, x_1, \cdots, x_{m-1}\}$ is PB and $\{y_0, y_1, \cdots, y_{m-1}\}$ is the corresponding DB calculated by Eq. (4.2).

From the duality relation and the properties of the trace function, c_0 can be obtained as follows (see Mastrovito [150], Section 3.2):

$$\begin{aligned} c_0 &= Tr(AB) = Tr(b_0 A) + Tr(b_1 \alpha A) + \cdots + Tr\left(b_{m-1}\alpha^{m-1} A\right) \\ &= a_0 b_0 + a_1 b_1 + \cdots + a_{m-1} b_{m-1} \\ &= A \cdot B \end{aligned} \tag{4.6}$$

where "$\cdot$" denotes the inner product.

To obtain the second coefficient c_1, A is replaced by xA in Eq. (4.6) and compute the inner product $(xA)\cdot B$, the third coefficient c_2 is obtained by computing $(x^2 A)\cdot B$ and so on for the remaining coefficients as $c_i = \left(x^i A\right)\cdot B$. The block diagram of DB multiplier is shown in Fig. 4.3.

Properties of the DB Multiplier: the complexity of DB multiplier is $3m$-bit register and $7m$ gates. The critical path has length $2 + \lceil log_2 m \rceil$ (one register + the inner-product logic). The execution of one modular multiplication is completed in m clock cycles.

4.1.3.4 Serial Normal-Basis (Massey-Omura) Multiplier

The NB multiplier was first proposed by Massey and Omura [187], hence it is known as the Massey-Omura (MO) multiplier. The key property of the NB representation is that squaring is a very simple operation, namely a single cyclic left shift of the operand. Let $A = a_0 x + a_1 x^2 + \cdots + a_{m-1} x^{2^{m-1}} \in GF(2^m)$. Then

$$\begin{aligned} A^2 &= a_0 x^2 + a_1 x^4 + \cdots + a_{m-1} x^{2^m} \\ &= a_{m-1} x + a_0 x^2 + a_1 x^4 + \cdots + a_{m-2} x^{2^{m-1}} . \end{aligned} \tag{4.7}$$

Here we use the linearity of the squaring operation and the fact that $x^{2^m} = x$.

Let $C = AB = c_0 x + c_1 x^2 + \cdots + c_{m-1} x^{2^{m-1}}$ be the product in $GF(2^m)$. Then the last coefficient c_{m-1} of C is a function of the coefficients of A and B, *i.e.*,

$$c_{m-1} = f(a_0, a_1, \cdots, a_{m-1}; b_0, b_1, \cdots, b_{m-1}) .$$

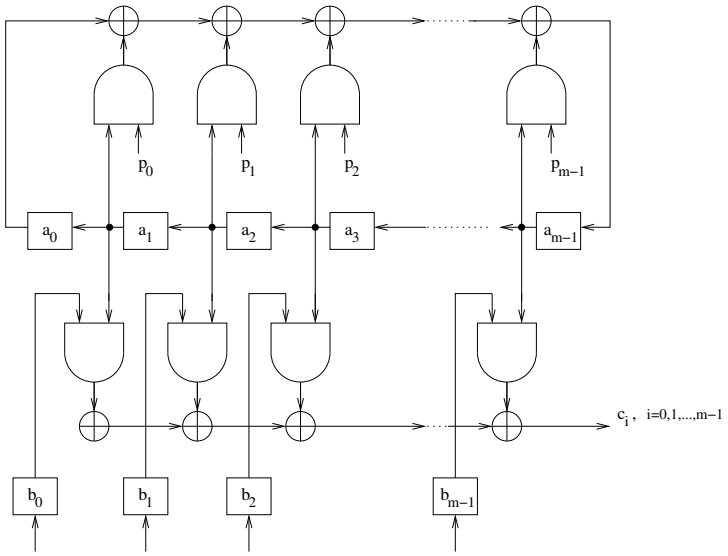


Figure 4.3: Block diagram of the dual basis (DB) multiplier over $GF\,(2^m)$

By Eq. (4.7) C^2 can be written as follows:

$$\begin{aligned} C^2 &= A^2B^2 \\ &= (a_{m-1}, a_0, \cdots, a_{m-2}) \cdot (b_{m-1}, b_0, \cdots, b_{m-2}) \\ &= (c_{m-1}, c_0, \cdots, c_{m-2}) \ , \end{aligned}$$

which means that the last coefficient c_{m-2} of C^2 can be obtained by applying the same function f to the components of A^2 and B^2. By squaring C repeatedly, all the coefficients of C can be found as

$$\begin{aligned} c_{m-1} &= f\,(a_0, a_1, \cdots, a_{m-1}; b_0, b_1, \cdots, b_{m-1}) \\ c_{m-2} &= f\,(a_{m-1}, a_0, \cdots, a_{m-2}; b_{m-1}, b_0, \cdots, b_{m-2}) \\ &\vdots \\ c_0 &= f\,(a_1, a_2, \cdots, a_0; b_1, b_2, \cdots, b_0) \ . \end{aligned}$$

The complexity of this multiplier depends on the function f which in turn, depends on the choice of $P(x)$. The schematic view of a MO multiplier over $GF\left(2^4\right)$ with $P(x) = x^4 + x^3 + 1$ is shown in Fig. 4.4.

Properties of Serial the NB (Massey-Omura) Multiplier: as a measure of the complexity of the Massey-Omura multiplier, the number of terms $a_i b_j$ in f is used. This number is denoted by N_m. The complexity of MO multiplier is: $(3.5N_m - 2)$ gates+ $2m$-bit register. The length of the critical path is $2 + \lceil log_2 N_m \rceil$ (one register and the logic function f). The execution of one modular multiplication is completed in m clock cycles.

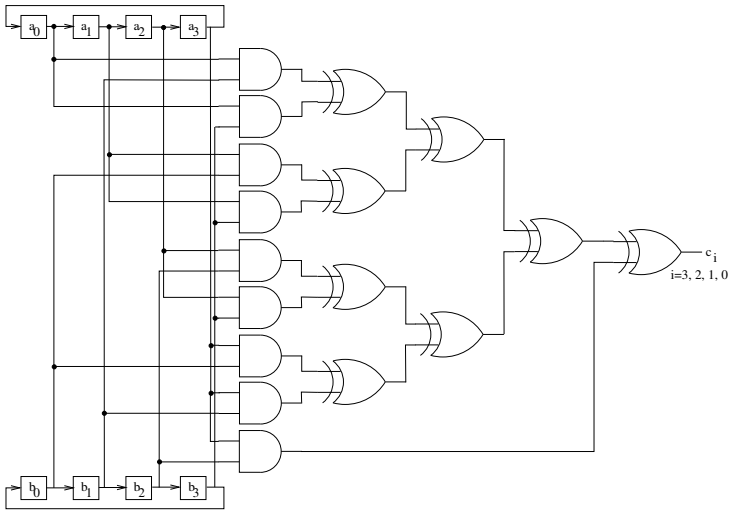


Figure 4.4: Schematic view of the Massey-Omura multiplier over $GF\left(2^4\right)$ with the irreducible polynomial $P(x) = x^4 + x^3 + 1$

Mullin *et al.* showed that $N_m \geq 2m - 1$ for any NB in [173] and the concept of *optimal normal basis* (ONB) is introduced. An NB is said to be optimum if $N_m = 2m - 1$. There are two constructions given in [173] to obtain an ONB in $GF\left(2^m\right)$; these constructions are called type I and type II.

4.1.3.5 State of the Art for $GF\left(2^m\right)$ Multiplier Hardware Implementations

Polynomial Basis

In 1971 Laws and Rushforth proposed a cellular-array multiplier [137]. This array exhibits a high degree of regularity and can be viewed as a device for performing computations in space rather than in time. The block diagram of the cellular-array multiplier is shown in Fig. 4.5. The schematic view of a general individual cell is shown in Fig. 4.6. The other work about this structure is from Jain *et al.* [114]. They also fabricated a multiplier chip for $GF\left(2^4\right)$ using CMOS 1.2 μm technology. The chip has an active area of 0.434 mm^2 and requires 1076 transistors; it is programmable for different irreducible polynomials.

In 1984 Yeh *et al.* [282] proposed a similar architecture as in [137]. They wanted to implement not only $AB \bmod P$ but also $AB + C \bmod P$. This is straightforward, because addition is simply bitwise XOR. Hence they add one more XOR gate to the cells of SSR and cellular-array multipliers to get their

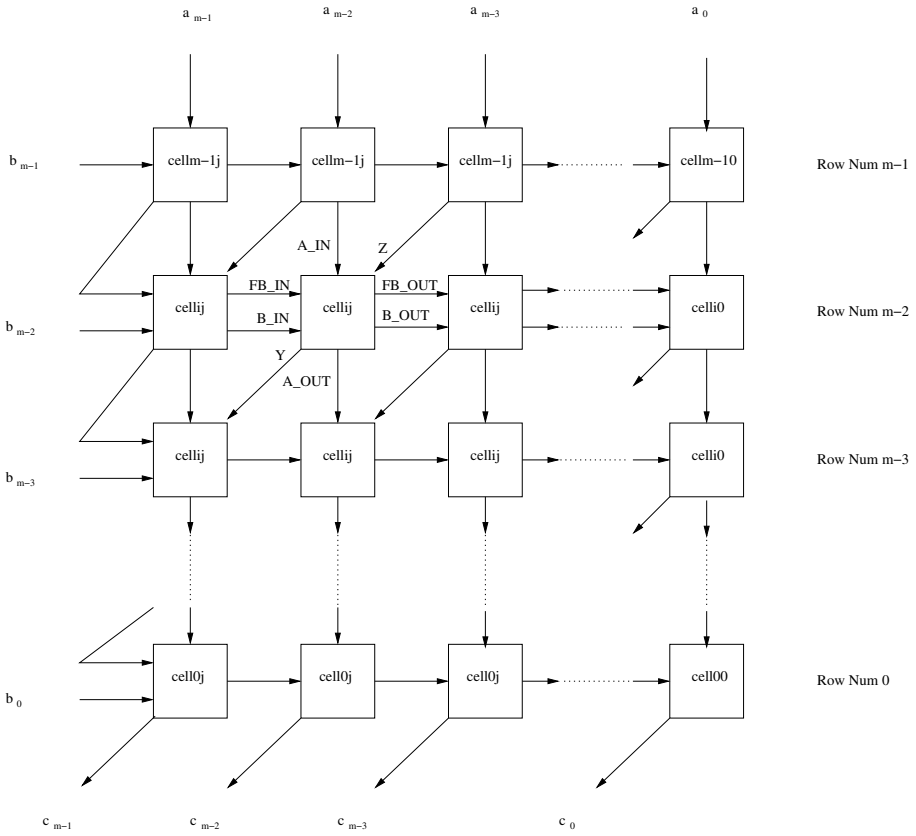


Figure 4.5: Block diagram of the cellular-array multiplier over $GF(2^m)$

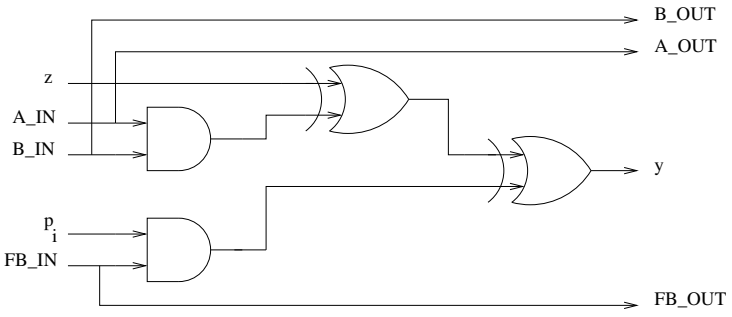


Figure 4.6: Schematic view of a cell of the cellular-array multiplier over $GF(2^m)$

architecture.

Hasan and Bhargava proposed a bit-serial systolic architecture for multiplication in [100]. Hasan also proposed to use a look-up table approach in [97]. Hasan *et al.* proposed two structures of parallel multipliers based on an irreducible all one polynomial (AOP) of degree m and equally spaced polynomial (ESP) of degree $m(m+1)^i$ [101]. A polynomial $f(z) = \sum_{i=0}^{m} f_i z^i$ over $GF(2)$ is called AOP of degree m if $f_i = 1$ for $i = 0, 1, \ldots, m$. A polynomial $g(z) = \sum_{i=0}^{sm} g_i z^i = z^{sm} + z^{s(m-1)+\cdots+z^s+1} = f(z^s)$ over $GF(2)$, where $f(z)$ is an AOP of degree m over $GF(2)$ is called an s-ESP of degree sm. The details of the proposed algorithms will not be given here. The total delay of the parallel AOP multiplier is $D_A + (m + \lceil \log_2(m-1) \rceil) D_X$ where D_X and D_A denote the delay for an XOR gate and an AND gate, respectively. The total number of XOR and AND gates are $m^2 + m - 2$ and m^2, respectively. Another result on multipliers for finite fields defined by AOP and ESP was reported by C.-Y. Lee *et al.* in [139]. They used a systolic architecture for bit-parallel multipliers.

The first result on composite field, $GF\left((2^n)^m\right)$, is from Paar [207, 208]. Also Paar and Rosner compare multiplication in composite fields and prime fields, $GF\left(2^m\right)$, with m prime, in [210].

S.-W. Wei and Tsai and Wang used systolic architectures for parallel multiplication in [269] and [257], respectively. Wu proposed explicit formulas to calculate the complexity of parallel multiplication in [272].

Orlando and Paar gave the results of their implementation of a serial multiplier on several FPGAs in [191]. They describe a prototype implementation on a Xilinx XC4000X FPGAs. They also present timing estimations of elliptic curve point multiplication using projective coordinates.

Song and Parhi, Sunar and Koç, Halbutoğulları and Koç reported their results on the Mastrovito multiplier in [243], [247] and [95], respectively. A design methodology for a Mastrovito multiplier was proposed by Zhang and Parhi in [284].

The multiplier architecture from Hasan and Wassal [102] which was designed by using triangular and polynomial basis together was fabricated. The design was optimized in CMOS 0.5 μm technology for a 3.3 V supply voltage. The prototype can support operations over finite fields up to $GF\left(2^{64}\right)$. The silicon area used was approximately $3.445\,\text{mm} \times 3.827\,\text{mm}$. The prototype chip was tested at a clock frequency of 50 MHz. This frequency limitation is reported as due to the packaging technology used. The authors claim that the implemented chip core could run at a frequency of more than 80 MHz, while an implementation with $m = 256$ was estimated to run at a frequency of more than 75 MHz.

Gao and Parhi implemented parallel multiplier by using two different ap-

proaches, irregular and regular semi-systolic [73]. They used 0.35 μm CMOS technology to implement both structures for $GF\left(2^8\right)$. Another parallel structure by Koç and Sunar can be found in [131].

Yu gives comparisons for power consumption of their architectures [283]: semi-systolic array, Mastrovito and composite field.

Hasan *et al.* proposed a squaring structure which requires $m-1$ XOR gates and has a time delay of one XOR gate in [101]. Jain *et al.* proposed semi-systolic architectures for parallel squaring in polynomial basis [114]. Information about the complexity of squaring can be found in the articles by Wu [272, 274]. Orlando and Paar proposed a squaring architecture which can also be used for multiplication [193].

Normal Basis

Wang *et al.* propose a parallel architecture for the Massey-Omura multiplier in [265]. They have fabricated a chip for $GF\left(2^4\right)$ using 4 μm negative-channel metal-oxide semiconductor (NMOS) technology. The chip has 8 pins and has an area of $1248\,\mu\text{m} \times 996\,\mu\text{m}$. Other parallel structures are proposed by Koç and Sunar, Reyhani-Masoleh and Hasan in [131] and [218], respectively.

Since the multiplication function, f, given in Section 4.1.1.3 depends on the irreducible polynomial, every time the polynomial is changed the multiplier circuit has to be changed. An algorithmic way to find f for any irreducible polynomial is proposed by Wang in [266, 267].

In 1988 Onyszchuk, Mullin and Vanstone propose a different approach from the Massey-Omura multiplier; this invention can be found in [188]. Agnew *et al.* improve the results in [2]. In 1992 the same authors report a Very Large Scale Integration (VLSI) implementation of a normal basis multiplier [4]. The chip has been fabricated using 1.5 μm high speed CMOS (HCMOS) gate array with a clock speed of 40 MHz; it requires less than 12 000 gates.

Lu gives maximum, minimum and average values of the measure of the complexity of the Massey-Omura multiplier, N_m, for different m in [144] (for a definition of N_m see Section 4.1.1.3). He proposes a way to find the optimal multiplication function f.

An architecture to realize composite field multiplication has been patented by Mullin [172]. Because he used composite fields, the multiplication is done with more than one bit of the operands in one clock cycle.

Gao and Sobelman gave some VLSI design results in [76].

Sunar and Koç present algorithms for a type II ONM multiplier in [248]. Sutikno and Surya propose architectures for ONB multiplier in extension and prime field [250]. Reyhani-Masoleh and Hasan propose architectures for a serial NB

multiplier in [216]. The same authors give comparisons of algorithms for NB, type I ONB and composite field multiplication in [217].

Dual Basis

Bit-serial systolic multiplier structures were proposed by Diab and Poli in [57]. Fenn *et al.* propose bit-serial and parallel multipliers and give delay and area comparisons in [67]. Wu, Hasan and Blake propose parallel multipliers in [276, 277]. Lee and Lim define a new dual basis called *circular dual basis* (CDB) in [138]. They give the algorithms for multiplication, squaring, conversion and inversion in [138]. Gollmann reports some results about ESP and dual basis in [82].

Comparison of Different Basis

Hsu *et al.* report their VLSI implementation results of 8-bit finite field multipliers using dual, normal and standard basis in [107]. Their conclusions are:

- The DB multiplier occupies the smallest amount of chip area.
- As the order of the field goes higher, the DB multiplier will increase its advantage over the other multipliers.
- The NB multiplier is very effective in performing operations such as inversion, squaring and exponentiation.
- The area of the NB multiplier grows faster than the area of the PB multiplier as the order of the field goes up.
- The PB does not require basis conversion.
- Due to the regularity and simplicity of PB, the design and expansion to higher order finite fields are easier.

Paar and Lange present VLSI implementation results of finite field multipliers using dual, normal and standard basis for different orders in [209].

Ahlquist *et al.* present FPGA implementation results of several finite field multipliers in [9]. The performance of each finite field multiplier is characterized on the Xilinx XC4062 FPGA. From the results they conclude that finite field multipliers optimized specifically for VLSI are not necessarily optimized for FPGAs. They claim that the following three significant flaws are responsible for this mismatch:

- multi-clock cycle operation;
- long unregistered data paths; and
- under utilized logic elements.

They modified their designs to avoid these flaws [9].

Composite extension fields, $GF((2^n)^m)$ are not recommended for security reasons [77, 71, 240]. Namely, the method of Weil descent for solving the ECDLP, as introduced by Frey [70], can be applied to these fields. This method does not apply for curves over $GF(2^p)$ where p is prime; most standards, except for Internet Protocol SECurity protocol(IPSEC [109], the use of $GF(2^p)$ with p prime.

4.1.4 Montgomery Modular Multiplication over $GF(2^m)$

The Montgomery multiplication method requires that $r(x)$ and $p(x)$ are relatively prime, *i.e.*,

$$gcd(r(x), p(x)) = 1\,.$$

For this assumption to hold, it suffices that $p(x)$ be not divisible by x. Since $p(x)$ is an irreducible polynomial over the field $GF(2)$, this will always be the case. Since $r(x)$ and $p(x)$ are relatively prime, there exist two polynomials $r^{-1}(x)$ and $p'(x)$ with the property that

$$r(x)r^{-1}(x) + p(x)p'(x) = 1\,, \tag{4.8}$$

where $r^{-1}(x)$ is the inverse of $r(x)$ modulo $p(x)$. The polynomials $r^{-1}(x)$ and $p'(x)$ can be computed using the extended Euclidean algorithm [153, 142]. The Montgomery modular multiplication (MMM) of $a(x)$ and $b(x)$ is defined as the product

$$c(x) = a(x)b(x)r^{-1}(x) \mod p(x)\,,$$

where $r(x) = x^m$. This equation can be computed using the following algorithm:

Algorithm 4.1: Montgomery modular multiplication over $GF(2^m)$

Require: $a(x)$, $b(x)$, $p(x)$, $P_0'(x)$
Ensure: $c(x) = a(x)b(x)x^{-m} \bmod p(x)$
$t(x) := a(x)b(x)$
$u(x) := t(x)n'(x) \bmod r(x)$
$c(x) := [t(x) + u(x)p(x)]\,/r(x)$

In order to prove the correctness of Algorithm 4.1, we note that $u(x) = t(x)p'(x) \bmod r(x)$ implies that there is a polynomial $K(x)$ over $GF(2)$ with the property

$$u(x) = t(x)p'(x) + K(x)r(x)\,. \tag{4.9}$$

We write the expression for $c(x)$ in Step 3 of Algorithm 4.1 and then substitute $u(x)$ with the expression (4.9) as

$$\begin{aligned} c(x) &= \tfrac{1}{r(x)}\left[t(x) + u(x)p(x)\right] \\ &= \tfrac{1}{r(x)}\left[t(x) + t(x)p'(x)p(x) + K(x)r(x)p(x)\right] \end{aligned}$$

Furthermore, we have $p'(x)p(x) = 1 + r(x)r^{-1}(x)$ according to Eq. (4.8). Thus, $c(x)$ is obtained as

$$\begin{aligned} c(x) &= \tfrac{1}{r(x)}\left[t(x) + t(x)\left[1 + r(x)r^{-1}(x)\right] + K(x)r(x)p(x)\right] \\ &= \tfrac{1}{r(x)}\left[t(x)r(x)r^{-1}(x) + K(x)r(x)p(x)\right] \\ &= t(x)r^{-1}(x) + K(x)p(x) \\ &= t(x)r^{-1}(x) \bmod p(x) \\ &= a(x)b(x)r^{-1}(x) \bmod p(x)\,, \end{aligned}$$

as required. Now, we will show that the degree of the polynomial $c(x)$ computed by Algorithm 4.1 is less than and equal $m-1$. Since the degrees of $a(x)$ and $b(x)$ are both less than and equal $m-1$, the degree of $t(x) = a(x)b(x)$ will be less than and equal to $2(m-1)$. Also note that the degrees of $p(x)$ and $r(x)$ are both equal to m. The degree of $u(x)$ computed in Step 2 of Algorithm 4.1 will be strictly less than m since the operation is performed modulo $r(x)$. Thus, the degree of $c(x)$ as computed in Step 3 of Algorithm 4.1 is found as

$$\begin{aligned} deg\left(c(x)\right) &\leq max\left(deg\left(t(x)\right), deg\left(u(x)\right) + deg\left(p(x)\right)\right) - deg\left(r(x)\right) \\ &\leq max\left(2m-2, m-1+m\right) - m \\ &\leq m-1\,. \end{aligned}$$

Thus, the polynomial $c(x)$ is reduced modulo $p(x)$.

4.1.4.1 Computation of Montgomery Multiplication

The product $c(x) = a(x)b(x)r^{-1}(x) \bmod p(x)$ can be written as

$$c(x) = x^{-m}a(x)b(x) = x^{-m}\sum_{i=0}^{m-1} a_i x^i b(x) \bmod p(x)\,.$$

The product

$$t(x) = \left(a_{m-1}x^{m-1} + a_{m-2}x^{m-2} + \cdots + a_1 x + a_0\right) b(x)$$

can be computed by starting from the MSB and then proceeding to the LSB, as follows:

$t(x) := 0$
for $i = m-1$ to 0
  $t(x) := t(x) + a_i b(x)$
  $t(x) := xt(x)\,.$

The shift factor x^{-m} in $x^{-m}a(x)b(x)$ reverses the direction of summation. Since

$$\begin{aligned} & x^{-m}\left(a_{m-1}x^{m-1} + a_{m-2}x^{m-2} + \cdots + a_1 x + a_0\right) \\ & = a_{m-1}x^{-1} + a_{m-2}x^{-2} + \cdots + a_1 x^{-m+1} + a_0 x^{-m}\,, \end{aligned}$$

we start processing the coefficients of $a(x)$ from the LSB and obtain the following bit-level algorithm in order to compute $t(x) = a(x)b(x)x^{-m}$ as follows:

$$\begin{array}{l} t(x) := 0 \\ \text{for } i = m-1 \text{to } 0 \\ \quad t(x) := t(x) + a_i b(x) \\ \quad t(x) := t(x)/x\,. \end{array}$$

We are interested in computing $c(x) = x^{-m}a(x)b(x) \bmod p(x)$. Because

$$c(x)x^{-1} \bmod p = \left(c(x) + u(x)p(x)\right) x^{-1} \bmod p\,,$$

$u(x)$ can be chosen in a way that $c(x) + u(x)p(x)$ will be divisible by x. This requires that the LSB of the sum $c(x) + u(x)p(x)$ is '0'. Since, $p(x)$ is a irreducible polynomial, the LSB of $p(x)$, p_0, is '1'. If $u(x)$ is chosen as the LSB of $c(x)$, c_0, then the requirement will be fulfilled. The bit-level algorithm for MMM is given in Algorithm 4.2.

Algorithm 4.2: Bit-level algorithm for the Montgomery modular multiplication over $GF\,(2^m)$

Require: $a(x)$, $b(x)$, $p(x)$
Ensure: $c(x) = a(x)b(x)x^{-m} \bmod p(x)$
1: $c(x) := 0$
2: **for** i from 0 to $m-1$ **do**
3: $\quad c(x) := \left(c(x) + a_i b(x) + c_0 p(x)\right)/x$
4: **end for**

Algorithm 4.2 for MMM is generalized to the word-level algorithm by proceeding word by word, where the word size is $\omega \geq 2$ and $s = \lceil m/\omega \rceil$. Let $A_i(x)$ represent one word of the polynomial $a(x)$. Step 3 of Algorithm 4.2 is then performed by multiplying $A_i(x)$ by $b(x)$ at the ith iteration. We then need to multiply the partial product $c(x)$ by $x^{-\omega}$ modulo $p(x)$. In order to perform this step using division, we add a multiple of $p(x)$ to $c(x)$ so that the least significant ω coefficients of $c(x)$ will be zero, *i.e.*, $c(x)$ will be divisible by x^ω. Thus, if $c(x) \neq 0 \bmod x^\omega$, then we find $M(x)$ (which is a polynomial of length ω) such that $c(x) + M(x)p(x) = 0 \bmod x^\omega$. Let $C_0(x)$ and $P_0(x)$ be the least significant words of $c(x)$ and $p(x)$, respectively. We calculate $M(x)$ as

$$M(x) = C_0(x)N_0^{-1}(x) \bmod x^\omega\,.$$

We note that $N_0^{-1} \bmod x^\omega$ is equal to $N_0'(x)$ since Eq. (4.8) implies that

$$\begin{aligned} x^{s\omega}x^{-s\omega} + p(x)p'(x) &= 1 \bmod x^\omega \\ P_0(x)P_0'(x) &= 1 \bmod x^\omega \end{aligned} \quad . \tag{4.10}$$

The word-level algorithm for the MMM obtained by Koç and Acar in [168, 130] is given in Algorithm 4.3. The papers by Wu [273, 275] are good references to learn about this technique.

Algorithm 4.3: Word-level algorithm for Montgomery modular multiplication over $GF(2^m)$

Require: $a(x)$, $b(x)$, $p(x)$, $P_0'(x)$
Ensure: $c(x) = a(x)b(x)x^{-m} \bmod p(x)$
1: $c(x) = 0$
2: **for** i from 0 to $s-1$ **do**
3: $\quad M(x) = (C_0(x) + A_i(x)B_0(x))\, P_0'(x) \bmod x^\omega$
4: $\quad c(x) = (c(x) + A_i(x)b(x) + M(x)p(x))\, /x^\omega$
5: **end for**

4.1.4.2 Montgomery Modular Multiplication Circuit

The design of the Montgomery modular multiplication circuit (MMMC) that performs Algorithm 4.3 is a contribution of this thesis. A systolic array with variable bit-length m and variable word length ω is used as an architecture. This systolic array architecture makes the clock frequency independent of the bit-length m. The clock frequency only depends on the word length ω. The results of our design for the MMMC are published in [160, 157, 158, 17].

As explained above $P_0'(x) = P_0^{-1}(x) \bmod x^\omega$ needs to be computed. This is done by using the observation that $P_0(x)$ and its inverse satisfy $P_0(x)P_0^{-1}(x) = 1 \bmod x^i$ for $i = 1, 2, \cdots \omega$ [130]. We have designed a circuit that calculates the coefficients of the polynomial P_0^{-1} during the first clock cycle and writes the result in a register. Then the value of this register is used as an input for the MMMC in Algorithm 4.3.

The architecture of the MMMC consists of a systolic array, a circuit to compute $P_0'(x)$, a read-in and read-out mechanism and a state machine to control the MMM. The read-in mechanism ensures that the inputs to the systolic array arrive at the correct moment. The read-out mechanism registers the result of the MMM.

4.1.4.3 Systolic array

The *ith* iteration of Step 4 in Algorithm 4.3 computes the temporary results

$$c_i(x) = x^{-\omega} \left(c_{i-1}(x) + A_i(x)b(x) + M_i(x)p(x) \right) , \tag{4.11}$$

where $i = 0, \cdots, m-1$ and $c_{-1}(x) = 0$. $c_i(x)$ can be divided into s words of length ω and Eq. (4.11) can be rewritten as follows:

$$\begin{aligned} &\sum_{j=0}^{s} C_{i,j}(x)x^{j\omega} \\ &\quad = x^{-\omega} \left(\sum_{j=0}^{s} C_{i-1,j}(x)x^{j\omega} + A_i(x) \sum_{j=0}^{s} B_j(x)x^{j\omega} + M_i(x) \sum_{j=0}^{s} P_j(x)x^{j\omega} \right) , \end{aligned} \tag{4.12}$$

where $i = 0, \cdots, s$, $C_{i,j}(x)$, $B_j(x)$ and $P_j(x)$ are the jth words of $c_i(x)$, $b(x)$ and $p(x)$, respectively. In order to calculate $C_{i,j}(x)$ we compute the product terms in Eq. 4.12 as follows:

$$\begin{aligned} A_i(x)B_j(x) &= \left(\sum_{k=0}^{\omega-1} a_{k+\omega i}x^k \right) \left(\sum_{k=0}^{\omega-1} b_{k+\omega j}x^k \right) \\ &= x^{\omega} \left(\sum_{l=1}^{\omega-1} a_{\omega-l+\omega i} \left(\sum_{k=l}^{\omega-1} b_{k+\omega j}x^{k-l} \right) \right) + \left(\sum_{l=0}^{\omega-1} a_{l+\omega i} \left(\sum_{k=0}^{\omega-1-l} b_{k+\omega j}x^{k+l} \right) \right) \\ &= x^{\omega} HAB_{i,j}(x) + LAB_{i,j}(x) , \end{aligned} \tag{4.13}$$

for $i = 0, \cdots, s-1$ and $j = 0, \cdots, s-1$.

$$\begin{aligned} M_i(x)P_j(x) &= \left(\sum_{k=0}^{\omega-1} m_{i,k}x^k \right) \left(\sum_{k=0}^{\omega-1} p_{k+\omega j}x^k \right) \\ &= x^{\omega} \left(\sum_{l=1}^{\omega-1} m_{i,\omega-l} \left(\sum_{k=l}^{\omega-1} p_{k+\omega j}x^{k-l} \right) \right) + \left(\sum_{l=0}^{\omega-1} m_{i,l} \left(\sum_{k=0}^{\omega-1-l} p_{k+\omega j}x^{k+l} \right) \right) \\ &= x^{\omega} HMP_{i,j}(x) + LMP_{i,j}(x) , \end{aligned} \tag{4.14}$$

for $i = 0, \cdots, s-1$ and $j = 0, \cdots, s$. We substitute Eq. (4.13) and Eq. (4.14) in Eq. (4.12) as follows:

$$\begin{aligned} \sum_{j=0}^{s} C_{i,j}(x)x^{j\omega} = x^{-\omega} \Bigg(&\sum_{j=0}^{s} C_{i-1,j}(x)x^{j\omega} + \sum_{j=0}^{s} \left(x^{\omega} HAB_{i,j}(x) + LAB_{i,j}(x) \right) x^{j\omega} \\ &+ \sum_{j=0}^{s} \left(x^{\omega} HMP_{i,j}(x) + LMP_{i,j}(x) \right) x^{j\omega} \Bigg) , \end{aligned}$$

for $i = 0, \cdots, s$. According to the above expression we can calculate jth word of $c_i(x)$, $C_{i,j}(x)$, as follows:

$$C_{i,j}(x) \tag{4.15}$$
$$= x^{-\omega}(C_{i-1,j}(x) + HAB_{i,j-1}(x) + LAB_{i,j}(x) + HMP_{i,j-1}(x) + LMP_{i,j}(x)) \ .$$

Every word $C_{i,j}(x)$ of $c_i(x)$ is computed in a separate cell; these cells are contained in the systolic array. There are three different kinds of cells. Most of the words are calculated by a regular cell (cell 1, ..., cell $s-1$). Two special cells, the rightmost cell (cell 0) and the leftmost cell (cell s), perform the rest of the calculations. Fig. 4.7 shows the different cells in the systolic array. In

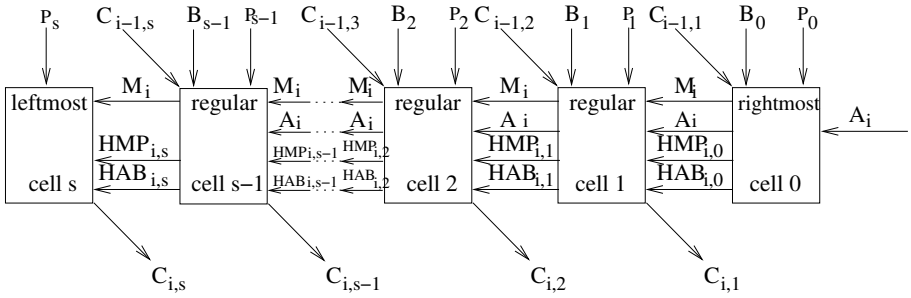


Figure 4.7: Schematic view of the one dimensional systolic array used in the Montgomery modular multiplier over $GF(2^m)$

the ith iteration step the array computes $c_i(x)$ by using $c_{i-1}(x)$, $A_i(x)$, $b(x)$ and $p(x)$. Fig. 4.7 shows a schematic view of the array. The output $C_{i,j+1}$ of the $(j+1)th$ cell is used as the input $C_{i-1,j+1}$ for the jth cell during the next iteration. This way the division by x^{ω} in Step 4 of Algorithm 4.3 is implemented.

Regular cell

Figure 4.8 shows the regular cell (cell 0, ..., cell $s-1$) which calculates Eq. (4.15). It consists of two different operations:

- **bitwise XOR ($\bigoplus$):** this is implemented as a bit-wise XOR array which consists of ω XOR gates with 5 inputs.
- **a multiplication ($*$):** this performs a multiplication of two ω-bit numbers and produces a 2ω-bit output which is divided into the least significant and the most significant parts according to Eq. (4.13) and Eq. (4.14).

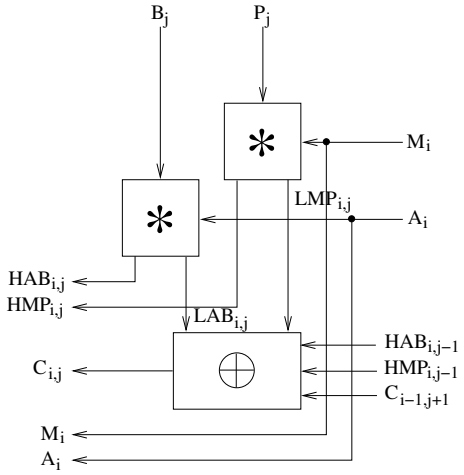


Figure 4.8: Schematic view of a regular cell in the systolic array used in the Montgomery modular multiplier over $GF(2^m)$ ($*$ = multiplication, $\bigoplus$ = bit-wise XOR)

Rightmost cell

Based on Step 3 in Algorithm 4.3 we can compute $M_i(x)$ as follows:

$$M_i(x) = (C_{i-1,1}(x) + LAB_{i,0}(x))\, P_0'(x) \bmod x^{\omega}\,,$$

with $i = 0, \ldots, s-1$ and $C_{-1,0}(x) = 0$. According to Step 4 in Algorithm 4.3 $M_i(x)$ is not an input to the rightmost cell, but obtained in the rightmost cell. Because $B_{-1}(x) = 0$ and $P_{-1}(x) = 0$ Eq. (4.15) can be simplified as follows:

$$\begin{aligned} C_{i,0}(x) &= C_{i-1,1}(x) + LAB_{i,0}(x) + LMP_{i,j}(x) \\ &= C_{i-1,1}(x) + LAB_{i,0}(x) + (C_{i-1,1}(x) + LAB_{i,0}(x))\, P_0'(x) P_0(x) \bmod x^{\omega}\,, \end{aligned}$$

for $i = 0, \ldots, s-1$. We substitute Eq. (4.10) in the above expression and the additions are bitwise XOR, then $C_{i,0} = 0$. This is the required result to be able to divide $c_i(x)$ by x^{ω} in Section 4.1.4.1. Hence, the rightmost cell does not have the C output. Figure 4.9 shows the rightmost cell (cell 0), which consists of three different operations:

- **bitwise XOR ($\bigoplus$):** this is implemented as a bit-wise XOR array which consists of ω XOR gates with 2 inputs.
- **a multiplication ($*$):** this performs a multiplication of two ω-bit numbers and produces a 2ω-bit output which is divided into the least significant and the most significant parts according to Eq. (4.13) and Eq. (4.14).

- **a modular x^ω multiplication ($\circledast$):** This is a multiplication modulo x^ω.

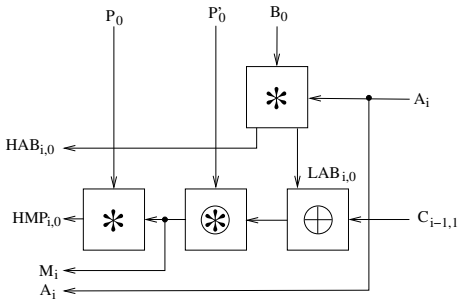


Figure 4.9: Schematic view of the rightmost cell in the systolic array used in Montgomery modular multiplier over $GF(2^m)$ ($*$ = multiplication, $\bigoplus$ = bitwise XOR, $\circledast$= multiplication modulo x^ω)

Leftmost cell

Because $B_s(x) = 0$, Eq. (4.15) for $C_{i,s}(x)$ can be simplified as follows:

$$C_{i,s}(x) = C_{i-1,s+1}(x) + HAB_{i,s-1} + HMP_{i,s-1}(x) + LMP_{i,s}\,, \qquad (4.16)$$

with $i = 0, \cdots, s-1$. As $P_s = 0\,\text{or}\,1$, $HMP_{i,s}(x) = 0$, also because $B_s(x) = 0$ and $P_{s+1}(x) = 0$ Eq. (4.15) for $C_{i,s+1}(x)$ can be simplified as follows:

$$C_{i,s+1}(x) = C_{i-1,s+2}(x)\,,$$

with $i = 0, \cdots, s-1$. Above equation means $C_{i,s+1}(x) = 0$. Eq. (4.16) is implemented by the leftmost cell (cell s), which is shown in Fig. 4.10. It consists of:

- **bitwise XOR ($\bigoplus$):** this is implemented as a bit-wise XOR array which consists of ω XOR gates with 2 inputs.
- **a multiplication ($*$):** this is implemented as a bit-wise AND array which consists of ω AND gates with 2 inputs.

4.1.4.4 Inversion

Eq. (4.17) finds the inversion of the polynomial $P_0(x)$ modulo x^ω. When $P_0(x) = p_{\omega-1}x^{\omega-1} + \cdots + p_1 x + p_0$ and $P_0^{-1}(x) = p'_{\omega-1}x^{\omega-1} + \cdots + p'_1 x + p'_0$,

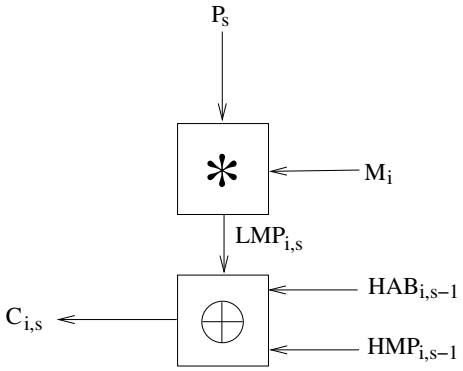


Figure 4.10: Schematic view of the leftmost cell in the systolic array used in Montgomery modular multiplier over $GF\,(2^m)$ ($*$ = bit-wise AND, $\bigoplus$ = bitwise XOR)

the coefficients $p'_i,\ i = 0, \cdots, w-1$ can be found as follows:

$$\begin{aligned}
&p_0 p'_0 = 1 \Rightarrow p'_0 = p_0 = 1\\
&p_1 p'_0 \oplus p'_1 p_0 = 0 \Rightarrow p'_1 = p_1\\
&p_2 p'_0 \oplus p_1 p'_1 \oplus p'_2 p_0 = 0 \Rightarrow p'_2 = p_2 \oplus p_1\\
&p_3 p'_0 \oplus p_2 p'_1 \oplus p_1 p'_2 \oplus p_0 p'_3 = 0 \Rightarrow p'_3 = p_3 \oplus p_1\\
&p_4 p'_0 \oplus p_3 p'_1 \oplus p_2 p'_2 \oplus p_1 p'_3 \oplus p_0 p'_4 = 0 \Rightarrow p'_4 = p_4 \oplus (p_1 + p_2)\\
&\cdots
\end{aligned} \tag{4.17}$$

We have designed a circuit that calculates the coefficients of the polynomial P_0^{-1} during the first clock cycle and writes the result in a register. Then the value of this register is used as an input for the MMMC.

Equation (4.17) depends on the word length ω. For every ω the circuit used to calculate it has to be designed specially. Hence if ω is changed frequently then $P_0^{-1}(x)$ can be computed in software and sent to the MMMC as an input. But this will bring extra communication delay with the user of the MMMC which will be repeated for every usage of the MMMC. The other possibility is to calculate $P_0^{-1}(x)$ once and store this in a LUT in the circuit, but this LUT table has to be updated for every change of ω; moreover, the LUT will use a larger area than the circuit used to calculate it. So depending on how frequent the MMMC is used, the area availability and the needed time to design the MMMC, one of the three solutions can be chosen.

4.1.4.5 Read-in and Read-out Mechanism

If the execution of every iteration of Step 3 and 4 in Algorithm 4.3 would happen in one clock cycle, the delay from $C_{i-1,1}$ to $C_{i,s}$ would be too large.

The leftmost cell would have to wait for $HAB_{i,s-1}$ and $HMP_{i,s-1}$ coming from cell $s-1$, while this cell has to receive $HAB_{i,s-1}$ and $HMP_{i,s-1}$ first, etc. This would imply that the minimum clock period would increase with m. Because the maximum clock frequency should not become too low and remain independent of the value of m, the design is implemented as a systolic array. Now, the maximum clock frequency only depends on the word length ω, because this value determines the number of logic gates in one cell. The critical path of the systolic array is the same as the critical path of one regular cell and it is independent of the bit-length m of the operands. It is equal to $T_{MULT/ADD} + 2T_{ADD} = T_{AND} + (w+3)T_{XOR}$, where T_{MULT} and T_{ADD} are the latencies of the multiplication/addition and the addition respectively. $C_{i,j}$ is calculated at the $2(i+1)+(j-1)$th clock cycle as the output of the $(j+1)$th cell.

Because of the registers in between, every cell evaluates another word of $a(x)$ in the same clock cycle. A left-shift register (LSR), `a_temp`, is used to provide every cell with the correct word of $a(x)$. Figure 4.11 shows an example for $m = 16$ and $\omega = 4$. For this example, the starting value, a_start, of `a_temp` is 0000 0000 0000 A_0 0000 A_1 0000 A_2 0000 A_3. In the same manner $M_i(x)$ is placed in the least significant word of a LSR, `m_temp`, to provide every cell with the correct version of $M_i(x)$.

Because the output words of $c(x)$ are not valid at the same time, a right-shift register (RSR), `counter`, is used to determine when the words are loaded in the output register as shown in Fig. 4.11. The length of `counter` is always $3s$. For this example, the starting value of `counter` is 100000000000. Every clock cycle the '1' in this register shifts one place to the right. At the end, this '1' is sent to the enable (E) of the output register.

4.1.4.6 State machine

Figure 4.12 shows the algorithmic state machine (ASM) chart of the MMMC. When the reset signal (RST) arrives, all the registers are reset. The circuit waits in the IDLE state for the START signal. When the START signal comes a_start is loaded into `a_temp`, the most significant bit of `counter` is set and the circuit goes to state S1. In every clock cycle `a_temp`, `m_temp` and `counter` are shifted ω, ω bits and 1 bit, respectively. Also, the outputs of the systolic array cells are returned to the inputs. When the least significant bit of `counter` is 1, after $3s$ clock cycles, a VALID signal is produced to indicate that the value of the output register, `result`, is ready. Hence the latency of the word level MMMC is $3s$ clock cycles.

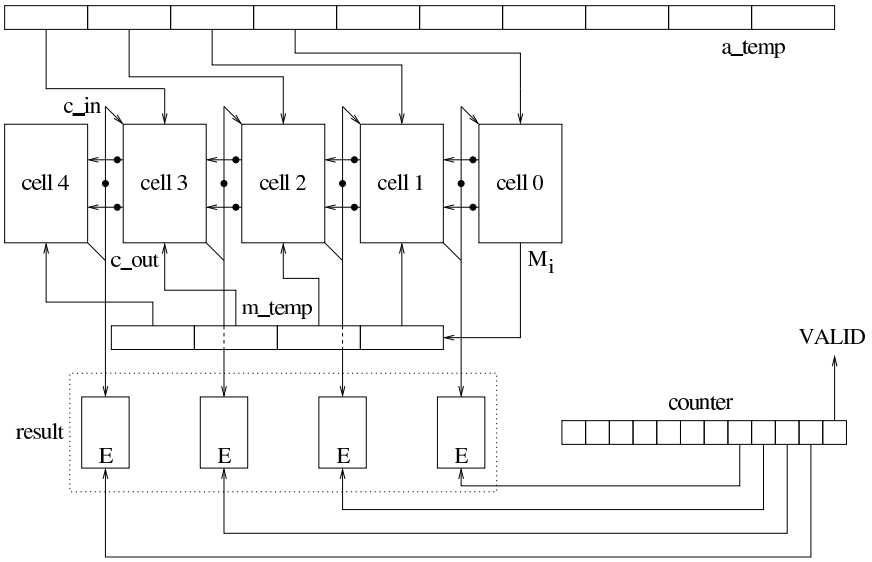


Figure 4.11: Architecture of the Montgomery modular multiplier circuit over $GF\,(2^m)$ for $m = 16$ and $\omega = 4$, as $m = s\omega$ ($\bullet$ = register, E=enable)

4.1.4.7 Implementation Results

The implementation results of the MMMC on a Xilinx Virtex XCV800-4 FPGA are indicated in Table 4.1. When the word length ω increases, the number of clock cycles needed for completing one MMM decreases, as it is $3s$ and $s = m/\omega$. The minimal clock period, T_p, increases by the increase of ω, because the critical path in the regular cell is increasing. The total MMM latency can be calculated as $T_{MMM} = 3\frac{m}{\omega}T_p$. As shown in Table 4.1, increasing the word length does not always give faster results. Figure 4.13.(a) shows that the total MMM latency reaches an optimum for $\omega = 16$, as the latency of MMM decreases until $\omega = 16$ and starts increasing with $\omega = 32$. Figure 4.13.(b) shows that the circuit becomes larger when ω increases.

An efficient implementation of the MMM over $GF\,(2^m)$ in hardware was considered by Wu [275]; the proposed parallel architecture is restricted to finite fields that are represented using irreducible trinomials. Our contribution consists of an FPGA implementation of MMM in a systolic array, which makes the clock frequency of the design independent of the field size m and allows for pipelining. The clock frequency depends of ω. The clock frequency of Wu's design decreases with the increase of m. Unlike Wu's design, our implementation is also suitable for finite fields that cannot be represented using a trinomial.

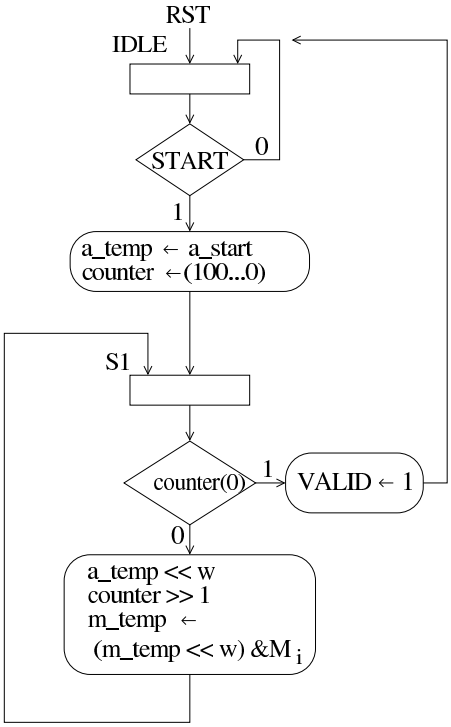


Figure 4.12: Algorithmic state machine chart of the Montgomery modular multiplier over $GF(2^m)$ ($<< x$ = left shift over x bits, $>> x$ = right shift over x bits, & = concatenation of two words)

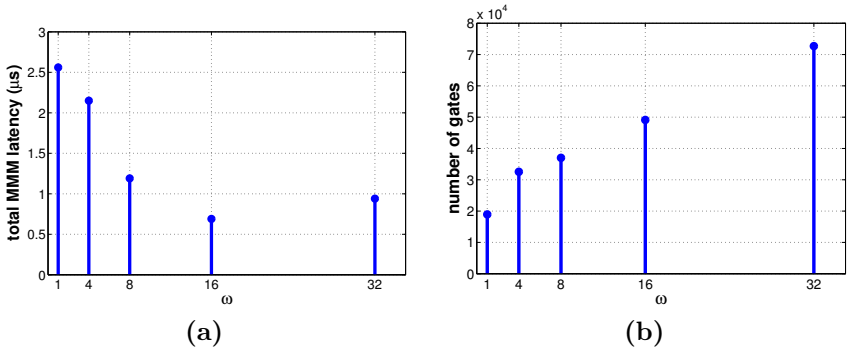


Figure 4.13: Implementation results of Montgomery modular multiplier over $GF(2^m)$ as a function of the word length ω: (a) latency (b) number of gates

Table 4.1: FPGA implementation results of the Montgomery modular multiplier over $GF(2^m)$

	$\omega = 1$	$\omega = 4$	$\omega = 8$	$\omega = 16$	$\omega = 32$
# of clock cycles	160	120	60	30	15
Minimum clock period	16.03 ns	17.89 ns	19.76 ns	23.162 ns	62.56 ns
Total MMM latency	2.56 μs	2.15 μs	1.19 μs	0.69 μs	0.94 μs
# of Gates	18 940	32 564	37017	49 112	72 690

4.1.5 Modular Multiplicative Inversion

Modular inversion is a vital operation for PKC protocols as shown in Chapter 2. It is used for example, to calculate a private RSA key for decryption, for ECDSA [108] etc. It is known to be the slowest operation that motivated many implementations to use projective coordinates. Yet, some inversions have to be performed even in this case and the fastest and the most secure way to do so is a dedicated inverter in hardware.

For any $\alpha \in GF(2^m)$, $\alpha^{-1} = \alpha^{2^m-2}$. Let $2^m - 2$ be decomposed as $2 + 2^2 + 2^3 + \cdots + 2^{m-1}$; then α^{-1} can be expressed as

$$\alpha^{-1} = \left(\alpha^2\right)\left(\alpha^{2^2}\right)\cdots\left(\alpha^{2^{m-1}}\right). \tag{4.18}$$

Thus, the computation of the inverse of α requires $m-1$ squaring operations and $m-2$ multiplications [265, 101].

Wang *et al.* propose a chip to realize Eq. (4.18) by using repeated square and multiply operations in normal basis [265]. Hasan *et al.* propose to use polynomial base representation for the same problem [101]. Feng, Fenn *et al.*, Calvo and Torres and Takagi *et al.* come up with different representations for $2 + 2^2 + 2^3 + \cdots + 2^{m-1}$ in [65], [66], [35] and [252], respectively. Wei uses the systolic architecture for modular multiplication and uses this architecture for repeated multiplications to realize Eq. (4.18) in [269]. Wang and Guo also propose a systolic architecture and claim that their architecture has half the area of Wei's in [89, 268].

Another decomposition for $2^m - 2$ is as follows

$$2^m - 2 = \left(2^{m/2} + 1\right)\left(2^{m/2} - 2\right) + 2^{m/2}.$$

Therefore α^{-1} can be computed by [111]

$$\begin{aligned}
\alpha^{-1} &= \left(\alpha^{2^{m/2}} \cdot \alpha\right)^{\left(2^{m/2}-2\right)} \cdot \alpha^{2^{m/2}} \\
&= y_1^{\left(2^{m/2}-2\right)} \cdot \alpha^{2^{m/2}} \\
&= y_1^{-1} \cdot \alpha^{2^{m/2}} \\
&= \left(y_1^{2^{m/4}} \cdot y_1\right)^{\left(2^{m/4}-2\right)} \cdot y_1^{2^{m/4}} \cdot \alpha^{2^{m/2}} \\
&= y_2^{-1} \cdot y_1^{2^{m/4}} \cdot \alpha^{2^{m/2}} \\
&\vdots
\end{aligned}$$

Applying a similar procedure iteratively a recursive algorithm for computing multiplicative inverses in $GF\left(2^m\right)$ is given in [111] by Itoh and Tsujii. They used a normal basis representation in order to compute the square easily. Asano *et al.* generalize in [14] the recursive algorithm proposed in [111] for multiplicative inverse computation in the composite field $GF\left(\left(2^m\right)^n\right)$.

Brunner *et al.* design a multiplicative structure which uses Euclid's algorithm to find the greatest common divisor (GCD) of two polynomials in [33]. Yan and Sarwate propose a systolic architecture for implementing Euclid's algorithm in [281].

Paar and Rosner give the result of their FPGA implementation of inversion in composite fields in [210]. Information about the complexity of inverse computation can be found in the work by Wu [272]. Gao and Sobelman gave some VLSI design results by using normal basis in [76]. Hasan and Wassal [102] implement inversion circuit which they proposed in the same work as a part of their $GF\left(2^m\right)$ arithmetic processor. They use normal basis for representation and extended Euclidean algorithm for GCD computation.

Modular inversion is often performed by the Extended Euclidean Algorithm (EEA) [127]. Kaliski [121] proposed a method of Montgomery Inverse which is also derived from the EEA. Some other modular inverse studies based on this technique are presented by Savaş and Koç in [225]. Their algorithm has been implemented in hardware in Gutub *et al.* [91]. In this architecture two VLSI hardware implementations are presented. Both are based on the same inversion algorithm with the difference of one being fixed but fully parallel and the other one being scalable. Both designs have been compared based on their speed and area. The area of the scalable design is on average 42% smaller than the fixed one.

4.2 Arithmetic Operations over $GF(p)$

In this section we introduce the arithmetic operations over $GF(p)$ used in ECC over $GF(p)$. The performance of an ECC depends on the performance of these operations specially the modular multiplication.

4.2.1 Addition/Subtraction

Modular addition and subtraction are executed according to Algorithm 4.4 and Algorithm 4.5, respectively [129].

Algorithm 4.4: Modular addition over $GF(p)$

Require: M, $0 \leq A < M$, $0 \leq B < M$
Ensure: $C = A + B \bmod M$
1: $C' = A + B$
2: $C'' = C' - M$
3: **if** $C'' < 0$ **then**
4: $\quad C = C'$
5: **else**
6: $\quad C = C''$
7: **end if**

Algorithm 4.5: Modular subtraction over $GF(p)$

Require: M, $0 \leq A < M$, $0 \leq B < M$
Ensure: $C = A - B \bmod M$
1: $C' = A - B$
2: $C'' = C' + M$
3: **if** $C' < 0$ **then**
4: $\quad C = C''$
5: **else**
6: $\quad C = C'$
7: **end if**

The numbers are represented in *two's complement* representation. In this representation, addition and subtraction can be realized with the same circuit [212]. Addition/subtraction circuit (ASC) includes a bit-serial adder with one full adder (FA), two shift registers, one flip-flop, a counter and a controller as shown in Fig. 4.14 [148]. One addition/subtraction is executed in l clock cycles, with l the bit-length of the A and B. Hence one modular addition/subtraction is executed in $2l + 1$ clock cycles.

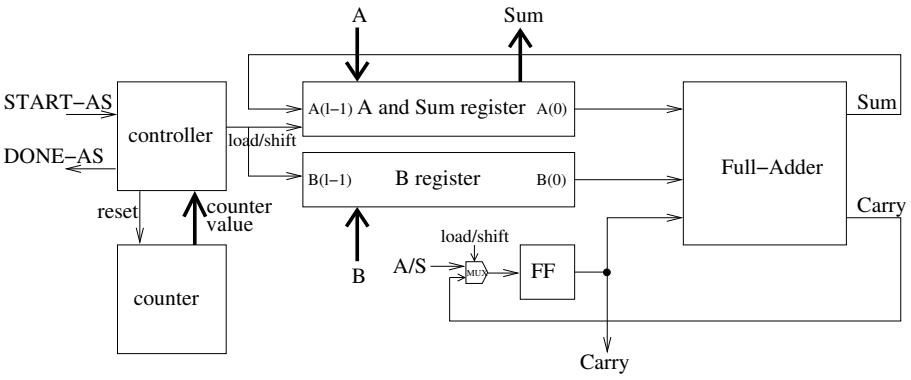


Figure 4.14: Architecture of the bit serial addition/subtraction circuit over $GF(p)$

4.2.2 Montgomery Modular Multiplier over $GF(p)$

Montgomery modular multiplication is defined as follows [168]:

$$\text{Mont}(x, y) = xyR^{-1} \mod N\,.$$

For a word base $b = 2^\alpha$, R should be chosen such that $R = 2^r = (2^\alpha)^l > N$. There is a one-to-one correspondence between each element $x \in \mathbb{Z}_N$ and its Montgomery representation $xR \bmod N$. This Montgomery representation allows very efficient modular arithmetic especially for multiplication. Montgomery's method for multiplying two integers x and y modulo N avoids division by N, which is the most expensive operation in hardware. The method requires conversion of x and y to Montgomery representation and conversion of the calculation result back to $\mathbb{Z}_N$. The procedure is as follows. To compute $Z = xy \bmod N$, one first has to compute the Montgomery multiplication of x and $R^2 \bmod N$ to get $Z' = xR \bmod N$. $\text{Mont}(Z', y)$ gives the desired result. When computing the Montgomery product $T = \text{Mont}(x, y) = xyR^{-1} \bmod N$, the procedure shown in Algorithm 4.6 is performed [155].

In the original notation of Montgomery after each multiplication a reduction was needed (Step 7 in Algorithm 4.6). The input had the restriction $x, y < N$ and the output T was bounded by $T < 2N$. As a consequence, if $T > N$, N must be subtracted so that the output can be used as input of the next multiplication. To avoid this subtraction a bound for R is known [262] such that for inputs $x, y < 2N$ the output is also bounded by $T < 2N$.

In [258] the need of avoiding reduction after each multiplication is addressed. In practice this means that the output of the multiplication can be directly used as an input of the next Montgomery multiplication. We want to find a bound

on R such that with $X, Y < 2N$ the output of the Montgomery multiplication $T < 2N$. Write $R \geq kN$, then:

$$T = \frac{xy + mN}{R} = \frac{xy}{R} + \frac{m}{R}N < \frac{4}{k}N + N\,,$$

where $m = (xy \mod R)N' \mod R$ [18].

Hence, $T < 2N$ for $k \geq 4$, implying: $4N \leq R$. We will use $4N < R = 2^{l+2}$, by taking $\alpha = 1$ for simplicity and executing the iteration starting from Step 2 $l + 2$ times. As the result of the choice of α, $-N^{-1} \bmod 2^{\alpha}$ can be written as $(2 - n_0)^{-1} \bmod 2$. Because N is odd for RSA and an odd prime for ECC, $n_0 = 1$ which results in $N' = 1$. We will use Algorithm 4.7 for MMM which includes these improvements.

Algorithm 4.6: Montgomery modular multiplication over $GF(p)$ with final subtraction

Require: $N = (n_{l-1} \cdots n_1 n_0)_{2^\alpha}$, $x = (x_{l-1} \cdots x_1 x_0)_{2^\alpha}$, $y = (y_{l-1} \cdots y_1 y_0)_{2^\alpha}$ with $x, y \in [0, N-1]$, $R = (2^\alpha)^l$, $gcd(N, 2^\alpha) = 1$ and $N' = -N^{-1} \mod 2^\alpha$

Ensure: $T = xyR^{-1} \mod N$

1: T $\leftarrow 0$.
2: **for** i from 0 to $(l-1)$ **do**
3: $\quad m_i \leftarrow (t_0 + x_i y_0)\, N' \bmod 2^\alpha$
4: $\quad T \leftarrow (T + x_i y + m_i N)/2^\alpha$
5: **end for**
6: **if** $T \geq N$ **then**
7: $\quad T \leftarrow T - N$
8: **end if**

Algorithm 4.7: Montgomery modular multiplication over $GF(p)$ without final subtraction

Require: $N = (n_{l-1} \cdots n_1 n_0)_2$, $x = (x_l \cdots x_1 x_0)_2$, $y = (y_l \cdots y_1 y_0)_2$ with $x, y \in [0, 2N-1]$, $R = 2^{l+2}$, $gcd(N, 2) = 1$

Ensure: $T = xyR^{-1} \mod 2N$

1: $T \leftarrow 0$
2: **for** i from 0 to $l+1$ **do**
3: $\quad m_i \leftarrow (t_0 + x_i y_0) \bmod 2$
4: $\quad T \leftarrow (T + x_i y + m_i N)/2$
5: **end for**

All the operations will be computed modulo $2N$ through the EC point multiplication. The final round in the EC point multiplication is the conversion to

the integer domain, *i.e.*, calculating the Montgomery multiplication of the last result and 1. The same arguments as above prove that this final step remains within the following bound: $\text{Mont}(T, 1) \leq N$.

Our system can be divided hierarchically into three levels [196, 198]:

1. Systolic Array Cell: computes 1 bit of T in Step 4 of Algorithm 4.7.
2. Systolic Array: computes one iteration of Step 2 of Algorithm 4.7.
3. MMMC: computes the complete Algorithm 4.7.

In the following sections we describe the system using a bottom-up approach.

4.2.2.1 Systolic Array Cells

The *ith* iteration of Step 2 in Algorithm 4.7 computes the temporary results

$$T_i = 2^{-1}(T_{i-1} + x_i Y + m_i N)\,,$$

where $i = 0, \cdots, l+1$ and $T_{-1} = 0$ [259]. The jth bit of T_i is obtained using the recurrence relation

$$2^2 c1_{i,j} + 2c0_{i,j} + t_{i,j} = t_{i-1,j+1} + x_i y_j + m_i n_j + 2c1_{i,j-1} + c0_{i,j-1}\,, \quad (4.19)$$

for $i = 0, \cdots, l+1$, $j = 0, \cdots, l+1, c1_{i,-1} = 0$ and $c0_{i,-1} = 0$. In Eq. (4.19), $2c1_{i,j} + c0_{i,j}$, $j = -1, \cdots, l$, denotes the carry chain up the adder.

The regular cell of the systolic array consists of two FAs, one half-adder (HA) and two AND-gates as shown in Fig. 4.15. We can calculate m_i from the following equation:

$$m_i = (t_{i-1,1} + x_i y_0) \bmod 2 = t_{i-1,1} \oplus x_i y_0\,, \quad (4.20)$$

for $i = 0, \cdots, l+1$ and $t_{-1,1} = 0$. Here m_i is not an input to the rightmost cell, but obtained in the rightmost cell.

Because there is no carry input to the rightmost cell, the equation for calculating $t_{i,0}$ can be simplified as shown by Eq. (4.21).

$$2c0_{i,0} + t_{i,0} = t_{i-1,1} + x_i y_0 + m_i\,, \quad (4.21)$$

for $i = 0, \cdots, l+1$ and $t_{-1,1} = 0$. By combining Eq. (4.20) and Eq. (4.21), it can easily be shown that $t_{i,0} = 0$ and the equation for calculating $c0_{i,0}$ is as follows:

$$c0_0 = t_{i-1,1} + x_i y_0\,,$$

for $i = 0, \cdots, l+1$ and $t_{-1,1} = 0$. The rightmost cell of the systolic array consists of one AND, one OR and one XOR gate as shown in Fig. 4.16.

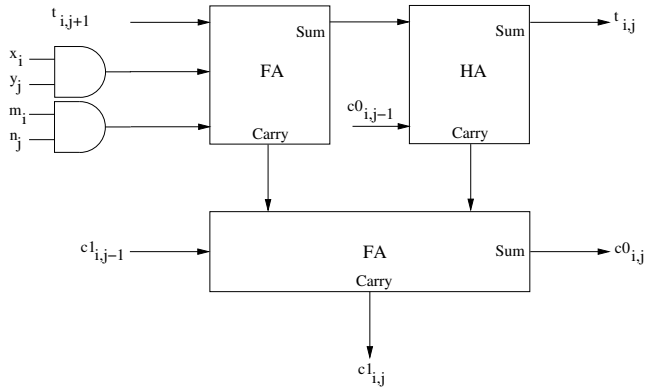


Figure 4.15: Schematic view of a regular cell in the systolic array used in the Montgomery modular multiplier over $GF(p)$

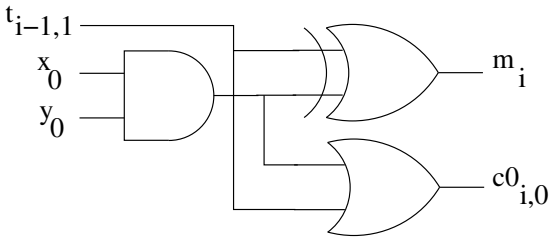


Figure 4.16: Schematic view of the rightmost cell in the systolic array used in the Montgomery modular multiplier over $GF(p)$

Because there is only one carry input from the rightmost cell, Eq. (4.19) can be simplified for $t_{i,1}$ as follows, which is implemented by the cell shown in Fig. 4.17. It consists of one FA, two HAs and two AND-gates.

$$2^2 c1_{i,1} + 2c0_{i,1} + t_{i,1} = t_{i-1,2} + x_i y_1 + m_i n_1 + c0_0 \,.$$

for $i = 0, \cdots, l+1$ and $t_{-1,2} = 0$.

Because $n_l = 0$, the equation of $t_{i,l}$ can be simplified as follows:

$$2t_{i,l+1} + t_{i,l} = t_{i-1,l+1} + x_i y_l + 2c1_{i,l-1} + c0_{i,l-1} \,,$$

for $i = 0, \cdots, l+1$ and $t_{-1,l+1} = 0$. This equation is implemented by the lth cell, which is shown in Fig. 4.18. This cell consists of one FA, one AND and one XOR-gate.

The ith row computes T_i from T_{i-1}. Each cell operates in a single clock cycle. Then the i, jth cell processes the bits of Eq. (4.19) at clock cycle time $2i + j$.

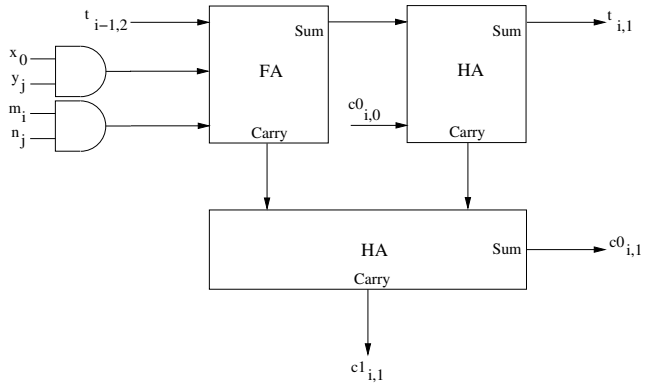


Figure 4.17: Schematic view of the first bit cell in the systolic array used in the Montgomery modular multiplier over $GF(p)$

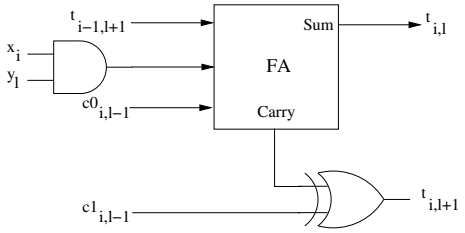


Figure 4.18: Schematic view of the leftmost cell in the systolic array used in the Montgomery modular multiplier over $GF(p)$

4.2.2.2 Systolic Array

In order to obtain a linear, pipelined modular multiplier, only one row of cells is taken. The jth cell behaves like cell (i, j), computing Eq. (4.19) at time $2i+j$ for $i = 0, \cdots, l+1$.

The schematic view of the systolic array is shown in Fig. 4.19. T denotes the intermediate value register. The carry chain is stored in the $C0$ and $C1$ registers.

Figure 4.19 shows that the $t_{i,j+1}$ output of the $(j+1)$-st cell is used as an input for the jth cell during the $(i+1)th$ iteration. This way the division by 2 in Step 4 of Algorithm 4.7 is realized.

The total area of the systolic array is $(5l-3)\text{XOR} + (7l-7)\text{AND} + (4l-5)\text{OR}$ gates and $4l$ flip-flops. The critical path is the same as the critical path of one regular cell and it is independent of the bit-length of the operands, hence equal

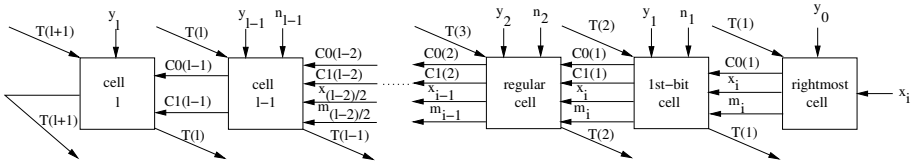


Figure 4.19: Schematic view of the one dimensional systolic array used in the Montgomery modular multiplier over $GF(p)$

to $2T_{FA} + T_{HA}$, where T_{FA} and T_{HA} are the critical paths of one FA and HA, respectively.

4.2.2.3 Modular Montgomery Multiplication Circuit

The MMMC has three l-bit data inputs X, Y and N, one START instruction input, one DONE output, which indicates that the operation is finished, and an l-bit RESULT output.

The MMMC is designed using the ASM approach; an ASM chart of the MMMC is shown in Fig. 4.21. For detailed information about the ASM approach, the reader is referred to Mano and Kim [148]. The circuit consists of a controller and a data path as shown in Fig. 4.20. The controller has four states, IDLE, MUL1, MUL2 and OUT. The data path consists of a systolic array, four internal registers, a counter and a comparator.

The controller stays in the IDLE state waiting for the START instruction. When the START input is set, the X, Y and N registers are loaded with the input values and the T register and the counter are reset.

In MUL1, the outputs of the systolic array cells are written to the T register and the controller goes to the MUL2 state. When the controller is in the MUL2 state, the counter is incremented by 1. When the counter value reaches $2(l+1)$, the comparator sets the "count-end" signal. Then the controller goes to the OUT state in which the value of the T register is written to the RESULT output and the acknowledgement signal DONE is set.

In the MUL2 state, the X register is shifted one bit to the right and the MSB of the X register is filled with 0. This ensures that, during the last iteration of Step 2 of Algorithm 4.7, the value of $x_{l+1,0}$ will be 0.

As mentioned before $t_{i,j}$ is calculated at the $(2i+j)th$ clock cycle, $i = 1, \cdots, l+2$ and $j = 1, \cdots, l$. Similarly $t_{l+2,l}$ is calculated at the $(2(l+2)+l)th$ clock cycle. Hence, the total number of clock cycles for completing one modular Montgomery multiplication equals $3l + 4$.

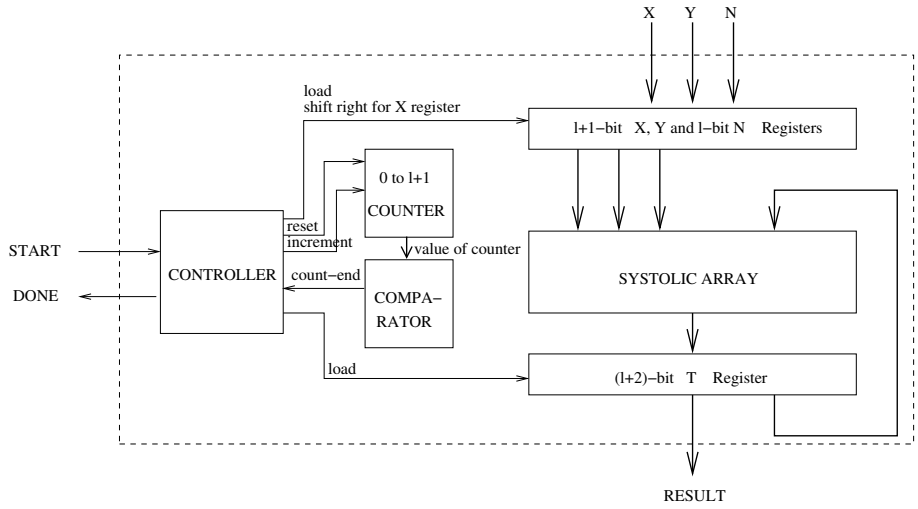


Figure 4.20: Architecture of the Montgomery modular multiplier circuit over $GF(p)$

In the previous work from Blum and Paar [25] the cells process u data bits in one clock cycle. The 3-bit control registers are put in the cells to control the output of four complex multiplexors. These bring a high latency on the critical path of one cell and as a consequence, the clock frequency is lower. In this work, cells process 1 bit in one clock cycle; the circuit is constructed using only combinational elements and the architecture is much simpler as shown in Figs. 4.15, 4.16, 4.17 and 4.18.

In [25], the total number of bits used for control logic is equal to $3\dot{\lceil} l/u \rceil$. The role of the control in their circuit and the implementation of the complete algorithm using it is not completely clear. In this work, the control logic requires $\log_2(l+2)+2$-bits, including a 2-bit state register, a $\log_2(l+2)$-bit counter and a comparator. It is implemented separately from the systolic array as shown in Fig. 4.20 and controls the execution of the modular Montgomery multiplication algorithm according to the ASM shown in Fig. 4.21.

Implementation Results of The Modular Montgomery Multiplication Circuit

The MMMC is implemented on a Xilinx V800-HQ (Virtex) FPGA. The number of slices (S), clock period (T_p), time-area product (TA) and time for one MMM (T_{MMM}) for different bit-lengths l are given in Table 4.2.

One can conclude from Table 4.2 that the clock frequency changes slightly with the bit-length. This property gives our circuit the advantage of suitability to

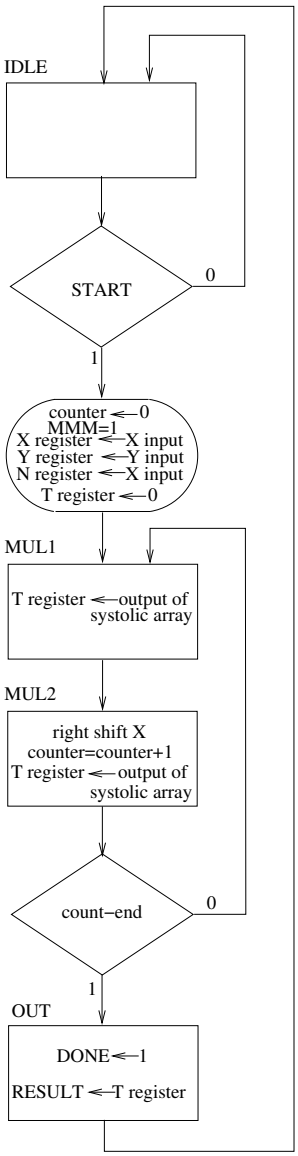


Figure 4.21: Algorithmic state machine chart of the Montgomery modular multiplier over $GF(p)$

various applications with different bit-lengths such as RSA and ECC.

Table 4.2: Number of slices (S), clock period (T_p), time-area products (TA) and time for one MMM (T_{MMM}) for different bit-lengths l on a Xilinx V800-HQ

l	# S	T_p ns	TA $S \cdot ns$	T_{MMM} μs
32	369	13.121	4841.65	1.311
64	697	12.252	8539.64	2.401
128	1355	12.846	17 406.33	4.984
256	2666	13.907	37 076.06	10.736
512	5290	15.036	79 540.44	23.155

4.3 FPGA Implementation of Elliptic Curve Cryptosystems over $GF(2^m)$

Our ECP can be divided into four hierarchical levels as shown in Fig. 4.22 [160, 159].

The operation blocks at each level from top to bottom are as follows:

- **Level 1:** Main Controller (MC)
- **Level 2:**
  1. affine to projective coordinates converter (AtoP)
  2. normal to Montgomery representation converter (NtoM)
  3. EC point multiplier (ECPM)
  4. projective to affine coordinates converter (PtoA)
  5. Montgomery to normal representation converter (MtoN)
- **Level 3:**
  1. EC point doubling circuit (ECPDC)
  2. EC point addition circuit (ECPAC)
  3. modular multiplicative inverter (MMI)
- **Level 4:**
  1. modular addition (MA)
  2. Montgomery modular multiplication circuit (MMMC)

The bit-length m, the word length ω and the key length l are input parameters to the circuit.

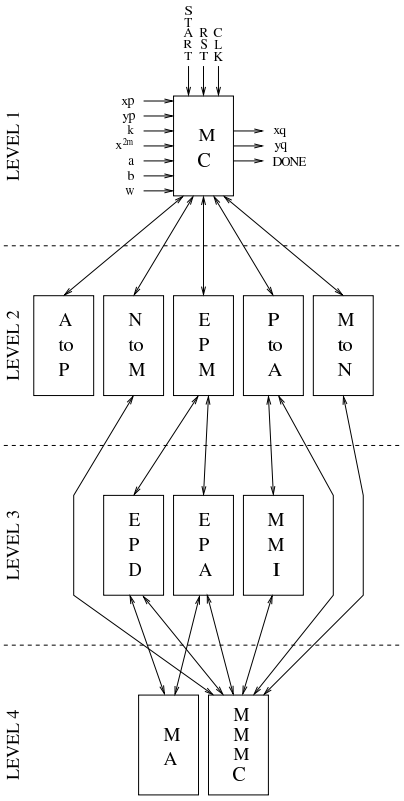


Figure 4.22: Block diagram of the elliptic curve point multiplier circuit over $GF(2^m)$

4.3.1 Elliptic Curve Point Doubling and Addition

Algorithm 4.8 and 4.9 realize an ECPA and an ECPD over $GF(2^m)$ based on [143], respectively. The inputs to the algorithms are points on the curve that are written in projective coordinates, as explained in Section 2.1.4.2. The output points are also in projective coordinates.

The doubling algorithm consists of eleven steps and needs three temporary registers. In each step a modular addition and/or a MMM is executed. Because MA and MMMC are separate hardware blocks, the operations can be performed in parallel. The first ten steps all contain a Montgomery modular multiplication that is sometimes accompanied by a modular addition, while the 11th step only consists of a modular addition. This makes the total execution time $10T_{MMM} + 1$, with T_{MMM} the latency of one MMM. In the same way, it can be found that the total execution time of one ECPA is $14T_{MMM} + 1$.

Algorithm 4.8: Elliptic Curve Point Addition over $GF(2^m)$

Require: $P_1 = (x, y, 1)$, $P_2 = (X_2, Y_2, Z_2)$
Ensure: $P_1 + P_2 = (X_3, Y_3, Z_3)$

1.	$T_1 \leftarrow Z_2^2$	
2.	$T_2 \leftarrow yT_1$	
3.	$T_3 \leftarrow xZ_2$	$T_2 \leftarrow T_2 + Y_2$
4.	$T_1 \leftarrow aT_1$	$T_3 \leftarrow T_3 + X_2$
5.	$T_4 \leftarrow Z_2T_3$	
6.	$T_3 \leftarrow T_3^2$	$T_1 \leftarrow T_4 + T_1$
7.	$T_1 \leftarrow T_3T_1$	
8.	$Z_3 \leftarrow T_4^2$	
9.	$T_4 \leftarrow T_2T_4$	
10.	$T_2 \leftarrow T_2^2$	$T_1 \leftarrow T_1 + T_4$
11.	$T_1 \leftarrow X_2Z_3$	$X_3 \leftarrow T_2 + T_1$
12.	$T_2 \leftarrow Y_2Z_3$	$T_1 \leftarrow X_3 + T_1$
13.	$T_1 \leftarrow T_4T_1$	$T_2 \leftarrow X_3 + T_2$
14.	$T_3 \leftarrow Z_3T_2$	
15.		$Y_3 \leftarrow T_1 + T_3$

4.3.2 Modular Multiplicative Inverter

Modular multiplicative inversion is computed using Fermat's theorem [127, 155], $a(x)^{-1} = a(x)^{2^m-2} \bmod p(x)$. This modular exponentiation of $a(x)$ by $2^m - 2 = (1, 1, \cdots 1, 1, 0)_2$, using the square-and-multiply algorithm [155], is shown in Algorithm 4.10.

The MMI circuit controls the execution of the square-and-multiply algorithm. It gets its START signal from the PtoA circuit. The MMI circuit then commands the MMMC to perform a MMM twice in every loop iteration in Algorithm 4.10 and once at the end.

4.3.3 Affine to Projective Representation Converter

As explained in Section 2.1.4.2, it is more efficient to compute point operations using projective instead of affine coordinates. The conversion to projective coordinates is as follows:

$$(xp(x), yp(x)) \rightarrow (X(x), Y(x), Z(x)) = (xp(x), yp(x), 1)$$

where $(xp(x), yp(x))$ is the point in affine coordinates and $(X(x), Y(x), Z(x))$ is the same point in projective coordinates.

Algorithm 4.9: Elliptic Curve Point Doubling over $GF(2^m)$

Require: $P_1 = (X_1, Y_1, Z_1)$
Ensure: $2P_1 = (X_3, Y_3, Z_3)$

1.	$T_1 \leftarrow Z_1^2$	
2.	$T_2 \leftarrow X_1^2$	
3.	$Z_3 \leftarrow T_1 T_2$	
4.	$T_1 \leftarrow T_1^2$	
5.	$T_2 \leftarrow T_2^2$	
6.	$T_1 \leftarrow bT_1$	
7.	$T_2 \leftarrow Y_1^2$	$X_3 \leftarrow T_1 + T_2$
8.	$T_3 \leftarrow aZ_3$	$T_2 \leftarrow T_2 + T_1$
9.	$T_1 \leftarrow T_1 Z_3$	$T_3 \leftarrow T_3 + T_2$
10.	$T_3 \leftarrow X_3 T_3$	
11.	$Y_3 \leftarrow T_3 + T_1$	

Algorithm 4.10: Modular multiplicative inversion over $GF(2^m)$

Require: polynomial $a(x)$, $0 \leq a(x) < p(x)$ and $p(x)$
Ensure: $b(x) = a(x)^{2^m-2} \bmod p(x) = a(x)^{-1} \bmod p(x)$

1: $b(x) \leftarrow a(x)$
2: **for** i from 2 to $m-2$ **do**
3: $\quad b(x) \leftarrow b(x)b(x) \bmod p(x)$
4: $\quad b(x) \leftarrow b(x)a(x) \bmod p(x)$
5: **end for**
6: $b(x) \leftarrow b(x)b(x) \bmod p(x)$

4.3.4 Normal to Montgomery Representation Converter

The conversion of $a(x)$ from the normal to the Montgomery representation is computed as $\text{Mont}(a(x), r(x)^2) = a(x)r(x) \bmod p(x)$. Multiplication by the MMMC of two polynomials that are in Montgomery representation will produce the Montgomery representation of the product as $\text{Mont}(a(x)r(x), b(x)r(x)) = a(x)b(x)r(x) \bmod p(x)$.

Modular addition of two polynomials that are in Montgomery representation will produce the Montgomery representation of the sum as $a(x)r(x) \bmod p(x) + b(x)r(x) \bmod p(x) = (a(x) + b(x))\, r(x) \bmod p(x)$. Because of these relations, the Montgomery representation of the coordinates of P, the coefficients a, b and the number 1 will be calculated at the beginning of the point multiplication by the NtoM circuit and all the operations during the EC point multiplication will be performed in Montgomery representation.

The conversion to Montgomery representation of the number 1 is computed as $\text{Mont}(1, r(x)^2) = r(x) \bmod p(x)$.

The other conversions in NtoM are performed by the following four operations:

$$\begin{aligned}
\text{Mont}(a(x), r(x)^2) &= a(x)r(x) \bmod p(x) \\
\text{Mont}(b(x), r(x)^2) &= b(x)r(x) \bmod p(x) \\
\text{Mont}(xp(x), r(x)^2) &= xp(x)r(x) \bmod p(x) \\
\text{Mont}(yp(x), r(x)^2) &= yp(x)r(x) \bmod p(x)\,.
\end{aligned}$$

4.3.5 Elliptic Curve Point Multiplier

The ECPM circuit controls the execution of Algorithm 2.1. In every iteration of the loop, an ECPD is executed. An ECPA is only performed when the evaluated key bit is 1.

4.3.6 Projective to Affine Coordinates Converter

After completing the ECPM the result point Q must be converted from projective coordinates to affine coordinates. This is done as $(X(x), Y(x), Z(x)) \rightarrow (xq(x),\ yq(x))$ such that $xq(x) = X(x)Z(x)^{-1}$ and $yq(x) = Y(x)Z(x)^{-2}$ [143]. PtoA controls four operations in the following order:

$$\begin{aligned}
Z(x)^{-1}r(x) \bmod p(x) &= \text{M}MIof\ Z(x) \\
\text{Mont}(X(x)r(x), Z(x)^{-1}r(x)) &= xq(x)r(x) \bmod p(x) \\
\text{Mont}(Z(x)^{-1}r(x), Z(x)^{-1}r(x)) &= Z(x)^{-2}r(x) \bmod p(x) \\
\text{Mont}(Y(x)r(x), Z(x)^{-2}r(x)) &= yq(x)r(x) \bmod p(x)\,.
\end{aligned}$$

4.3.7 Montgomery to Normal Representation Converter

Because the coordinates of the product point must be in normal representation, as a last action a conversion from Montgomery representation to normal representation is needed. This conversion requires two additional executions of the MMMC operation with the inputs $xq(x)r(x) \bmod p(x)$ and 1, then $yq(x)r(x) \bmod p(x)$ and 1, as $xq(x) = \text{Mont}(xq(x)r(x), 1)$, $yq(x) = \text{Mont}(yq(x)r(x), 1)$.

4.3.8 Implementation Results

The EC processor has been implemented on a Xilinx Virtex XCV800-4-HQ240 FPGA. The implementation results are given in Table 4.3. The number of gates in the table is not the same as ASIC gates. They are equivalent gates computed by the Xilinx Project Manager. The table shows that the minimum clock period and the area of the circuit decrease when the word length ω of

Table 4.3: Area and latency results of the elliptic curve processor implementations over $GF\left(2^{163}\right)$ for different word lengths (the key is 160-bit and its Hamming weight is 80)

w	1	4	8
Number of gates	138 528	149 037	150 678
Clock Period (ns)	19.956	19.977	20.923
Latency of ECP (ms)	9.652	7.249	3.801

MMM is enlarged. For a larger ω, the latency of the circuit becomes smaller, because the point multiplication can be done in fewer clock cycles.

The maximum clock frequency is independent of the bit-length m. The total latency of the elliptic curve point multiplication can be calculated as

$$(1900 + 14hw)\,T_{MMMC} + 316 + 2hw\,,$$

where $h\omega$ is the Hamming Weight of the key and T_{MMMC} is the latency of one MMM. $T_{MMMC} = m$ for $\omega = 1$ and $T_{MMMC} = 3\frac{m}{\omega}$ for $\omega > 1$. One ECPM requires 3.810 ms at 47 MHz. In order to verify the correctness of the execution of the circuit we simulated the behavior for $m = 8$ and $m = 16$ by the simulator provided by Xilinx Foundation software. Then we produced 10 000 test vectors by Magma, after implementing and uploading the circuit on the FPGA, we let the FPGA executing 10 000 EC point multiplications with the input points in the test vectors and a fixed key. We verified that the outputs are the same as the ones in the test vectors.

Because there is no previous work which uses MMM and all other designs use different platforms, it is hard to compare the area of the current work with the previous ones. The time needed for one ECPM with our design is approximately the same as the result reported in [5], smaller than the results reported in [249, 74, 75, 140, 63, 84, 116]. Unfortunately our circuit is slower than the ones reported in [192] and [60].

4.4 State of the Art for Elliptic Curve Cryptosystem Implementations over $GF\,(2^m)$

The following sections discuss earlier work on hardware implementations of elliptic curve point multiplication over $GF\,(2^m)$.

In 1989 Agnew *et al.* report the first result for performing elliptic curve operations in hardware [3]. To achieve this they used their earlier normal basis

multiplier as arithmetic unit and a Motorola M68008 as control unit. The throughput of the system is about 5 Kbps. Agnew *et al.* also used a Motorola M68030 as control unit and implemented a $GF\left(2^{155}\right)$ processor in [4]. If point multiplication by an integer with Hamming weight 30 is considered, this requires about 154 point doublings and 29 additions. The device is able to perform at least 145 integer multiplication per second, which corresponds to approximately 50 Kbps. In elliptic curve systems, the same base point P can be used repeatedly. Then all of the squares can be pre-computed, which increases the throughput by a factor of 4 to approximately 200 Kbps. The storage requirements for the point squarings is less than 6 Kbyte.

Agnew *et al.* describe how Diffie-Hellman and ElGamal protocols can be efficiently implemented using the group of an elliptic curve over a finite field; a VLSI implementation of an arithmetic processor in $GF\left(2^{155}\right)$ is discussed [5]. The finished device requires the equivalent of about 11 000 gates and runs at the design target speed of 40 MHz (clock).

Sutikno *et al.* also proposed a VLSI design and implementation of arithmetic processor over $GF\left(2^{155}\right)$ in [249]. In this work the field inversion is performed by the method of Agnew *et al.* [1]. It needs 23 field multiplications for $m = 155$. This method is used for accomplishing area optimization. The arithmetic processor has 32-bit wide internal bus. It has 15 instructions and 5 bit-length for its operation codes (3 bit LSB used for selecting instruction and 2 MSB for selecting the active register).

Sutikno *et al.* also presented the use of non-supersingular elliptic curve group over $GF\left(2^{155}\right)$ and a VLSI implementation of an ElGamal ECC processor in [251]. There are eleven 155-bit wide registers to store data and intermediate results from the arithmetic process. In the control module, there are five control blocks: main control, repeat square and multiply control, adding point control, doubling point control and inversion control. The throughput rate for encryption is estimated as $6.5 \cdot 10^{-4}$ bit/clock cycle and for decryption $13.1 \cdot 10^{-4}$ bit/clock cycle. The area on FPGA is estimated as 40 000-50 000 gates.

Gao *et al.* proposed in [74, 75] an ECC coprocessor with variable key size, which utilizes the internal static random access memory (SRAM) in an FPGA. The controller is the kernel of the scalar multiplier and has the format of a finite state machine (FSM) with a table look-up to implement the logic functions. All operations can be categorized as one of the following atomic operations: unconditional jump, conditional jump, operand load, operand store, finite field addition, finite field squaring, finite field multiplication and finite field inversion. The scalar is decomposed as a non-adjacent form (NAF) and the EC point multiplication is computed with a series of addition/subtractions of EC points.

The cost of one EC point multiplier with key size m is given as:

- total flip-flops (FFs) $= 21m + 3\log_2 m + 48$;
- total function generators (FGs) $= 24m + 3\log_2 m + 308$;
- minimal number of CLBs $= 12m + (3\log_2 m)/2 + 154$; and
- maximal number of CLBs $= 45m + 6\log_2 m + 356$.

Hauck *et al.* proposed an ECC chip using asynchronous wave pipelines (AWPs) in [103]. They give the simulation results of their design and report that the circuit runs at a rate of 1.5 GHz in 0.35 μm CMOS technology.

Leung *et al.* described a Xilinx Virtex based FPGA implementation of an EC processor in [140]. It consists of an arithmetic logic unit (ALU), register file, a microcode sequencer and microcode storage. In the ALU a Massey-Omura Multiplier is used. A $16 \times n$-bit dual-port synchronous register file is constructed from the 16×1-bit distributed random access memory (RAM) feature of the Xilinx Virtex series. Apart from instructions which directly control the ALU, there are three types of jump instructions: JMP-jump unconditionally, JKZ-jump if the least significant bit of K counter is zero and JCZ-jump if the C register is zero. Another FPGA implementation for 270-bit operations similar to the previous was proposed by Ernst *et al.* in [63]. They used a XC4085XLA FPGA for implementation and reported that 180 000 system gates were used. The clock speed was 34 MHz and resulting performance is 146 point multiplications per second.

Orlando and Paar proposed a scalable elliptic curve processor architecture which operates over $GF(2^m)$ in [192]. The EC processor (ECP), consists of three main components. These components are the main controller (MC), the arithmetic unit controller (AUC) and the arithmetic unit (AU). The AU performs the $GF(2^m)$ field additions, squares, multiplications and inversions under AUC control. The MC executes the double-and-add and the Montgomery scalar multiplication functions, the AUC performs all the other subroutines and the AU is the hardware that computes the finite field operations. Three ECP prototypes were built. These prototypes support ECs over the field $GF\left(2^{167}\right)$, with this field being defined by the field polynomial $F(x) = x^{167} + x^6 + 1$. Each prototype used a 16-bit MC processor with 256 words of program memory, a 24-bit AUC processor with 512 words of program memory and 128 registers, each of which is 167 bits wide. They also provided a 32-bit I/O interface to the host system. The prototypes used least significant digit (LSD) multipliers with digit sizes equal to 4, 8 and 16. The prototypes were implemented using the Xilinx's XCV400E8BG432 (Virtex E) FPGA. The prototype implementations used between 15% and 28% of the look-up tables (LUTs) (depending on the digit size), 16% of the FFs and 25% of the Block RAMs available in the XCV400E8BG432 FPGA.

Janssens *et al.* proposed a high-speed hardware/software co-design for computing EC point multiplications over $GF(2^m)$ and implemented it on an Atmel

Field Programmable System Level Integration Circuit (FPSLIC) [116]. The design consists of a Data-path, which performs the finite field arithmetic and two FSMs that each control a particular part of the functionality at a particular level of hierarchy. At the highest level the Software Controller is the master. It gives instructions to the Hardware Controller, which translates these instructions in a sequence of direct control signals for the operators in the Data-path.

The design flow which is proposed by the Atmel System Designer software was followed. On average, $125\,m$ instructions for the AVR are needed to read in data, perform the point multiplication and write out the results. Since the embedded AVR core achieves a throughput approaching 1 MIPS per MHz, this corresponds with $125\,m$ clock cycles. The critical path after synthesis is 33 ns. The corresponding maximum clock frequency is nearly 30 MHz. After place-and-route it turned out that 23 CLBs are needed per bit slice and 496 CLBs for the hardware controller. Since there are only 2304 CLBs available on the FPGA, a design for a key length of 72 bits can be implemented.

The results above are used to estimate the total time it takes to perform a complete EC point multiplication. Since the Software Controller is always waiting on an interrupt from the Hardware Controller before sending a new instruction, the Hardware Controller receives this new instruction almost directly after it has finished performing the former instruction. Therefore we can take only the number of clock cycles needed by the hardware part, which is on average $12m^2$.

Schaumont and Verbauwhede present an elliptic curve processor over $GF\,(2^m)$ in [227, 228]. The architecture has a layered structure with the layers corresponding to the operations described in the security pyramid shown in Fig. 4.23. The authors propose a language and simulation environment that allows to explore the design of security domain specific processors at a high abstraction level.

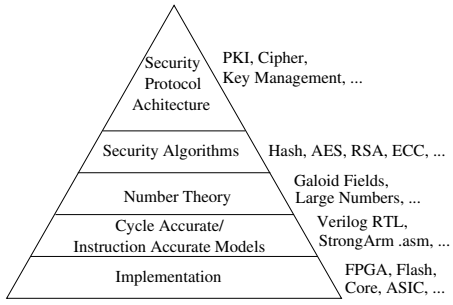


Figure 4.23: Security pyramid

Bednara *et al.* compare several approaches to hardware implementation of ECCs in [20]. They especially focused on FPGA based implementations. Kim

and Lee presented in [122] an ECP which consists of control unit, arithmetic unit and register unit.

We present a hardware architecture of a processor for an EC cryptosystem over the finite field $GF(2^m)$ in Section 4.3. We use Montgomery modular multiplier over $GF(2^m)$ proposed by Koç and Acar in [130] in our elliptic curve processor. The maximum clock frequency is independent of the bit length m. The total latency of the ECP can be calculated as $(1900 + 14hw)\,T_{MMMC} + 316 + 2hw$, where hw is the Hamming weight of the key and T_{MMMC} is the latency of one MMM. $T_{MMMC} = m$ for $w = 1$ and $T_{MMMC} = 3\frac{m}{w}$ for $w > 1$. One elliptic curve point multiplication (ECPM) requires 3.810 ms at 47 MHz.

Creating a working implementation was a significant challenge in the 1980s; the number of hardware implementations that made it to prototype or production phase was very limited. In the 1990s, we have seen significant progress due to a combination of better algorithms and advances in VLSI technology.

Reviewing various hardware architectures for ECC one conclusion is imposing itself. Hardware as proposed in the 1980s is still relevant in the sense of its fundamentals such as finite field arithmetic, the hierarchy of the operations, etc. The algorithms for ECC are constantly being pushed by current applications to be even faster, in order to fulfill industry and government demands.

The implementations that were proposed in the last years use the software/hardware co-design technique. Only the basic operations are implemented in hardware and the software part of the implementation call these coprocessors. This approach gives the possibility to use the same hardware for different applications. The performance of these designs will be less than the performance of a fixed design, but they will be more flexible.

There is no doubt that in the coming years even more performant hardware implementations will be developed for high-end applications. We believe that the biggest challenge ahead may be the development of very compact and inexpensive low-power implementations that allow protection for personal and wireless devices. A second challenge is to develop efficient implementations that offer adequate security against sophisticated side-channel attacks.

4.5 FPGA Implementation of Elliptic Curve Cryptosystems over $GF(p)$

In this section we present the results of our elliptic curve processor over $GF(p)$. The elliptic curve we use in this design is defined by Weierstrass equation as Eq. (2.2). Our ECP can be divided into five hierarchical levels as shown in Fig. 4.24 [202, 195, 197, 16, 198]. The operation blocks on each level from top

to bottom are as follows:

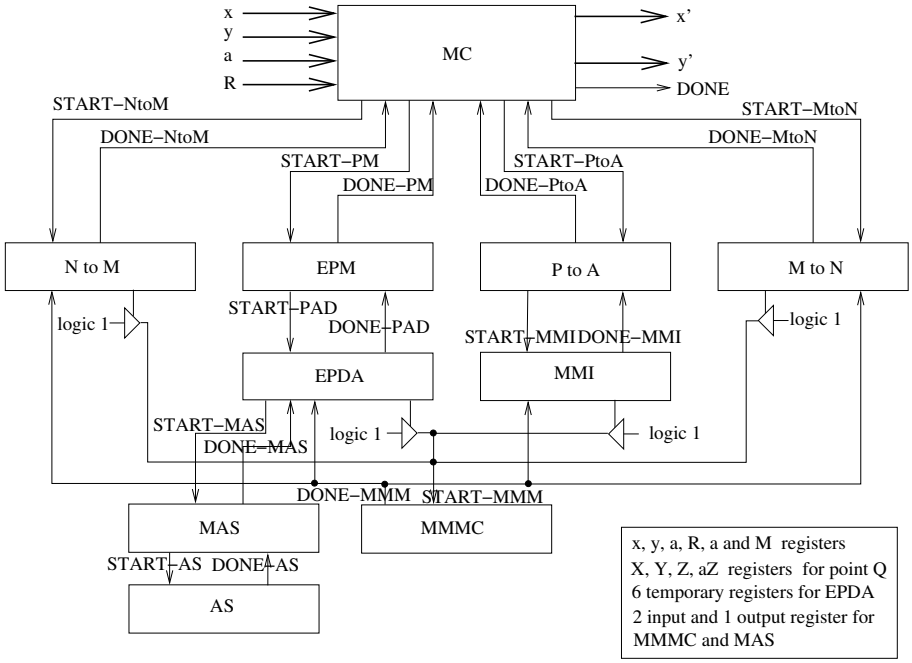


Figure 4.24: Block diagram of the elliptic curve processor over $GF(p)$

- **Level 1:** Main Controller (MC)
- **Level 2:**
  1. Affine to projective coordinates converter (AtoP):
     $(x, y) \rightarrow (X, Y, Z, aZ^4)$ such that $X = x$, $Y = y$, $Z = 1$, $aZ^4 = a$.
  2. Normal to Montgomery representation converter (NtoM)
  3. EC point multiplier (EPM)
  4. Projective to affine coordinates converter (PtoA)
  5. Montgomery to normal representation converter (MtoN)
- **Level 3:**
  1. EC Point doubling, addition circuit (EPDA)
  2. Modular Multiplicative Inverter (MMI)
- **Level 4:**
  1. Montgomery Modular Multiplication Circuit (MMMC)

2. Modular Addition, Substraction circuit (MASC)

- **Level 5:** Addition, Substraction circuit (ASC)

For simplicity all blocks were designed separately with their own FSMs and data paths. This allows for independent optimization and testing of the building blocks. The VHDL code was written by describing the bit-length N of the coordinates x and y of P and the bit-length l of k as parameters. Hence this design is suitable for any N and l. In the following sections we have described the system using a top-down approach.

4.5.1 Main Controller

MC includes an FSM with five states. The ASM chart [148] of the MC is shown in Fig. 4.25. The START signal is the instruction signal from the host. MC issues the instructions NtoM to start conversion from normal to Montgomery representation, EPM to start point multiplication, PtoA to start conversion from projective to affine coordinates and MtoN to start a conversion from Montgomery to normal representation one after another by setting the START-NtoM, START-PM, START-PtoA and START-MtoN signals, respectively. The DONE-NtoM, DONE-PM, DONE-PtoA and DONE-MtoN signals indicate that the related operations are finished. The DONE signal indicates to the host that a complete EC point multiplication operation is finished and the results are ready on the output ports.

4.5.2 Normal to Montgomery Representation Converter

The conversion of an integer x from the normal representation to the Montgomery representation is calculated as

$$\mathrm{Mont}(x, R^2) = xR^2R^{-1} \bmod M = xR \bmod M\,.$$

This conversion operation is similar to the normal to Montgomery representation conversion over $GF\,(2^m)$ presented in Section 4.3.4. Multiplication by MMMC of two numbers that are in Montgomery representation will produce the Montgomery representation of their product as $\mathrm{Mont}(xR, yR) = xRyRR^{-1} \bmod M = xyR \bmod M$. Modular addition and subtraction of two numbers that are in Montgomery representation will produce the Montgomery representation of the sum or difference as $xR \bmod M \pm yR \bmod M = (x \pm y)R \bmod M$. Because of these relations, the Montgomery representation of the coordinates of P, the coefficient a and the number 1 will be calculated in the beginning of the point multiplication by the NtoM circuit and all the operations during the EC point multiplication will be computed in Montgomery representation.

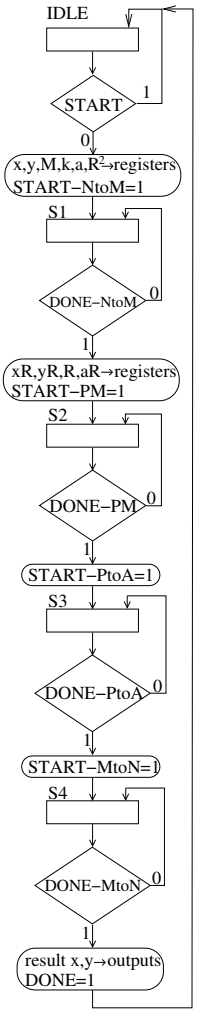


Figure 4.25: Algorithmic state machine chart of the main controller in the elliptic curve processor over $GF(p)$

NtoM includes an FSM with 5 states. The ASM chart of NtoM is shown in Fig. 4.26. NtoM waits in first (ini-IDLE) state until the START-NtoM signal from MC is set. NtoM lets the MMMC to execute 4 MMMs,

$$\begin{aligned} \text{Mont}(1, R^2) &= R \bmod M, \\ \text{Mont}(x, R^2) &= xR \bmod M, \\ \text{Mont}(y, R^2) &= yR \bmod M, \\ \text{Mont}(a, R^2) &= aR \bmod M. \end{aligned}$$

After DONE-MMM is set in the last state, NtoM sets DONE-NtoM signal and goes back to (ini-IDLE) state.

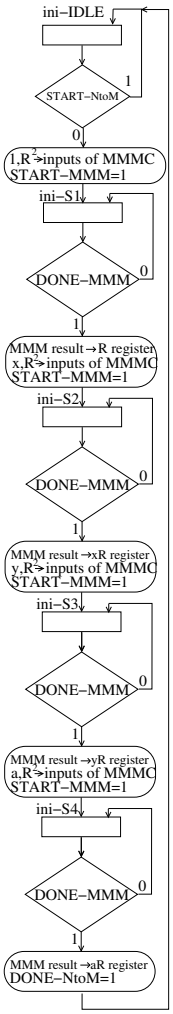


Figure 4.26: Algorithmic state machine chart of the normal to Montgomery representation converter circuit in the elliptic curve processor over $GF(p)$

4.5.3 Elliptic Curve Point Multiplier

The ECPM includes an FSM with four states to control the execution of Algorithm 2.1. The ASM chart of ECPM is shown in Fig. 4.27. The circuit stays in the first (mul-IDLE) state until the START-PM signal from the MC is set. The

DONE-PM signal indicates that the scanning of the bits of k is finished, so the result of the operation can be read from the output ports. The ECPM instructs the EPDA to start a point double operation by setting the START-PAD signal and resetting the ADD-DOUBLE signal and a point addition operation by setting the START-PAD and ADD-DOUBLE signals. The DONE-PAD from the EPDA indicates that a point double or addition operation is finished.

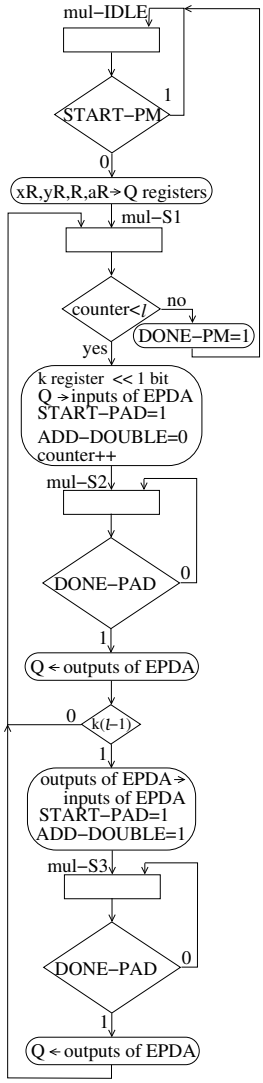


Figure 4.27: Algorithmic state machine chart of the elliptic curve point multiplication circuit in the elliptic curve processor over $GF(p)$

4.5.4 Projective to Affine Coordinates Converter

After finishing the EC point multiplication, the resulting point Q must be converted from J^m coordinates to affine coordinates. This is done as follows [43]:

$$\left(X, Y, Z, aZ^4\right) \rightarrow (x, y) \text{ such that } x = XZ^{-2} \text{ and } y = YZ^{-3}\,.$$

PtoA includes an FSM with six states to control the above operations. PtoA waits in the first (PtoA-IDLE) state until the signal START-PtoA from MC is set. After it is set, PtoA visits the other five states in the following order and after DONE-MMM signal from the MMM circuit is set in the (PtoA-S5) state, PtoA sets DONE-PtoA signal and goes back to (PtoA)-IDLE state.

- PtoA-S1: $Z^{-1}R$ =Modular Multiplicative Inversion of Z
- PtoA-S2: $Z^{-2}R = \text{Mont}(Z^{-1}R, Z^{-1}R)$
- PtoA-S3: $xR = XZ^{-2}R = \text{Mont}(XR, Z^{-2}R)$
- PtoA-S4: $Z^{-3}R = \text{Mont}(Z^{-1}R, Z^{-2}R)$
- PtoA-S5: $yR = YZ^{-3}R = \text{Mont}(YR, Z^{-3}R)$

4.5.5 Montgomery to Normal Representation Converter

Because the coordinates of the product point must be in normal representation, as a last action a conversion from Montgomery representation to normal representation is needed. This conversion requires two additional executions of the MMM operation with the inputs xR and 1, then yR and 1, as $x = \text{Mont}(xR, 1) = xRR^{-1}$, $y = \text{Mont}(yR, 1) = yRR^{-1}$.

4.5.6 Elliptic Curve Point Doubling, Addition

When we convert the input point P from affine coordinates to projective coordinates we take Z as 1. The J^m representation of $P(x, y)$ is $(x, y, 1, a)$. The EC point multiplication is calculated by Algorithm 2.1. One of the input points of the EC point addition at Step 5 of Algorithm 2.1 is always P. According to these properties we can take $Z_1 = 1$ and simplify Eq. (2.4) for EC point addition in projective coordinates as follows:

$$\begin{aligned}
&U_1 = X_1Z_2^2,\ S_1 = Y_1Z_2^3,\ H = X_2 - U_1,\ r = Y_2 - S_1,\\
&X_3 = -H^3 - 2U_1H^2 + r^2,\ Y_3 = -S_1H^3 + r\left(U_1H^2 - X_3\right),\\
&Z_3 = Z_2H,\ aZ_3^4 = aZ_3^4\,.
\end{aligned}$$

Since both the MMMC and the modular addition/subtraction (MAS) circuits are available, these operations can be executed in parallel. EC point addition

Algorithm 4.11: Elliptic curve point addition over $GF(p)$

Require: $P_1 = (x, y, 1, a)$, $P_2 = (X_2, Y_2, Z_2, aZ_2^4)$
Ensure: $P_1 + P_2 = P_3 = (X_3, Y_3, Z_3, aZ_3^4)$

1.	$T_1 \leftarrow Z_2^2$	
2.	$T_2 \leftarrow xT_1$	
3.	$T_1 \leftarrow T_1 Z_2$	$T_3 \leftarrow X_2 - T_2$
4.	$T_1 \leftarrow yT_1$	
5.	$T_4 \leftarrow T_3^2$	$T_5 \leftarrow Y_2 - T_1$
6.	$T_2 \leftarrow T_2 T_4$	
7.	$T_4 \leftarrow T_4 T_3$	$T_6 \leftarrow 2T_2$
8.	$Z_3 \leftarrow Z_2 T_3$	$T_6 \leftarrow T_4 + T_6$
9.	$T_3 \leftarrow T_5^2$	
10.	$T_1 \leftarrow T_1 T_4$	$X_3 \leftarrow T_3 - T_6$
11.	$aZ_3^4 \leftarrow Z_3^2$	$T_2 \leftarrow T_2 - X_3$
12.	$T_3 \leftarrow T_5 T_2$	
13.	$aZ_3^4 \leftarrow \left(aZ_3^4\right)^2$	$Y_3 \leftarrow T_3 - T_1$
14.	$aZ_3^4 \leftarrow a\left(aZ_3^4\right)$	

(ECPA) and doubling (ECPD) can be realized by Algorithm 4.11 and Algorithm 4.12, respectively.

Fourteen states and six temporary registers are needed for EC point addition and also for EC point doubling. Because completing one MAS operation takes a shorter time than one MMM, the latency of one state is the same as for one MMM. Hence the total execution time of an EC point addition is $14T_{MMM}$, with T_{MMM} the latency of one MMM. The total execution time of an EC point doubling is $8T_{MMM} + 6T_{MAS}$, with T_{MAS} the latency of one MAS.

4.5.7 Modular Multiplicative Inverter

Modular multiplicative inversion is calculated according to Fermat's theorem, $a^{-1} = a^{p-2} \bmod p$, if $gcd(a, p) = 1$ [127, 155]. Because the curves we are interested in are defined over $GF(p)$ (with p prime), we can use this theorem to find multiplicative inverses modulo p. Hence multiplicative inversion can be performed by modular exponentiation of a by $p - 2$. Modular exponentiation can be realized by using the square and multiply algorithm given in [155].

The MMI controls the execution of the square and multiply algorithm. It includes an FSM with four states. The ASM chart of MMI is shown in Fig. 4.28. The START-INV signal is the instruction signal from PtoA. The DONE-INV signal indicates that the scanning of the bits of the $T1$ register is finished.

Algorithm 4.12: Elliptic curve point doubling over $GF(p)$

Require: $P_1 = (X_1, Y_1, Z_1, aZ_1^4)$
Ensure: $2P_1 = P_3 = (X_3, Y_3, Z_3, aZ_3^4)$

1.	$T_1 \leftarrow Y_1^2$	$T_2 \leftarrow 2X_1$
2.	$T_3 \leftarrow T_1^2$	$T_2 \leftarrow 2T_2$
3.	$T_1 \leftarrow T_2 T_1$	$T_3 \leftarrow 2T_3$
4.	$T_2 \leftarrow X_1^2$	$T_3 \leftarrow 2T_3$
5.	$T_4 \leftarrow Y_1 Z_1$	$T_3 \leftarrow 2T_3$
6.	$T_5 \leftarrow T_3 \left(aZ_1^4\right)$	$T_6 \leftarrow 2T_2$
7.		$T_2 \leftarrow T_6 + T_2$
8.		$T_2 \leftarrow T_2 + \left(aZ_1^4\right)$
9.	$T_6 \leftarrow T_2^2$	$Z_3 \leftarrow 2T_4$
10.		$T_4 \leftarrow 2T_1$
11.		$X_3 \leftarrow T_6 - T_4$
12.		$T_1 \leftarrow T_1 - X_3$
13.	$T_2 \leftarrow T_2 T_1$	$aZ_3^4 \leftarrow 2T_5$
14.		$Y_3 \leftarrow T_2 - T_3$

4.5.8 Implementation Results of The Elliptic Curve Processor

The proposed processor is implemented on a Xilinx V800-HQ-240-4 Virtex FPGA by taking the bit-length of EC parameters N and the bit-length of k, l as 160. According to the implementation results, the number of flip-flops and 4-input LUTs are equal to 11 192 and 14 393, respectively. This is equivalent to 175 952 gates. The minimum clock period is 40.550 ns (maximum clock frequency: 24.661 MHz). LUTs are lookup-tables that are used as RAMs or 4-input gates. The latency of the operations according to the clock frequency of the implemented circuit is given in Table 4.4. In order to verify the correctness of the execution of the circuit we simulated the behavior for two p values with bit-length 8 and 16 by the simulator provided by Xilinx Foundation software. Then we produced 10 000 test vectors by Magma, after implementing and uploading the circuit on the FPGA, we let the FPGA executing 10 000 EC point multiplications with the input points in the test vectors and a fixed key. We verified that the outputs are the same as the ones in the test vectors.

The only existing previous work done on FPGA is from Orlando and Paar [194]. They reported that their processor used 11 416 LUTs, 5735 flip-flops and 35 BlockRAMs; BlockRAM is a block memory on Virtex FPGAs. On their FPGA one BlockRAM consists of 4096 bits of memory. The clock frequency was reported as 40 MHz. If we compare both results, we can say that our processor uses less memory.

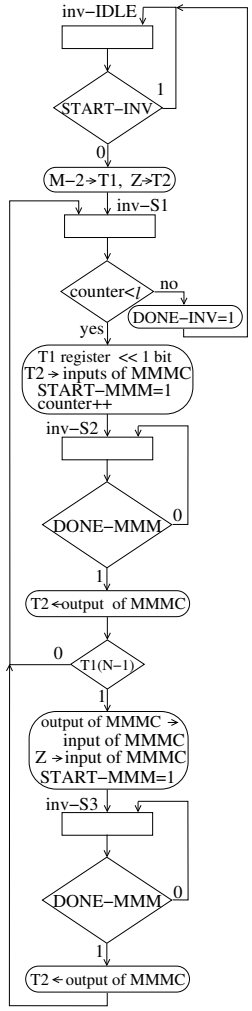


Figure 4.28: Algorithmic state machine chart of the modular multiplicative inverter in the elliptic curve processor over $GF(p)$

We use our ECP to estimate the performance of one signature generation and verification. The ECDSA algorithm is given in Section 2.2. A circuit to execute SHA-1 is needed. We will use the performance data for SHA-1 given by Helion Technology Lim. in [104].

The operations used for ECDSA signature generation starting from the second step are as follows:

1. Elliptic Curve point multiplication: 56.705 ms. One modular reduction:

Table 4.4: Latency of the operations executed in the elliptic curve processor over $GF(p)$

Operation	Sub-operations	# of clock cycles	Latency* ms
NtoM	4 MMM	$12N + 16$	0.078
EPM	l ECPD+ $l/2$ ECPA	$l(51N + 66)$	53.370
PtoA	MMI+4 MMM	$3N^2 + 16N + 16$	3.218
MtoN	2 MMM	$6N + 8$	0.039
ECPD	8 MMM+6 MAS	$40N + 38$	0.261
ECPA	14 MMM	$42N + 56$	0.275
MMI	$3N/2$ MMM	$9/2N^2 + 6N$	4.710
MMM		$3N + 4$	0.02
MAS		$2N + 1$	0.013

* for $N = l = 160$ at 24.661 MHz

0.013 ms.

2. Modular multiplicative inversion: 4.71 ms
3. SHA-1: 0.009 ms
4. Two multiplications: 0.08 ms. One modular addition: 0.013 ms

Total latency for one signature generation with ECDSA is 61.53 ms.

The operations used for ECDSA signature verification are as follows:

1. SHA-1: 0.009 ms
2. Modular multiplicative inversion: 4.71 ms
3. Two multiplications: 0.08 ms
4. Two elliptic curve point multiplications: 56.705 ms. One elliptic curve point addition: 0.275 ms. One modular reduction: 0.013 ms.

Total latency for one signature generation with ECDSA is 61.873 ms.

4.6 State of the Art for Elliptic Curve Cryptosystem Implementations over $GF(p)$

To the best of our knowledge, the first documented ECC processor over $GF(p)$ before our design was proposed in Orlando and Paar [194]. This so-called

Elliptic Curve Processor (ECP) is scalable in terms of area and speed and especially suited for FPGAs. The ECP is best suited for projective coordinates and it uses a new type of high-radix precomputation-based MMM. It consists of three main components: the MC, the AUC and the AU. The MC is controlling the point multiplication. The AUC is responsible for point operations and it controls the AU. The AU is in charge of $GF(p)$ operations. It consists of a register file, an adder and a multiplier as the most critical component. This multiplier performs a generalized version of the MMM algorithm with quotient pipelining introduced in Orup [203]. This version supports positive and negative operands and features Booth recoding and precomputation. Positive and negative numbers appear often in ECC algorithms and both are equally treated as subtraction has the same cost as addition. The proposed ECP architecture was verified on an example of the field $GF\left(2^{192} - 2^{64} - 1\right)$ which is one of the fields recommended by FIPS 186-2 in [174]. The scalability of the multiplier to larger fields was also verified in the field whose size is 521 bits. The authors have estimated eventual a timing of 3 ms for computing one point multiplication. The authors estimate that it takes 3 ms to compute one 192-bit point multiplication. However, this estimate assumes 100% throughput from the multiplier; the expected latency was not considered.

In this thesis we present a FPGA implementation of an elliptic curve processor [202, 195, 197, 16, 198]. The processor consists of special operational blocks for Montgomery Modular Multiplication, modular addition/subtraction, EC Point doubling/addition, modular multiplicative inversion, EC point multiplier, projective to affine coordinates conversion and Montgomery to normal representation conversion. Hence it can be programmed by the host to execute any of these operations in any order. It is possible to use the proposed processor not only for ECC, but also for any system for which modular arithmetic operations are essential, such as the RSA cryptosystem.

The basic operations are MMM and MAS. The other blocks include a FSMs, which control the execution of these operations. The critical path depends only on the critical path of the circuits for MMM and MAS. The architecture of these blocks is designed to ensure a short critical path to allow for high clock frequencies which are independent from the bit-length of the EC parameters.

Our processor is implemented on a Xilinx V800-HQ-240-4 Virtex FPGA by taking the bit-length of EC parameters N and the bit-length of key, l as 160. According to the implementation results, the number of flip-flops and 4-input LUTs are equal to 11 192 and 14 393, respectively. This is equivalent to 175 952 gates. The minimum clock period is 40.550 ns.

Orlando and Paar use a multiplier which is also based on the MMM algorithm but it is a generalized version with quotient pipelining introduced by Orup in [203]. We use the basic MMM algorithm from which we only exclude the modular reduction as a result of the bound adjustment. In this way

no pre-computation is required, which results in a substantial memory reduction. More importantly, this property also facilitates modular exponentiation as temporary results of multiplications can be fed back directly without any modular subtraction. Their multiplier has a semi-systolic architecture while the multiplier presented here is fully systolic. This results in an increased flexibility which is unrelated to any specific parameter choice. Orlando and Paar also used an adaptation of a fixed base exponentiation method as introduced by Brickell *et al.* in [29]. This algorithm is assumed to be four times faster than standard double-and-add algorithm which we use. However, it involves a known point calculation which is a limiting factor with respect to various applications of ECC.

Goodman and Chandrakasan proposed a domain-specific reconfigurable cryptographic processor (DSRCP) in [84]. The instruction set definition of the DSRCP was dictated by the IEEE 1363 Public Key Cryptography Standard document [108]. A list of the arithmetic functions required to implement the various primitives defined in the standard was tabulated in a functional matrix, which was then used to define the instruction set architecture (ISA) of the processor. The ISA contains 24 instructions broken up into six types of operations: conventional arithmetic, modular integer arithmetic, GF arithmetic, EC field arithmetic over GF, register manipulation and processor configuration. The processor consists of four main architectural blocks: the global controller and microcode read only memories (ROMs), the I/O interface, the shutdown controller and the reconfigurable datapath. Operands used within the processor can vary in size from 8 to 1024 bits requiring the use of a flexible I/O interface that allows the user to transfer data to/from the processor in a very efficient manner. The primary component of the DSRCP is the reconfigurable datapath. The datapath consists of four major functional blocks: an eight-word register file, a fast adder unit, a comparator unit and the main reconfigurable logic unit.

The DSRCP performs a variety of algorithms ranging from modular integer arithmetic to EC arithmetic over GF. All operations are universal in that they can be performed using any valid n-bit modulus ($8 \leq n \leq 1024$), $GF\,(2^m)$ field polynomial and non-supersingular elliptic curve over $GF\,(2^n)$. The various complex modular arithmetic operations (multiplication, reduction, inversion and exponentiation) are implemented using microcode, while simple operations (addition and subtraction) are implemented directly in hardware using the wide adder and comparator units. Multiplication is performed using MMM [168].

The processor is fabricated in a 0.25 μm CMOS technology with five levels of metallization. The core contains 880 000 devices and measures $2.9 \times 2.9\,\mathrm{mm}^2$. The datapath consists of 1024 processing bit-slices, each of which measures $30 \times 150\,\mu\mathrm{m}^2$. At 50 MHz, the processor operates at a supply voltage of 2 V and consumes at most 75 mW of power. In ultra-low-power mode (3 MHz at $V_{DD} = 0.7V$), the processor consumes at most 525 μW.

Wolkerstorfer has proposed a dual-field arithmetic unit that offers all instructions required for both types of finite fields: $GF(p)$ and $GF(2^m)$ in [271]. He uses a redundant number representation and a special multiplication with interleaved modular reduction. Inversion is performed by the Extended Euclidean Algorithm. This is a low-power architecture that can be realized on a moderate silicon area; the author claims that it requires just a little more hardware resources than for a pure $GF(p)$ multiplier.

The idea of unified multiplier was first introduced by Savaş *et al.* in [226]. The authors presented a scalable and unified architecture for a Montgomery multiplication module. They deployed an array of word size processing units organized in a pipeline. The same idea forms the basis of the design of Grosschädl in [87]. His bit-serial multiplier performs multiplications in both types of fields. The author also modifies the classical MSB-first version for iterative modular multiplication. All concepts are introduced in detail, but the actual VLSI implementation is not given. Most recent published work is the one of Satoh and Takano [224]. They presented a dual field multiplier with the best performance so-far in both type of fields. The throughput of EC scalar multiplication is maximized by Montgomery multiplier and a on-the-fly redundant using binary converter. The great quality of their design lies in the scalability in operand size and also the flexibility between speed and hardware area.

4.7 Conclusions

We have presented an overview of the wide variety of architectures which have been designed to implement elliptic curve cryptosystems. In the 1990s, we have seen significant progress due to a combination of better algorithms and advances in VLSI technology. In addition, Elliptic Curve Cryptography may allow more compact implementations. Cryptographic hardware accelerator modules are now a commodity for Virtual Private Networks (VPNs) and e-commerce transactions; they can even be found in smart card co-processors.

There is no doubt that in the coming years even better performance hardware implementations will be developed for high-end applications. We believe that the biggest challenge ahead may be the development of very compact and inexpensive low-power implementations that allow protection for personal and wireless devices and devices used in the context of ambient intelligence. A second challenge is to develop efficient implementations that offer adequate security against the ever more sophisticated side-channel attacks.

We have presented an efficient hardware implementation of the Montgomery modular multiplication over $GF\,(2^m)$ and $GF(p)$ in an FPGA. The designs have a systolic array architecture to allow pipelining and to make the clock frequency independent of the operand bit-length $m = s\omega$. In this way, the clock frequency

does not change when the bit-length is enlarged for security reasons. The clock frequency only depends on the word length ω, which determines the amount of logic in one cell of the systolic array. The word length is an input parameter for the implementation of the circuit. The MMM over $GF\,(2^m)$ is not restricted to field representations using irreducible trinomials: every irreducible polynomial of degree m can be used. In the design of MMM over $GF(p)$ the optimal bound is used which, with some savings in hardware, omits completely all the conditional reduction steps that are known to be vulnerable to the side-channel analysis attacks presented in 3.

We have described efficient implementations of elliptic curve processors over $GF(p)$ and $GF\,(2^m)$. The processors can be programmed to execute a modular multiplication, addition/subtraction, modular multiplicative inversion, EC point addition/doubling and multiplication.

Chapter 5

Measurement Setup

As part of a modern design flow, FPGAs are gaining more importance. Reasons for this include their relatively low cost and the available tools. Register transfer level descriptions (like VHDL for example) for a circuit can easily be ported, if not directly used, for an FPGA implementation of the circuit. Naturally, it is desirable to use the resulting FPGA implementation also for an evaluation of the designed circuit against power-analysis attacks.

In this chapter we describe the first realization of power-analysis attacks on a Virtex FPGA [201]. We can prove that this FPGA leaks a significant amount of information about its internal computations through the power supply lines. We can even provide evidence that the power consumption characteristics are comparable with the power consumption characteristics of ordinary application specific integrated circuits (ASICs).

In Section 5.1 we summarize a related work that is useful to understand the power consumption characteristics of FPGAs. In Section 5.2 we briefly introduce the FPGAs in general and in Section 5.3 the specific Xilinx Virtex FPGA of our setup. In order to use this FPGA for our experiments we have designed a board to measure the side-channel information while the FPGA is running. This board is introduced in Section 5.4. The experiments for characterizing the power consumption of our FPGA and the results produced with them are given in Section 5.5.

5.1 Related Work

The characterization of the power-consumption characteristics of FPGAs has received little attention so far. Shang *et al.* [237] is the only recent article in that field. In their article, Shang *et al.* analyze the dynamic power consumption of the XILINX Virtex-II family. They conclude that 60% of the dynamic power consumption is due to the interconnects, 14% is due to the clocking, 16% is due to the logic and 10% is due to the IOBs. As we explain in Section 3.3, in differential power analysis we exploit the correlation between the power consumption predictions and the measured power consumption for a set of inputs. We use the number of bit transitions in the registers as the power consumption prediction. Based on the result in [237] the part we predict is just 16% of the total power consumption of an FPGA. Hence it looks much more difficult to conduct power-analysis attacks on FPGAs than on ASICs. However, as we will demonstrate in this chapter, such attacks are feasible and can be realized in practise.

5.2 Field Programmable Gate Arrays

A field programmable gate array (FPGA) consists of an array of configurable logic blocks (CLBs), surrounded by programmable I/O blocks, and connected with programmable interconnections as shown in Fig. 5.1 [190]. A typical FPGA contains from 64 to tens of thousands of logic blocks and an even greater number of flip-flops. Most FPGAs do not provide a 100% interconnect between the logic blocks. Instead, sophisticated software places and routes the logic on the device.

Two main classes of FPGA architectures can be distinguished. Coarse-grained architectures consist of fairly large logic blocks, often containing two or more look-up tables and two or more flip-flops. Fine-grained architectures consist of a large number of relatively simple logic blocks. Another difference in the architectures is the underlying process technology used to manufacture the device. Currently, the highest-density FPGAs are built using SRAM technology, which is similar to microprocessors. The other common process technology is called anti-fuse which are one-time programmable.

SRAM-based devices are inherently re-programmable, even in-system. After a power-up is applied to the circuit, the program data defining the logic configuration must be loaded into the SRAM [148]. The program data defines how each of the logic blocks functions, which I/O blocks are inputs and outputs, and how the blocks are interconnected.

The FPGA either self-loads its configuration memory, or an external processor

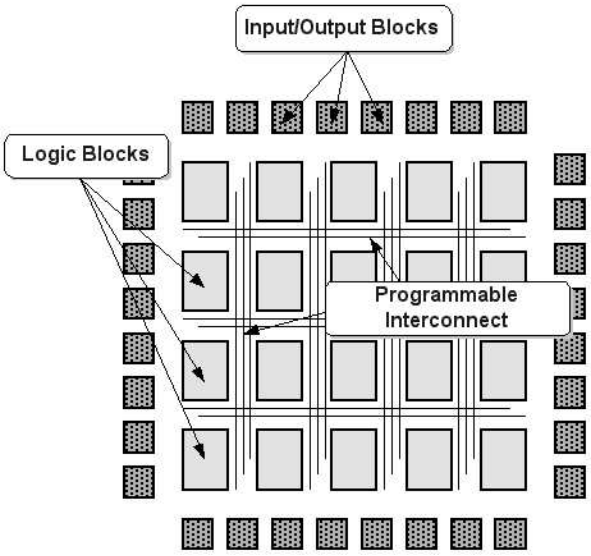


Figure 5.1: The architecture of the field programmable gate array (FPGA) used in the measurement setup

downloads the configuration file into the memory of the FPGA. The configuration time is typically less than 200 ms, depending on the device size and configuration method. In contrast, anti-fuse devices are one-time programmable (OTP). Once programmed, they cannot be modified, but they also retain their program when the power is off. Anti-fuse devices are programmed in a device programmer either by the end user or by the factory or distributor.

5.3 Xilinx Virtex Architecture

Virtex devices feature a flexible, regular architecture that comprises an array of CLBs surrounded by programmable input/output blocks (IOBs), all interconnected by a rich hierarchy of fast, versatile routing resources. Virtex FPGAs have a coarse-grained architecture, are SRAM-based, and are customized by loading configuration data into the internal memory cells.

5.3.1 Configurable Logic Block

The basic building block of the Virtex CLB is the logic cell (LC) [279]. A LC includes a 4-input function generator, carry logic, and a storage element. The output from the function generator in each LC drives both the CLB output

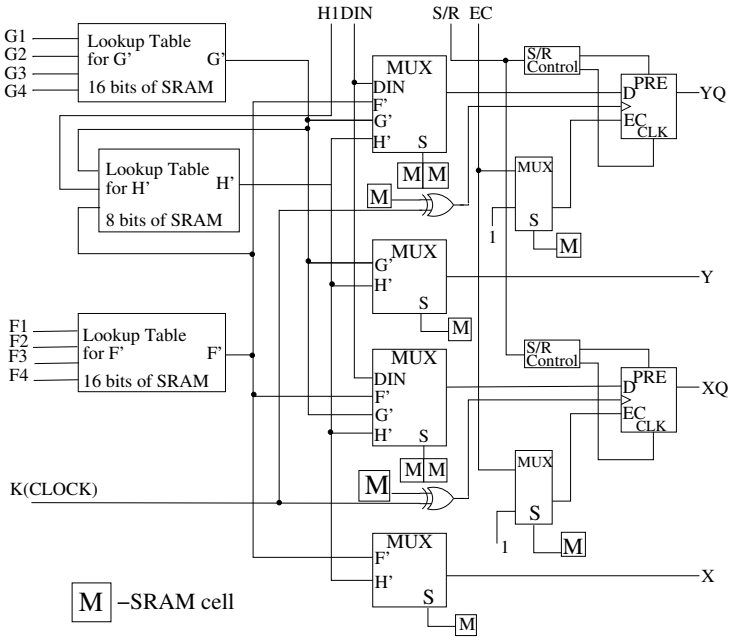


Figure 5.2: Simplified diagram of a slice in the Xilinx Virtex FPGA

and the D input of the flip-flop. Each Virtex CLB contains four LCs, organized in two similar slices. Figure 5.2 shows a more detailed view of a single slice. In addition to the four basic LCs, the Virtex CLB contains logic that combines function generators to provide functions of five or six inputs.

The Virtex function generators are implemented as 4-input LUTs. In addition to operating as a function generator, each LUT can provide a 16×1-bit synchronous RAM. The storage elements in the Virtex slice can be configured either as edge-triggered D-type flip-flops or as level-sensitive latches. The D inputs can be driven either by the function generators within the slice or directly from the slice inputs, bypassing the function generators. In addition to Clock and Clock Enable signals, each Slice has synchronous set and reset signals (SR and BY). All the control signals can be inverted independently and are shared by the two flip-flops within the slice.

5.3.2 I/O Block

The Virtex IOB features SelectIO inputs and outputs that support a wide variety of I/O signalling standards [279]. The three IOB storage elements function either as edge-triggered D-type flip-flops or as level sensitive latches.

Prior to configuration, all pins not involved in configuration are forced into their high-impedance state.

5.3.3 I/O Banking

Some of the I/O standards require VCCO and/or VREF voltages. These voltages are connected to the device pins that serve groups of IOBs, called banks. Consequently, not all I/O standards can be combined within a given bank. Each bank has multiple VCCO (Output supply voltage) pins, all of which must be connected to the same voltage. This voltage is determined by the output standards in use.

5.3.4 Configuration of the FPGA

Virtex devices are configured by loading configuration data into the internal configuration memory. Some of the pins used for this are dedicated configuration pins, while others can be re-used as general purpose inputs and outputs once configuration is complete. Virtex supports four configuration modes which are the Slave-serial mode, the Master-serial mode, the SelectMAP mode and the Boundary-scan mode. The configuration mode pins (M2, M1, and M0) define which of these modes is used. Our board supports three of these configuration modes.

5.4 Measurement Setup

Our setup consists of two boards (see Fig. 5.3), an oscilloscope, current probe, a passive probe, a power supply and a function generator. The main board is responsible for interfacing to the PC via the parallel port. It is connected with the XILINX parallel cable in order to program the VIRTEX FPGA and it provides some LEDs, switches and buttons for testing purposes. The daughter board itself just carries the VIRTEX FPGA; it allows to access some pins for triggering and to measure the power consumption of the VIRTEX FPGA in a convenient way.

5.4.1 Mother Board

The Parallel Port [15] is the most commonly used port for interfacing home made projects. This port allows the input of 5 bits and the output of 12 bits. The port is composed of 4 control lines, 5 status lines and 8 data lines. The communication between the FPGA and the PC uses this parallel port. We need only 17 input/output pins to send data or commands to the FPGA and

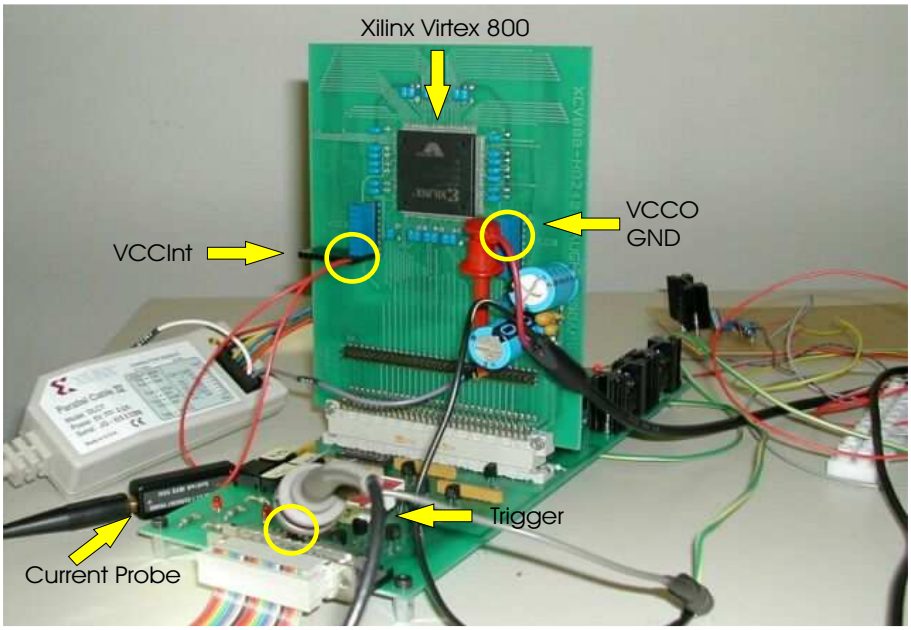


Figure 5.3: The measurement setup. On the daughter board the current probe is connected to VCCINT. Alternatively it can be connected to the VCCO of the individual banks, or the GND.

receive the result, but we designed the board in such a way that it gives us more monitoring points and thus connected 32 input/output pins of the FPGA to the board. We use the additional 15 input/output pins for different reasons during different stages of the development. Some of them are connected to some LEDs and buttons. So during the first tests of the board these were used to apply some simple inputs and to receive some outputs without the need for the parallel port communication. During the circuit development stage, we design the circuits in a way that they send some signals showing the internal states to these pins. Hence they give us extra control points for the correct functionality of the circuits. During the measurement phase we use them as trigger signals, which show us starting or ending of the events that we are interested in. The unused input/output pins are pulled up after the configuration.

We have also designed a protocol to send and receive data to and from the FPGA. When the FPGA communicates with the PC, it uses the three most significant bits of the status lines to indicate its status. The two remaining bits of the status lines are used for sending the result from the FPGA to the PC. The protocol is independent from the operation executed in the FPGA. Only the length of the data which is communicated can be modified by the PC. This provides a flexible setup where experiments with different algorithms can be performed in a coherent manner.

5.4.2 Daughter Board

We use a Xilinx XCV800 FPGA from the Virtex series in a HQ240C package. Reasons for this particular choice include:

1. The resources are sufficient to implement a 160-bit elliptic-curve point-multiplication.
2. This is the most powerful FPGA that can be used for hand-mounting on the board. This is because the pins of this FPGA are on its sides. The more powerful FPGAs have the pins underneath with a grid structure and so special machines are needed to mount them.
3. The architecture is made of combinational and memory elements. Because of this property it is a good representative of application specific integrated circuits (ASICs).

The XCV800 has 12 core voltage supply (VCCINT) pins, 16 output voltage supply (VCCO) pins and 32 ground (GND) pins. The FPGA is divided into 8 banks each with their own VCCINT and VCCO pins. After the implementation of the desired circuit and the configuration of the FPGA with the implementation data, the banks whose CLBs are used more than the others should draw more current from their supply lines. The CLBs that are not used do not

draw current from their banks supply lines. With our setup it is possible to verify this hypothesis. If different parts of a design (such as an elliptic-curve addition and an elliptic-curve doubling) are mapped to different banks of the FPGA, measuring the current of the individual banks allows us to take more precise measurements. By measuring VCCINT and VCCO of the same bank separately, we can detect the input/output and core activity timing and power consumption separately.

Therefore we use three headers with two lines for VCCINT, VCCO and GND as shown in Fig. 5.3. During the normal operation of the board without measurement, the two pins are connected by a jumper. When we want to measure the current flow from a specific bank, the associated jumper is replaced by a cable that is going through the hole in the current probe as shown in Fig. 5.3.

This setup creates the possibility of making measurements on different points at the same time and makes it easy to modify the measurement point.

5.4.2.1 Bypassing Considerations

With high-speed, high-density FPGA devices, maintaining signal integrity is the key to reliable, repeatable designs [280]. Proper power bypassing and decoupling improves the overall signal integrity. Without it, power and ground voltages are affected by logic transitions and can cause operational issues.

When a logic device switches from a logic one to a logic zero, or a logic zero to a logic one, the output structure is momentarily at a low impedance across the power supply. Each transition requires that a signal line be charged or discharged, which requires energy. As a result, many electrons are suddenly needed to keep the voltage from collapsing. The function of the bypass capacitor is to provide local energy storage.

10 nF capacitors are placed between every VCCINT and VCCO pin of the FPGA and the nearest GND. Because we designed the setup in two different cards, the daughter card can be thought of as a stand-alone chip taking power from the mother board. Bypass capacitors had to be placed between the power supplies of the card and the GND line.

5.4.3 Measurement Equipment

The measurement equipment consists of a Tektronix TDS714L digital storage oscilloscope (DSO), a Tektronix CT1 current probe, a passive probes, a power supply and a function generator.

Tektronix TDS714L oscilloscope has 500 MHz bandwidth, 500 MS/s sample rate, 8-bit vertical resolution, 1 mV/div to 10 V/div sensitivity, record lengths to 8 M points. The full record length cannot be used with the mathematical

functions. Because of this we measure the data by the oscilloscope, transfer it to the PC via GPIB connection and then perform the needed operations on the data with MATLAB. For each measurement the PC sends the inputs and then the signal to start the execution of the operation to the FPGA. This signal is used as a trigger to start the measurements by the oscilloscope. After the execution of the operation on the FPGA is completed and simultaneously the measurement is done, the PC receives the outputs from the FPGA, compares them with the ones calculated by the C program. If the results from the C program and the FPGA are the same then the PC takes the measurement data from the oscilloscope.

Tektronix CT1 current probe is used to measure the dynamic power consumption of the FPGA. The probe is a closed-circuit design which will accept isolated wire sized up to 20 gauge. The probe converts the current flows on this wire to voltage and we measure this voltage as a representative of the power consumption of the circuits.

We use a function generator instead of a fixed frequency oscillator on the FPGA board. This gives us the possibility to change the clock frequency as needed. Hence we can observe different power consumption characteristics of the circuit for different clock frequencies.

The power supply provide 8 V DC to the main board. This voltage level is divided to get 3.3 V for the I/O and 2.5 V for the core power supply of the FPGA.

5.5 Power Consumption Characteristics

We now describe the experiments conducted on the measurement setup described above. As the aim of our work was to build an alternative platform for power-analysis attacks, we decided to perform first some basic experiments to verify that the assumption which we usually make for such attacks are also valid for our FPGA setup.

As discussed in Section 3.3, we should be able to detect either transition-count leakage or Hamming weight leakage in our setup. This is because according to Section 5.2, the CLBs consist of flip-flops (and other logic) which exhibit the power consumption characteristics of CMOS technology. The only problem can be that if the circuit, which we load into the FPGA, does not use all of the FPGAs resources, then the noise which is produced by the unused parts might be larger than the signal produced by the circuit.

As we explained in Section 3.3, transition count information leaks when the dominant source of the current is due to the switching of the gates and Ham-

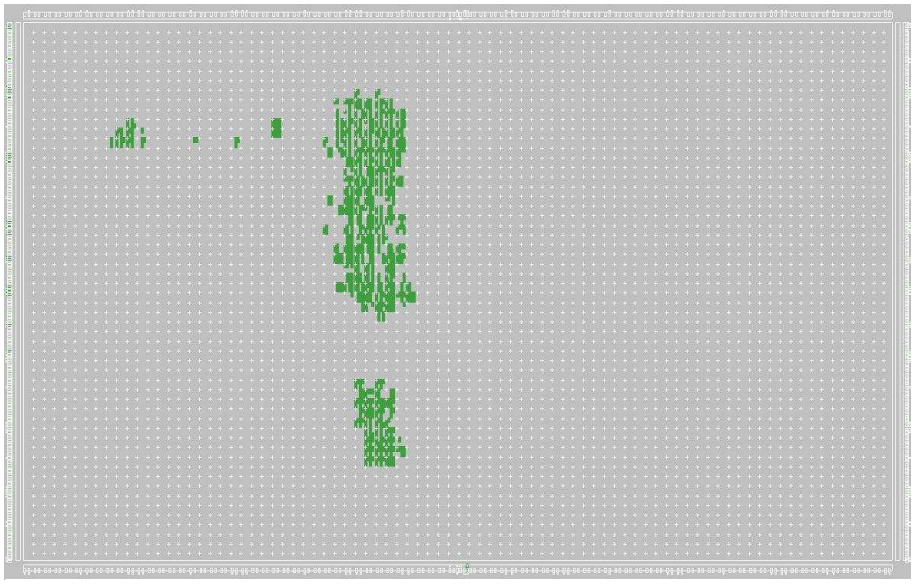


Figure 5.4: Floor plan of the 8-bit elliptic curve point addition circuit as a result of the implementation on the FPGA

ming weight information leaks when a pre-charged bus design is used. We do not use pre-charged bus design. Hence, we expect that our setup leaks transition-count information.

In order to evaluate the behavior of the FPGA we implemented an 8-bit EC point addition circuit on the FPGA. The floor plan of the FPGA as the result of this implementation is shown in Fig. 5.4. The floor plan shows the CLBs of the FPGA. The dark areas on the floor plan are the used CLBs after the implementation of 8-bit EC point addition circuit on the FPGA. We use only 3% of the FPGA and of the FPGA's flip-flops.

Next, we measured the power consumption of the whole FPGA and, at the same time, the power consumption of an empty (idle) bank and partially full bank (see Fig. 5.5, Fig. 5.6 and Fig. 5.7 for the power consumption traces). The overall power consumption shows clearly higher peaks than both of the other traces. The idle bank's power consumption trace however does not exhibit any peaks during the whole computation. This experiment confirms that idle parts of the FPGA will not influence the overall power consumption. Moreover, even the power consumption of a very small circuit (we used only 3% of the FPGA and of the FPGAs flip-flops) can be easily detected.

With another simple set of experiments we confirmed that the amount of power

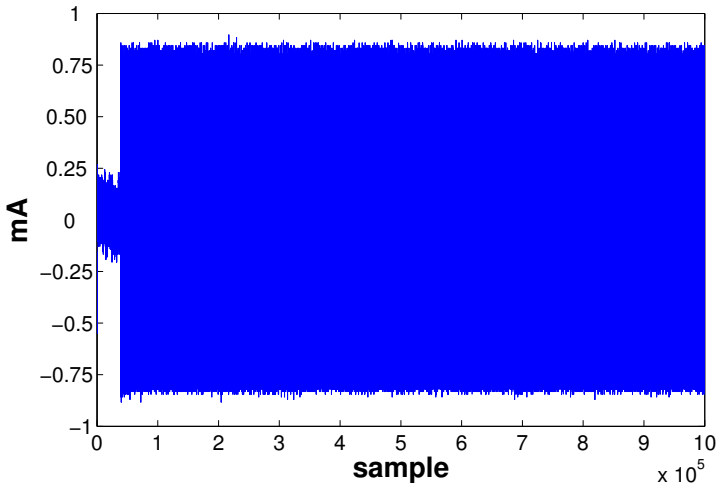


Figure 5.5: Current consumption trace measured from the total VCCINT of the FPGA for the 8-bit elliptic curve point addition

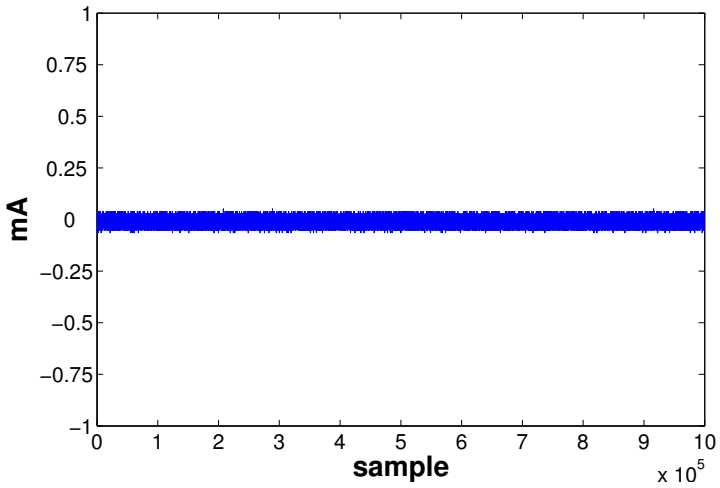


Figure 5.6: Current consumption trace measured from the VCCINT of an empty bank of the FPGA for the 8-bit elliptic curve point addition

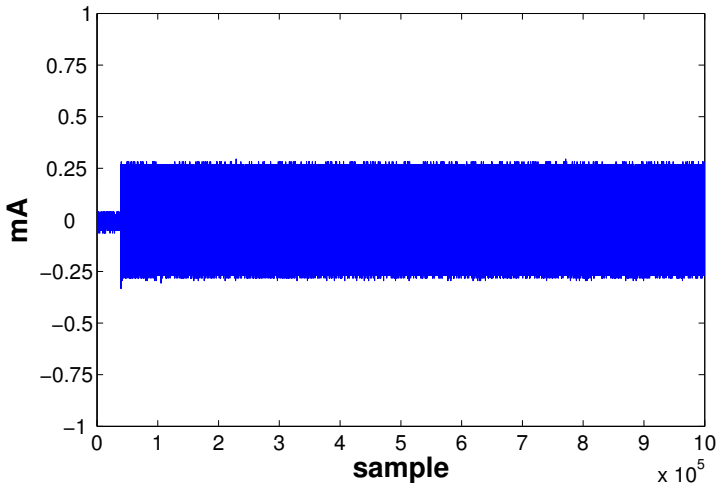


Figure 5.7: Current consumption trace measured from the VCCINT of a partially full bank of the FPGA for the 8-bit elliptic curve point addition

consumed of the FPGA is linear in the number of flip-flops that change their outputs. We have designed registers of a specific size and loaded them on the FPGA. Then we let them repeatedly store 0 and 1 values and measured the FPGA's power consumption. Figure 5.8 and 5.9 illustrate that the power consumed by the 6000-bit register for flipping from all 0s to all 1s is about twice as high as the power consumed by the 3000-bit register.

A direct conclusion from such experiments is that the power consumption characteristics are essentially the same as of an ordinary CMOS circuit. Idle CLBs or even idle banks do not add too much noise to the overall power consumption.

5.6 Conclusions

We have introduced a new platform for evaluating power analysis. Our approach consists of an FPGA, which is placed on a hand-made board which makes it very easy to conduct power-analysis attacks. We have characterized the power consumption of a XILINX Virtex 800 FPGA and conclude that it is similar to the power consumption of an ordinary ASIC in CMOS technology. Therefore, it is possible to draw conclusions about the vulnerability of a certain circuit by performing power-analysis attacks on an FPGA-implementation. Since programming an FPGA is considerably less expensive than manufacturing an ASIC, assessing vulnerability of a device to power analysis attacks is

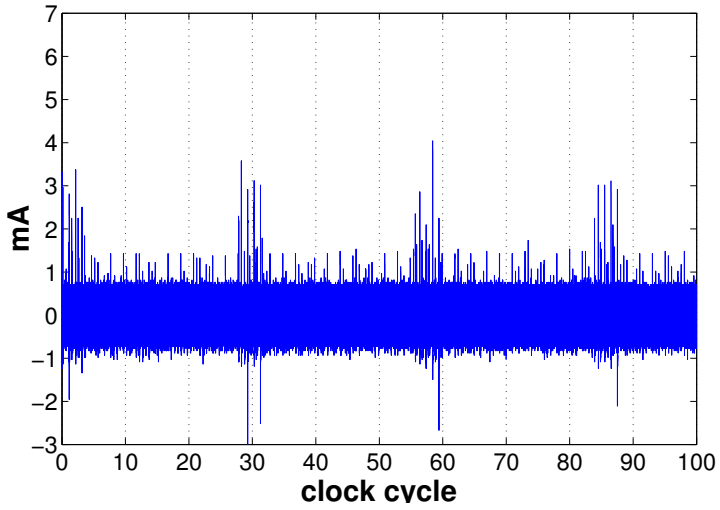


Figure 5.8: Current consumption trace of a 3000-bit register during some transitions on its outputs

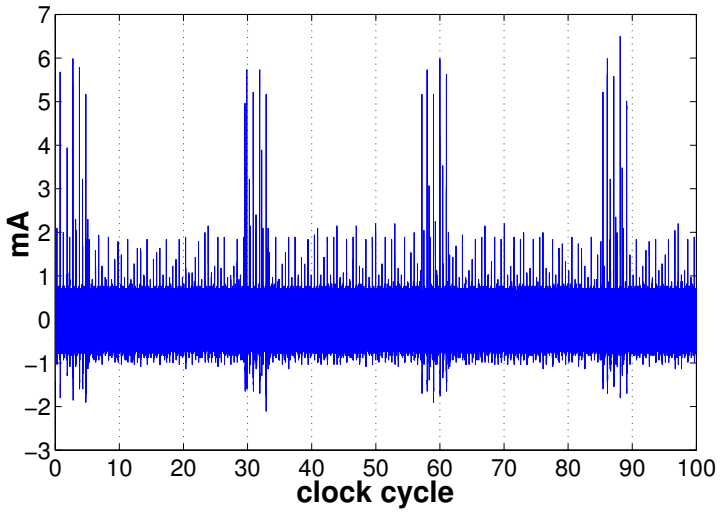


Figure 5.9: Current consumption trace of a 6000-bit register during some transitions on its outputs

much cheaper on our platform. Consequently, our approach describes the first inexpensive and efficient way to conduct power-analysis attacks on a real implementation (*i.e.*, not on a software simulation) of a circuit in a very early stage of the design.

Chapter 6

Side Channel Analysis of FPGA Implementations of Elliptic Curve Cryptosystems

As explained in Chapter 3 side-channel analysis attacks form important threats against hardware implementations of cryptographic algorithms. Hence it is natural to study the weaknesses of our circuits, presented in Chapter 4, against these attacks. We have implemented SCA attacks on our FPGA implementations of ECCs and we have improved the resistance of the circuits against side channel attacks.

In Section 6.1 we show a timing analysis attack against an FPGA Implementation of ECC over $GF(p)$. Our initial design was vulnerable and it was possible to learn the Hamming weight of the key by using the timing information. Then we improved our circuit and give the TA attack results on it in Section 6.1.1. It is not possible to find out any information about the key by a TA attack on the improved design. In Section 6.2 we give the results of the simple power analysis attack on the same initial circuit. As it is expected the SPA attack given in Section 6.2.1 was successful and we could learn all the bits of the key by using only one measurement. After improving our initial design, our elliptic curve processor became resistant against a SPA attack. We implemented a DPA attack on the improved implementation mentioned above in Section 6.2.2 and showed that it is possible to find the value of the MSB of the key. By using this information of the previous key bit we can implement the DPA attack targeting the next MSB. Hence, the DPA attack presented in this chapter reveals

all the bits of the key. Also we present simple and differential electromagnetic analysis attacks on the same circuits in Sections 6.3.1 and 6.3.2. The SEMA attack is as successful as the SPA attack. In order to succeed with the DEMA attack we need more measurements for the DEMA attack than the DPA attack, because the noise level of the electromagnetic radiation measurements is higher than that of the power consumption measurements.

6.1 Timing Analysis of a FPGA Implementation of Elliptic Curve Cryptosystem over $GF(p)$

The clock frequency we use during these analysis is 5 MHz, hence one clock period is 200 ns. According to the timings given in Section 4.2.2.3 and Section 4.2.1, one 160-bit MMM will take 485×200 ns $= 97$ μ s and one 160-bit MAS will take 321×200 ns $= 64.2$ μ s.

As explained in Section 2.1.4, an elliptic curve point multiplication consists of point additions and doublings. The latency of one EC point addition and doubling can be calculated as $T_{PAD} = 14T_{MMM} = 1.358$ ms and $T_{PDB} = 8T_{MMM} + 6T_{MAS} = 1.161$ ms as shown in Section 4.5.6, respectively.

If we use Algorithm 2.1 for a 160-bit elliptic curve point multiplication with an l-bit key with the MSB of the key equals to 1, the latency of one point multiplication will be

$$T_{PMUL} = (l-1)T_{PDB} + (w-1)T_{PAD} = (l1.161 + w1.358 - 2.519)\ \text{ms}\,,$$

with w the Hamming weight of the binary representation of the key [160]. It means that somebody who can measure the execution time of one 160-bit elliptic curve point multiplication will learn the Hamming weight of the key by using the above expression. Hence Algorithm 2.1 is vulnerable to TA attacks.

6.1.1 Countermeasures Against Timing Attacks

As explained in Section 6.1, Algorithm 2.1 is vulnerable to timing analysis attack because of the conditional branch at Step 4. In order to get rid of this weakness we use the algorithm proposed by Coron in [44]: we execute always a point doubling and a point addition independent from the value of the current key bit. After finishing both point operations, we select the needed result according to the value of the current key bit as shown in Algorithm 6.1.

If we use Algorithm 6.1 for a 160-bit elliptic curve point multiplication with an l-bit key with the MSB of the key equals to 1, then the latency of one

Algorithm 6.1: Elliptic curve point multiplication: binary method, left to right, always double and add

Require: EC point $P = (x, y)$, integer k, $0 < k < M$, $k = (k_{l-1}, k_{l-2}, \cdots, k_0)_2$, $k_{l-1} = 1$ and M
Ensure: $Q = [k]P = (x', y')$
1: $Q \leftarrow P$
2: **for** i from $l-2$ downto 0 **do**
3: $Q_1 \leftarrow 2Q$
4: $Q_2 \leftarrow Q_1 + P$
5: **if** $k_i = 0$ **then**
6: $Q \leftarrow Q_1$
7: **else**
8: $Q \leftarrow Q_2$
9: **end if**
10: **end for**

point multiplication will be $T_{PMUL} = (l-1)\left(T_{PDB} + T_{PAD} + 200 \cdot 10^{-9}\right) = (l-1)2.519$ ms. This latency depends only on the key bit-length l, but not on the Hamming weight of the key.

As a second countermeasure against timing analysis attack, we use the binary right to left point multiplication algorithm by executing point addition and doubling in parallel as proposed by Izu and Takagi in [112, 113].

Algorithm 6.2: Elliptic curve point multiplication: binary method, right to left, parallel double and add

Require: EC point $P = (x, y)$, integer k, $0 < k < M$, $k = (k_{l-1}, k_{l-2}, \cdots, k_0)_2$, and M
Ensure: $Q = [k]P = (x', y')$
1: $Q \leftarrow \mathcal{O}$, $S \leftarrow P$
2: **for** i from 0 to $l-1$ **do**
3: $Q_2 \leftarrow Q + S \parallel S \leftarrow 2S$
4: **if** $k_i = 1$ **then**
5: $Q \leftarrow Q_2$
6: **else**
7: $Q \leftarrow Q$
8: **end if**
9: **end for**

If we use Algorithm 6.2 for the elliptic curve point multiplication the inputs of the point addition in Step 3 can be any point. This is different from the case explained in Section 4.5.6. In Section 4.5.6, Algorithm 4.11 calculates a

simplified EC point addition by using the fact that one of the input points to the point addition in Algorithm 2.1 is always the input point P and its Z coordinate in projective coordinates is always 1. The non-simplified point addition algorithm is described below in Algorithm 6.3. Algorithm 6.3 computes Eq. (2.4).

Algorithm 6.3: Non-simplified elliptic curve point addition over $GF(p)$

Require: $P_1 = (X_1, Y_1, Z_1, aZ_1^4)$, $P_2 = (X_2, Y_2, Z_2, aZ_2^4)$
Ensure: $P_1 + P_2 = P_3 = (X_3, Y_3, Z_3, aZ_3^4)$

1. $T_1 \leftarrow Z_2^2$
2. $T_2 \leftarrow X_1 T_1$
3. $T_3 \leftarrow Z_1 Z_1$
4. $T_4 \leftarrow X_2 T_3$
5. $T_1 \leftarrow X_1 T_1 \quad T_4 \leftarrow T_4 - T_2$
6. $T_1 \leftarrow Z_2 T_1$
7. $T_3 \leftarrow Z_1 T_3$
8. $T_3 \leftarrow Y_2 T_3$
9. $T_5 \leftarrow Z_2 T_4 \quad T_3 \leftarrow T_3 - T_1$
10. $Z_3 \leftarrow Z_1 T_5$
11. $T_5 \leftarrow T_4 T_4$
12. $T_2 \leftarrow T_2 T_5$
13. $T_5 \leftarrow T_4 T_5 \quad T_4 \leftarrow 2T_2$
14. $T_6 \leftarrow T_3 T_3 \quad T_4 \leftarrow T_5 + T_4$
15. $T_5 \leftarrow T_1 T_5 \quad X_3 \leftarrow T_6 - T_4$
16. $T_6 \leftarrow Z_3^2 \quad T_2 \leftarrow T_2 - X_3$
17. $T_2 \leftarrow T_3 T_2$
18. $T_6 \leftarrow T_6 T_6 \quad Y_3 \leftarrow T_2 - T_5$
19. $aZ_3^4 \leftarrow aT_6$

Nineteen states and six temporary registers are needed for EC point addition. The total execution time of EC point addition is $19T_{MMM}$. The latency of one 160-bit point addition can be calculated as, $T_{PAD} = 19T_{MMM} = 1.843$ ms.

One point doubling takes less time than one point addition, thus in Algorithm 6.2 for elliptic curve point multiplication the latency of Step 3 is the latency of one point addition. The execution time of one 160-bit elliptic curve point multiplication with an l-bit key with the MSB of the key equals to 1 is $T_{PMUL} = (l-1)\left(T_{PAD} + 200 \cdot 10^{-9}\right) = (l-1)1.843$ ms. This latency depends only on the key bit-length l, but not on the Hamming weight of the key.

6.2 Power Analysis of a FPGA Implementations of Elliptic Curve Cryptosystem over $GF(p)$

In this section, we conduct power analysis attacks on the FPGA implementation of our elliptic curve processor over $GF(p)$ presented in Section 4.5. First, we demonstrate that our initial design is vulnerable to simple power analysis attack and show that it is possible to find out the key with only one measurement. Then we improve our design in order to be resistant against these attacks. Our improved design is vulnerable to differential power analysis attack as presented in the following sections.

6.2.1 Simple Power Analysis of a FPGA Implementation of Elliptic Curve Cryptosystem over $GF(p)$

The hardware design of the elliptic curve processor over $GF(p)$ is explained in Section 4.5. Because we use a Montgomery modular multiplier circuit as a basic operational block in the hardware design of our elliptic curve processor over $GF(p)$, we first show the current consumption trace of this block. Then we analyze the power consumption of the EC point addition, doubling and point multiplication blocks presented in Section 4.5.

The measurement of a 480-bit MMMC is depicted in Fig. 6.1. The three parts shown in the figure can be explained according to the algorithm and the architecture used. The T register in Fig. 4.19 is reset at the start of the MMM operation and then it is being updated for 1440 clock cycles. The number of bits in T which are updated is increasing until clock cycle 480. This stage corresponds to the first part shown in the current consumption trace. After 480 clock cycles all the bits of the T register have been assigned a value and all of them are updated before clock cycle 960. This stage is visible in the second part of Fig. 6.1. The last part of Fig. 6.1 corresponds to reading out the result from the pipeline. Because there is no new input on the LSB of the systolic array, starting from clock cycle 961 the number of bits of the T register that is updated decreases.

The three parts of Fig. 6.1 prove one more time that our circuit leaks transition count information. The experiment on the MMMC also teaches us the shape of the current consumption trace of one of the building blocks of our elliptic curve processor. During the later experiments we recognize this pattern easily in order to implement SPA. In the DPA attack this pattern shows us the right time to make the measurements.

The circuits for EC point addition and EC point doubling are described in Section 4.5.6. The current consumption trace of one 160-bit EC point addition implemented with Algorithm 4.11 is shown in Fig. 6.2. Fourteen states can be

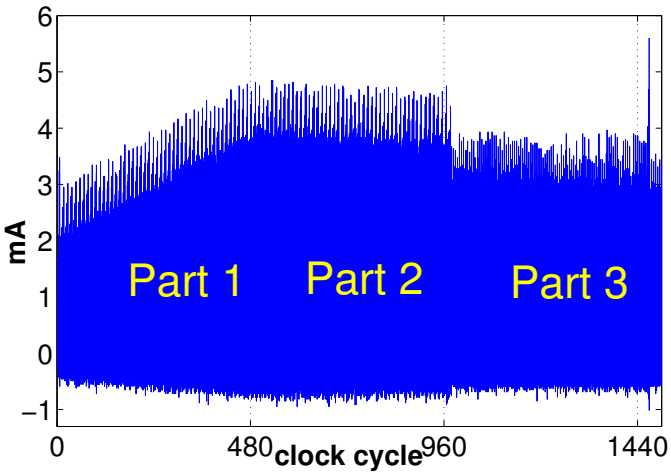


Figure 6.1: Current consumption trace of a 480-bit Montgomery modular multiplier over $GF(p)$

counted easily from the trace. All the states are completed in nearly 500 clock cycles which corresponds $T_{MMM} = 3 \cdot 160 + 4$. The power consumption during Step 3, 5, 7, 8, 10, 11 and 13 seems higher than for the other steps. In these steps a modular addition or subtraction is taking place as well as an MMM.

The current consumption trace of one 160-bit EC point doubling is shown in Fig. 6.3. As expected, the number of clock cycles for EC point doubling is less than the number of clock cycles for EC point addition. The main difference in power consumption between EC point addition and EC point doubling can be observed by looking at Step 7, 8, 10, 11, 12 and 14. In these steps only a modular addition or subtraction takes place. Obviously the latency and power consumption of these are smaller than the others. This means that a simple power analysis attack is easy to perform.

The EC point multiplication is implemented with the simple double-and-add algorithm given by Algorithm 2.1. The current consumption trace of a 160-bit EC point multiplication is shown in Fig. 6.4. It can be easily seen from Fig. 6.4 that the key used during this measurement is 1001100.

We have changed our design to work with Algorithm 6.1 as a countermeasure for the attack given above. The current consumption trace of one EC point multiplication is shown in Fig. 6.5. It follows from Fig. 6.5 that it is no longer possible to attack this circuit by SPA. So we will use a DPA attack to find the key bits in the following section.

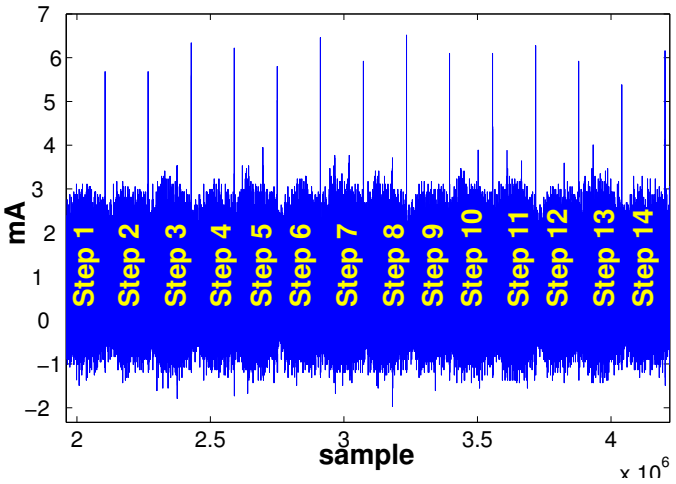


Figure 6.2: Current consumption trace of a 160-bit elliptic curve point addition over $GF(p)$

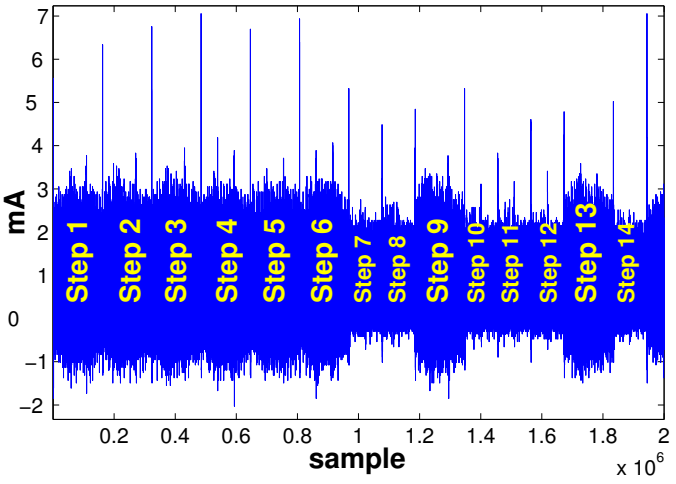


Figure 6.3: Current consumption trace of a 160-bit elliptic curve point doubling over $GF(p)$

6.2.2 Differential Power Analysis of a FPGA Implementation of Elliptic Curve Cryptosystem over $GF(p)$

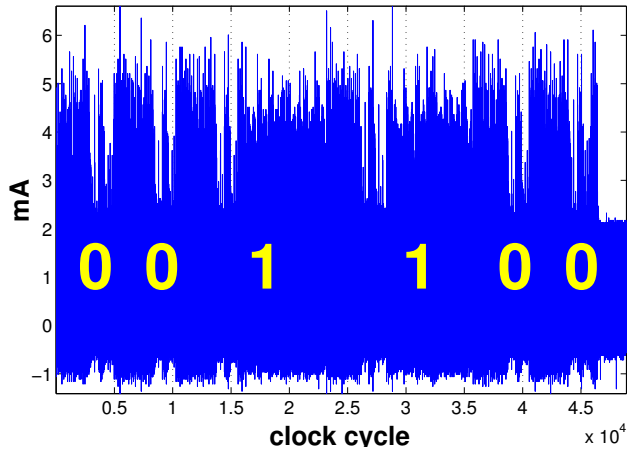


Figure 6.4: Current consumption trace of a 160-bit elliptic curve point multiplication over $GF(p)$ with Algorithm 2.1

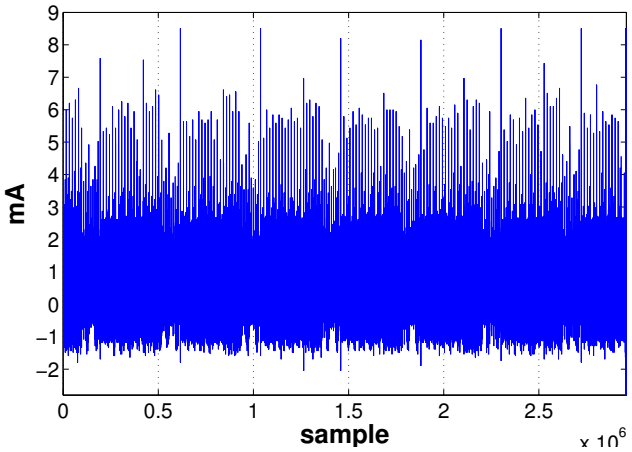


Figure 6.5: Current consumption trace of a 160-bit elliptic curve point multiplication over $GF(p)$ with Algorithm 6.1

The target for our DPA attack is the second MSB of the key, k_{l-2}, in Algorithm 6.1. There are two temporary point registers in the design, Q_1 and Q_2. These temporary points and the output point Q are updated in the following order.

Step 1: $Q \leftarrow P$
Step 3: $Q_1 \leftarrow 2Q = 2P$
Step 4: $Q_2 \leftarrow Q_1 + P = 3P$

Step 5: $$Q \leftarrow \begin{cases} Q_1 = 2P & k_{l-2} = 0 & \text{Step 6} \\ Q_2 = 3P & k_{l-2} = 1 & \text{Step 8} \end{cases}$$

Step 3: $$Q_1 \leftarrow \begin{cases} 2Q = 4P & k_{l-2} = 0 \\ 2Q = 6P & k_{l-2} = 1 \end{cases}$$

Step 4: $$Q_2 \leftarrow \begin{cases} Q_1 + P = 5P & k_{l-2} = 0 \\ Q_1 + P = 7P & k_{l-2} = 1 \end{cases}$$

Step 5: $$Q \leftarrow \begin{cases} 2Q = 4P & k_{l-2} = 0 & k_{l-3} = 0 & \text{Step 6} \\ 2Q = 6P & k_{l-2} = 1 & k_{l-3} = 0 & \text{Step 6} \\ 2Q = 5P & k_{l-2} = 0 & k_{l-3} = 1 & \text{Step 8} \\ 2Q = 7P & k_{l-2} = 1 & k_{l-3} = 1 & \text{Step 8} \end{cases}$$

The first step of the DPA attack is to find the point of the measurements. The current consumption trace of an EC point multiplication is shown in Fig. 6.5. The highest seven spikes on Fig. 6.5 show the end of seven EC point doubling operations. Because these spikes are clearly higher than other spikes on the current consumption trace, our attack point is one of these seven spikes. The first one corresponds to the end of the first EC doubling operation. As shown above this spike shows the ending of the second operation which is $Q_1 \leftarrow 2P$ and this step is executed independent from the key bits. The third, fourth and so an spikes need the knowledge of the k_{l-2}, k_{l-3} etc. Hence our choice for the measurement point is the second update of Q_1 after the second EC point doubling (Step 3). We use the transitions between the previous value of Q_1, $2P$, and the new value at our target point, $4P$ or $6P$ according to the value of k_{l-2} as the power consumption predictions.

6.2.2.1 Correlation Analysis

In the first step of our attack, we have produced a power consumption file. For this purpose, we have chosen N random points on the EC and one fixed, but random key, k. The FPGA executes N point multiplications such that $Q_i = [k]P_i$ for $i = 1, 2, \cdots, N$. We have measured the power consumption of the FPGA during 2400 clock cycles around the second update of Q_1. The clock frequency applied to the chip was around 300 kHz and the sampling frequency of the oscilloscope was 250 MHz. With these measurements, we have produced a $N \times 2\,000\,000$ matrix, M_1. The current consumption trace of one of these measurements, is shown in Fig. 6.6.

We have applied a pre-processing technique to reduce the amount of measurement data in every clock cycle. We have found the maximum value of the measurement data in each clock cycle as follows:

$$M_2(i,j) = max(M_1(i, D_i \cdot (j-1) + 1 : D_i \cdot j)), \tag{6.1}$$

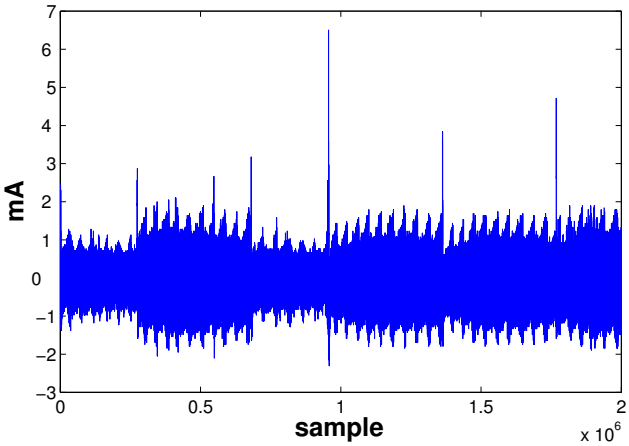


Figure 6.6: Current consumption trace of the FPGA during the execution of one EC point multiplication for DPA attack (1st row of M_1)

where $i = 1, \ldots, N$, $j = 1, \ldots, 2400$. $M_2(i, j)$ is the element of the matrix M_2 at the ith row and the jth column. D_i is the number of samples per clock cycle during ith measurement, $M_1(i, D_i \cdot (j-1) + 1 : D_i \cdot j)$ is the row vector $[M_1(i, D_i \cdot (j-1) + 1) \;\; M_1(i, D_i \cdot (j-1) + 2) \;\; \cdots \;\; M_1(i, D_i \cdot j)]$.

Because the clock frequency of the function generator in our experiments was varying slightly during the measurements we have to find D_i. In order to compute D_i we have to know the exact clock frequency. We have calculated the DFT of ith measurement, corresponding the ith row of M_1. As the clock frequency can be between 250 kHz and 375 kHz, we have searched between these frequencies for the maximum value in the DFT trace. The result for the first measurement is shown in Fig. 6.7. According to this figure the clock frequency during the first measurement was 302.8 kHz. Hence, $D_1 = 250{\cdot}10^6/302.8{\cdot}10^3 = 825.63$. Figure 6.8 shows the first measurement after taking the maximum value in every clock cycle.

We have implemented the EC point multiplication with Algorithm 6.1 in the C programming language. The C program computes N EC point multiplications with N EC points and the key. The EC points and the key are the same as the ones given to the FPGA during the measurements. During the execution of the EC point multiplications, the C program computes the number of bits that change from 0 to 1 in some registers at the steps corresponding to the five spikes shown in Fig. 6.8. The number of transitions is used as the power consumption prediction.

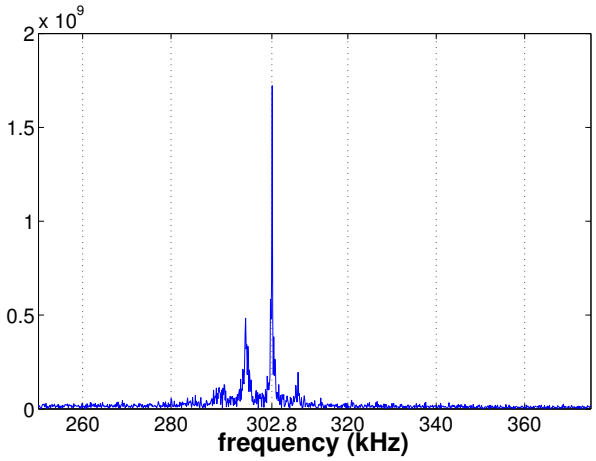


Figure 6.7: DFT of the current consumption trace of the first measurement between 250 kHz and 375 kHz; the clock frequency is 302.8 kHz

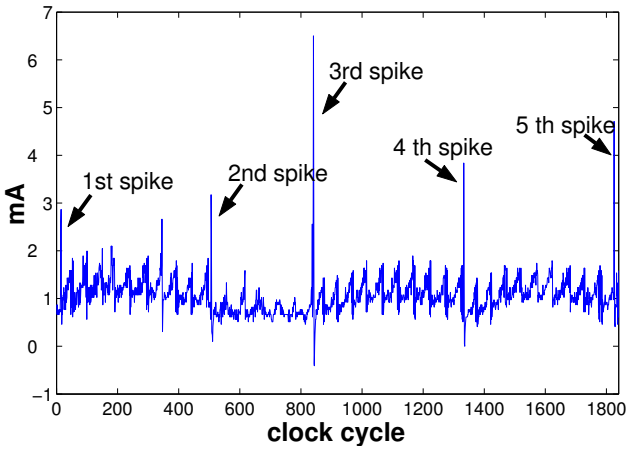


Figure 6.8: Current consumption trace of the first measurement after taking the maximum value in every clock cycle (1st row of M_2)

One of the steps of the attack is to find the right steps in the C program to predict the power consumption which corresponds to the measurement points. As we aimed to measure the power consumption of the FPGA around the last clock cycle of the second EC doubling in Step 3 of Algorithm 6.1, the third spike in Fig. 6.8 corresponds to this event. Hence we can predict it by counting

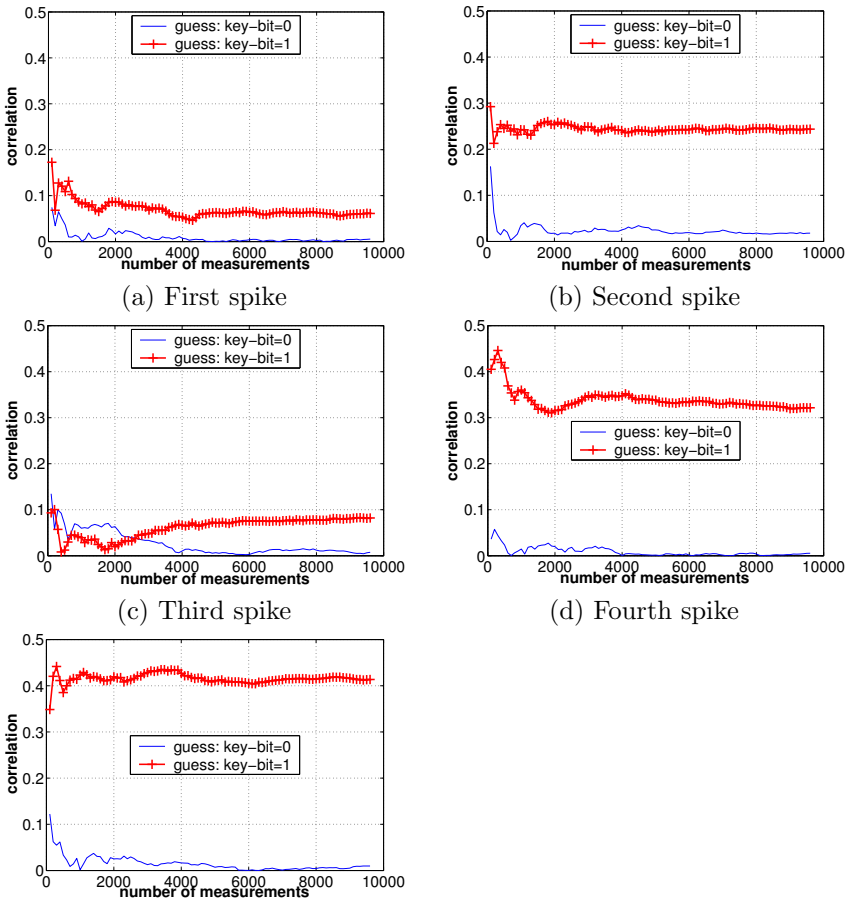
the number of transitions between the bits of the coordinates of Q_1 and $2Q$; for $k_{l-2} = 0$ we count the number of transitions between $2P$ and $4P$ and for $k_{l-2} = 1$ between $2P$ and $6P$.

The first and the second spikes in Fig. 6.8 occur at the end of the twelfth and the thirteenth steps of the EC point doubling with Algorithm 4.12, respectively. The fourth and the fifth spikes in Fig. 6.8 occur at the end of the first and the second steps of the EC point addition with Algorithm 4.11, respectively. For $k_{l-2} = 0$ the second EC doubling is executed with the input $2P$ and the second EC addition is executed with the inputs $4P$ and P. For $k_{l-2} = 1$ the second EC doubling is executed with the input $3P$ and the second EC addition is executed with the inputs $6P$ and P. We have produced two power consumption prediction matrixes, M_3 and M_4, which are for the $k_{l-2} = 0$ and $k_{l-2} = 1$ guesses, respectively. M_3 and M_4 have five columns for the five spikes and N rows for the N EC points.

Now we can learn the right value of k_{l-2} by finding the correlations between the columns of M_3 and M_4 and the five columns of M_2 which correspond to the five spikes in Fig. 6.8. If the correlations between the columns of M_3 and M_2 are higher than the correlations between the columns of M_4 and M_2, then we decide that $k_{l-2} = 0$, otherwise we decide that $k_{l-2} = 0$. We are also interested in finding the minimum number of measurements that are necessary to find the right key-bit. Figure 6.9 shows the changes in correlations between the columns of M_3 and M_4 and the five columns of M_2 which correspond to the five spikes in Fig. 6.8 according to the number of measurements used.

It is visible from Fig. 6.9 that the correlations for the predictions for the $k_{l-2} = 1$ guess are higher than the correlations for the predictions for the $k_{l-2} = 0$ guess. By just using the first 500 measurements the decision of $k_{l-2} = 1$ can be made for all the spikes expect the third spike. For the third spike the correlation for the $k_{l-2} = 1$ guess becomes higher by using around 4000 measurements. As it can be seen from Fig. 6.8, the third spike is the highest one of all the five spikes, hence the number of registers that are updated at this moment is higher than for the other spikes. Since we count the transitions of only the coordinates of Q_1, the signal to noise ratio (SNR) is lower for the third spike. This shows that the decrease of the SNR results in a larger number of measurements for a successful attack.

We have calculated the SNR of the power consumption measurements. The signal is calculated by averaging the 100 measurements for the same point on the elliptic curve. The noise is calculated by subtracting the signal from one of the measurements. The we use Eq. (3.2) in order to calculate the SNR. Figure 6.10 shows the SNR for all the clock cycles in Fig. 6.8. The SNRs at all the clock cycles which five spikes occur are above 20dB. This means we can distinguish the signal from the noise. The highest SNR occurs at the clock cycle for the 5th spike and the lowest for the 3rd spike. This result matches



(a) First spike

(b) Second spike

(c) Third spike

(d) Fourth spike

(e) Fifth spike

Figure 6.9: Changes in correlations between the current consumption measurements and the predictions of the five spikes in Fig. 6.8 according to the number of measurements

with the findings by the correlation analysis above.

6.2.2.2 Distance of Mean Test

We produce a new measurement matrix, M_5, from M_2 by taking 20 columns around each column of M_2 that corresponds to a spike in Fig. 6.8. Thus, M_5 has 100 columns and N rows. We use the prediction matrices M_3 (for $k_{l-2} = 0$ guess) and M_4 (for $k_{l-2} = 1$ guess) explained in Section 6.2.2.1 in order to split the measurements into sets. For each spike in Fig. 6.8 there are two sets for $k_{l-2} = 0$ guess and $k_{l-2} = 1$ guess. If the measurements are split

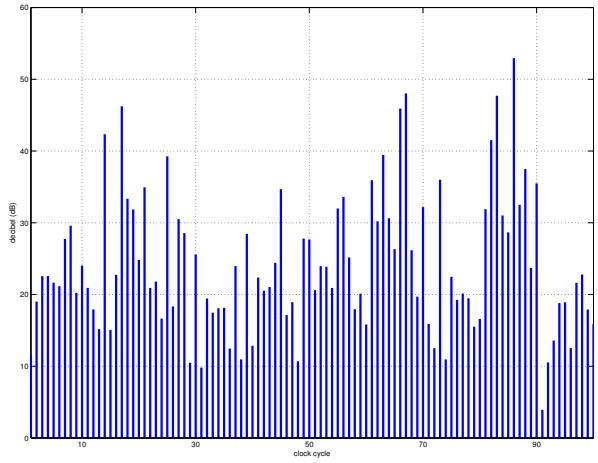


Figure 6.10: Signal to noise ratio of the power consumption measurements

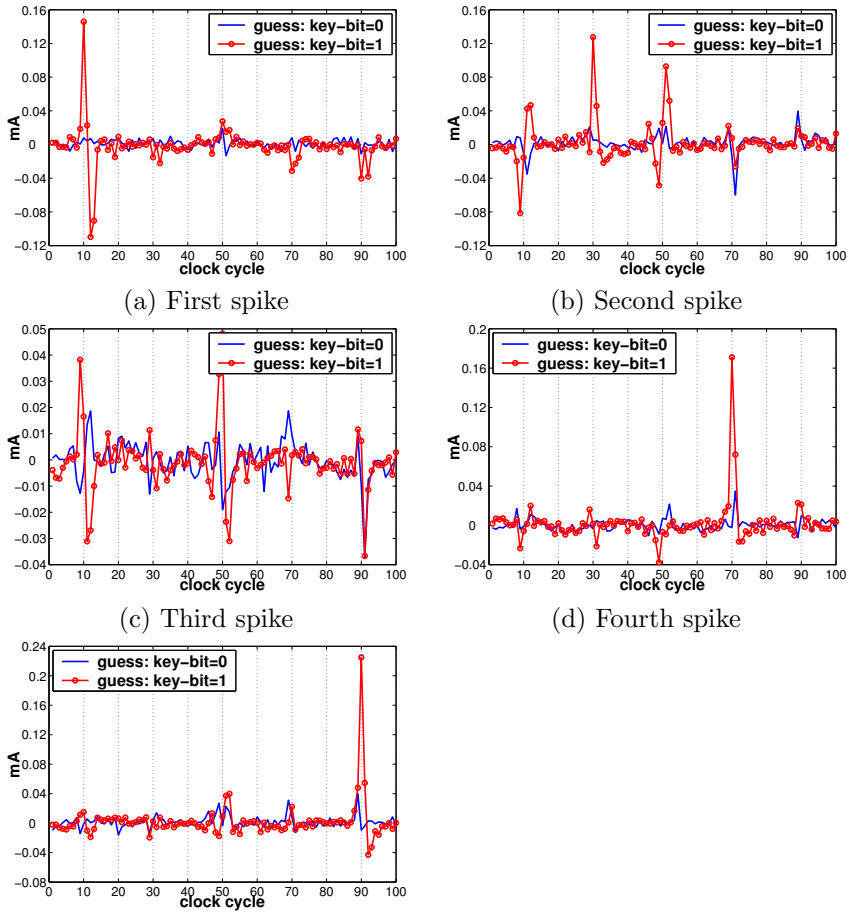
according to the predictions of the ith ($i = 1, 2, \cdots, 5$) spike then we calculate the mean value of the ith column of M_3 and M_4, $E(M_3(:, i))$ and $E(M_4(:, i))$, respectively. $M_3(:, i)$ is the ith column of M_3. If the prediction for the jth ($j = 1, 2, \cdots, 10\,000$) measurement for $k_{l-2} = 0$ guess is smaller than $E(M_3(:, i))$ ($M_3(j, i) < E(M_3(:, i))$) then jth measurement is put in set $S_{i,1,1}$, otherwise in set $S_{i,1,2}$. If the prediction for the jth ($j = 1, 2, \cdots, 10\,000$) measurement for $k_{l-2} = 1$ guess is smaller than $E(M_4(:, i))$ ($M_4(j, i) < E(M_4(:, i))$) then jth measurement is put in set $S_{i,2,1}$, otherwise in set $S_{i,2,2}$.

Then we calculate the mean values of all the sets and find the bias signal for the ith spike and the jth guess as follows

$$T_{i,j} = E(S_{i,j,2}) - E(S_{i,j,1})\,.$$

The current consumption bias signals for the five spikes and for $k_{l-2} = 0$ and $k_{l-2} = 1$ guesses are shown in Fig. 6.11. All the figures show a high peak on the expected spot on the line for the $k_{l-2} = 1$ guess. Hence the decision for the right key-bit is again as 1.

In order to compare the correlation analysis and distance of mean test we should also find the number of measurements needed to find the right key bit for the distance of mean test. Figure 6.12 shows the change in the amplitude of all the clock cycles on the current consumption bias signals for all the spikes for the $k_{l-2} = 1$ guess. The number of measurements on these traces are the number of measurements in the sets $S_{i,2,1}$, $S_{i,2,2}$ described above. The number of measurements in these sets are nearly the same. Hence we should multiply



(a) First spike

(b) Second spike

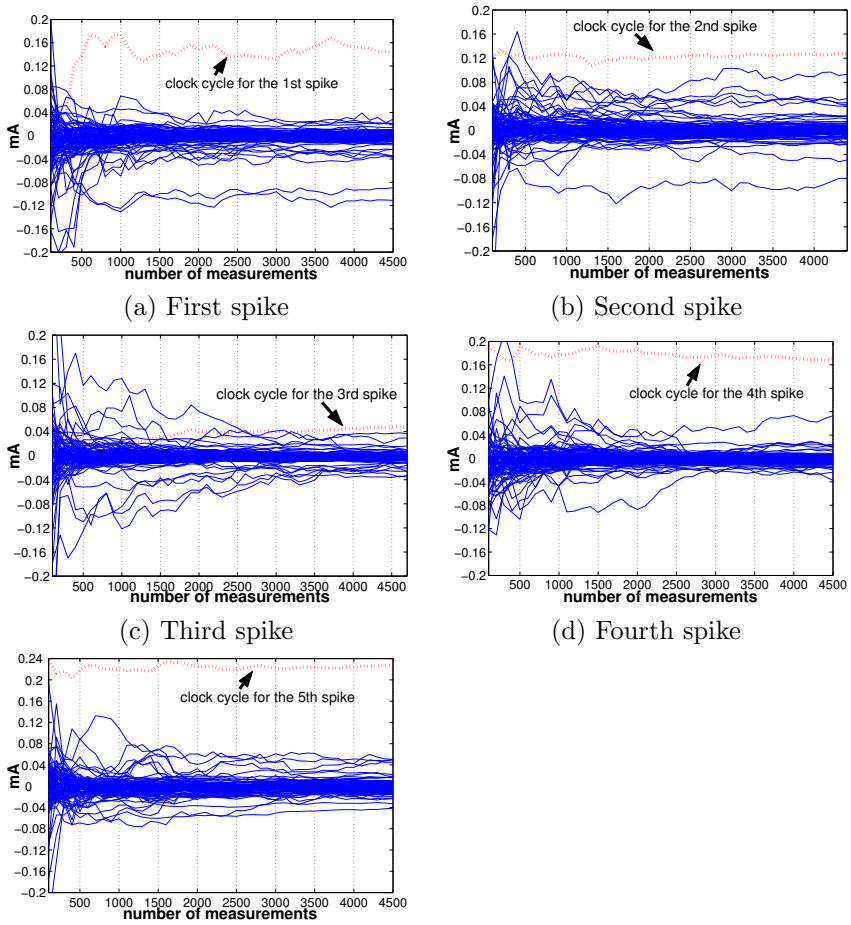
(c) Third spike

(d) Fourth spike

(e) Fifth spike

Figure 6.11: Current consumption bias signals for the five spikes in Fig. 6.8 and the $k_{l-2} = 0$ and $k_{l-2} = 1$ guesses

the number of measurements seen in Fig. 6.12 by two in order to find the needed number of measurements. As it is shown in Fig. 6.12.(c) 9000 measurements are not enough to distinguish the right clock cycle from the wrong ones. For the first, second, fourth and fifth spikes the right clock cycle is visible by using 1000, 2000, 1000 and 500 measurements in Fig. 6.12.(a), (b), (d) and (e), respectively. When we compare the results shown in Fig. 6.9 and Fig. 6.12, we should use two times more measurements for the distance of mean test than for correlation analysis.



(a) First spike

(b) Second spike

(c) Third spike

(d) Fourth spike

(e) Fifth spike

Figure 6.12: Change in the amplitude of the current consumption bias signal for the five spikes in Fig. 6.8, the $k_{l-2} = 1$ guess and for all clock cycles

6.3 Electromagnetic Analysis of a FPGA Implementation of Elliptic Curve Cryptosystem over $GF(p)$

In this section, we conduct electromagnetic analysis (EMA) attacks on the FPGA implementation of our elliptic curve processor over $GF(p)$ presented in Section 4.5. First, we demonstrate that our initial design is vulnerable to simple electromagnetic analysis (SEMA) attack and that it is possible to obtain the key with only one measurement. Then we conduct differential electromagnetic

analysis attack on our improved design explained in Section 6.2.1.

6.3.1 Simple Electromagnetic Analysis of a FPGA Implementation of Elliptic Curve Cryptosystem over $GF(p)$

In our measurement we connect an antenna directly to an oscilloscope as shown in Fig. 6.13.

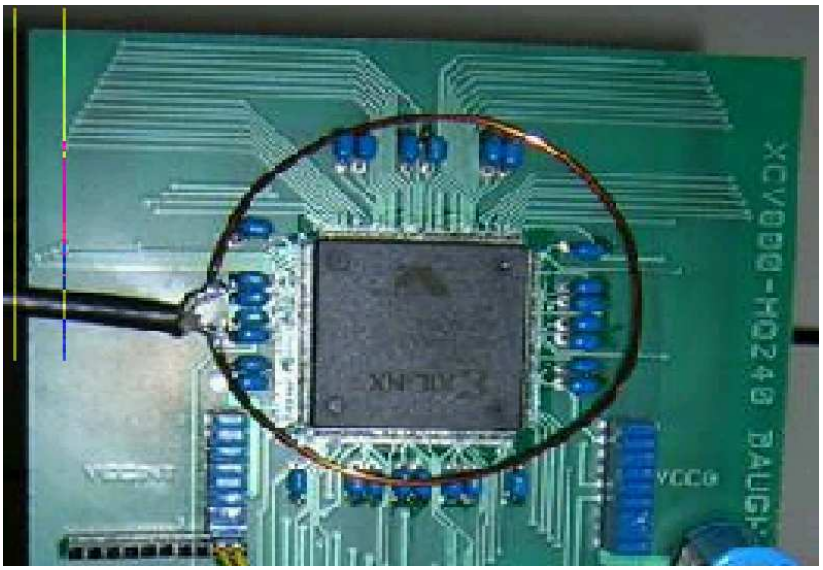


Figure 6.13: The measurement setup. The loop antenna is placed vertically on the FPGA.

The electromagnetic radiation trace of a 160-bit EC point multiplication is shown in Fig. 6.14 [34]. The SEMA attack is implemented on the EC processor given in Section 4.5 which uses Algorithm 2.1 for EC point multiplication. It can be easily seen from Fig. 6.14 that the key used during this measurement is 11001100, because there is a clear difference between the traces of EC point addition and doubling. The SEMA attack was successful because of the conditional branch in Step 4 of Algorithm 2.1.

As a countermeasure to this attack we have implemented the EC point multiplication with Algorithm 6.1. One EM measurement of this architecture is shown in Fig. 6.15.

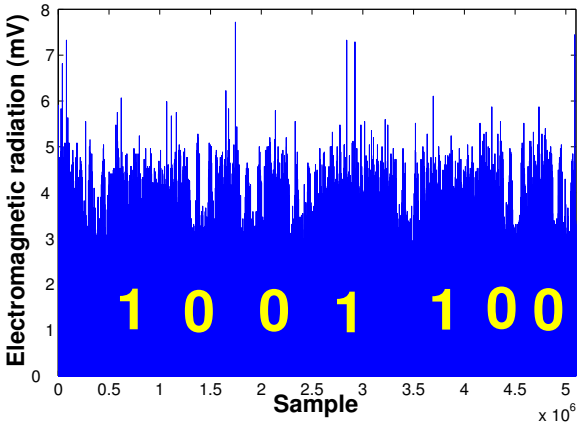


Figure 6.14: Electromagnetic radiation trace of a 160-bit elliptic curve point multiplication over $GF(p)$ with Algorithm 2.1

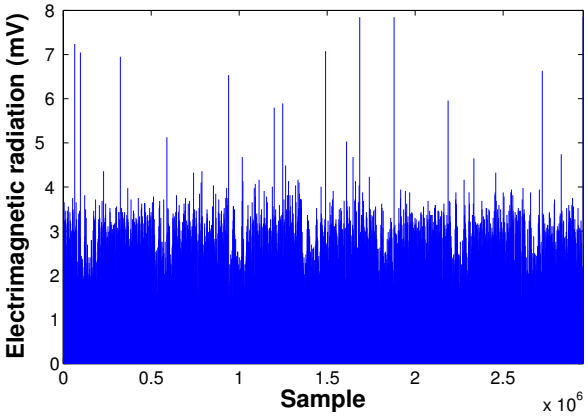


Figure 6.15: Electromagnetic radiation trace of a 160-bit elliptic curve point multiplication with Algorithm 6.1

6.3.2 Differential Electromagnetic Analysis of a FPGA Implementation of Elliptic Curve Cryptosystem over $GF(p)$

The target for our DEMA attack is the second MSB of the key, k_{l-2}, in Algorithm 6.1. If $k_{l-2} = 0$, then Q will be updated by $2P$, otherwise by $3P$ at step

5 in Algorithm 6.1.

6.3.2.1 Correlation Analysis

In the first step of our attack, we have produced an electromagnetic radiation file. For this purpose, we have measured the electromagnetic radiation at the same time as the power consumption measurements given in Section 6.2.2.1 and produced M_1 in the same way. The electromagnetic radiation trace of one of these measurements is shown in Fig. 6.16.

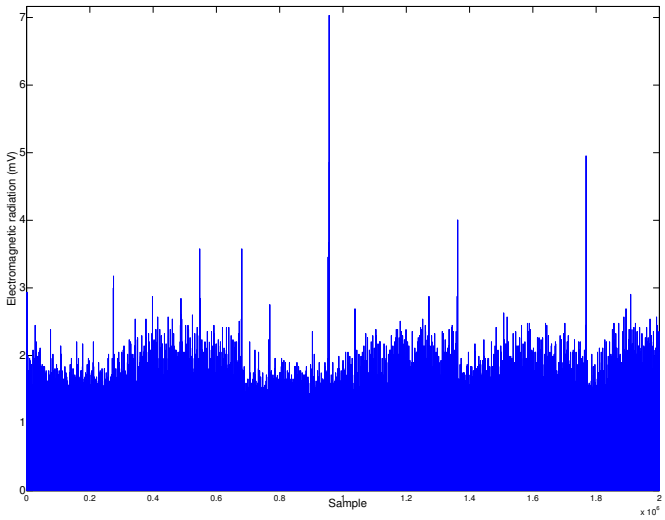


Figure 6.16: Electromagnetic radiation trace of the FPGA during the execution of one EC point multiplication for a DEMA attack (1st row of M_1)

We have applied the same pre-processing technique as in Section 6.2.2.1 to data from the oscilloscope. The DFT of the first measurement between 250 kHz and 375 kHz is shown in Fig. 6.17. Figure 6.18 shows the $1st$ measurement after taking the maximum value in every clock cycle.

We have predicted the electromagnetic radiation of the FPGA during the events which correspond to the five spikes shown in Fig. 6.18 for $k_{l-2} = 0$ and $k_{l-2} = 1$ guesses as explained in Section 6.2.2.1.

It is visible from Fig. 6.19 that the correlations for the predictions for the $k_{l-2} = 1$ guess are higher than the correlations for the predictions for the $k_{l-2} = 0$ guess in Fig. 6.19.(b), (d) and (e). By just using the first 1000

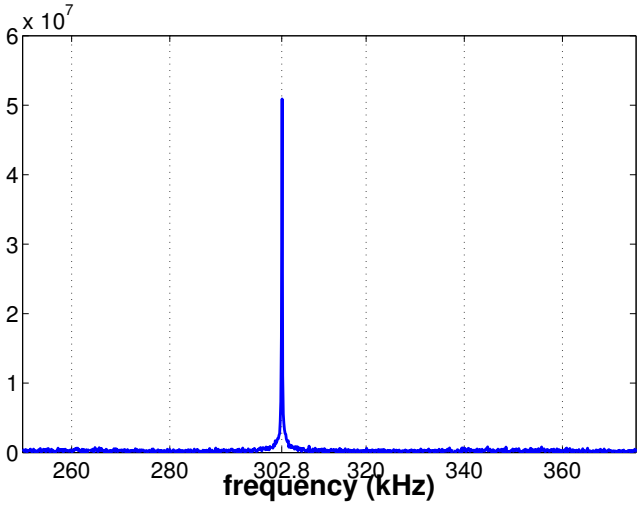


Figure 6.17: DFT of the electromagnetic radiation trace of the first measurement between 250 kHz and 375 kHz; clock frequency is 302.8 kHz

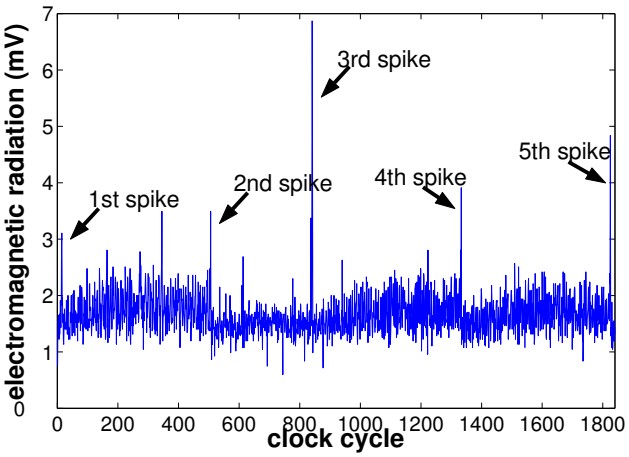
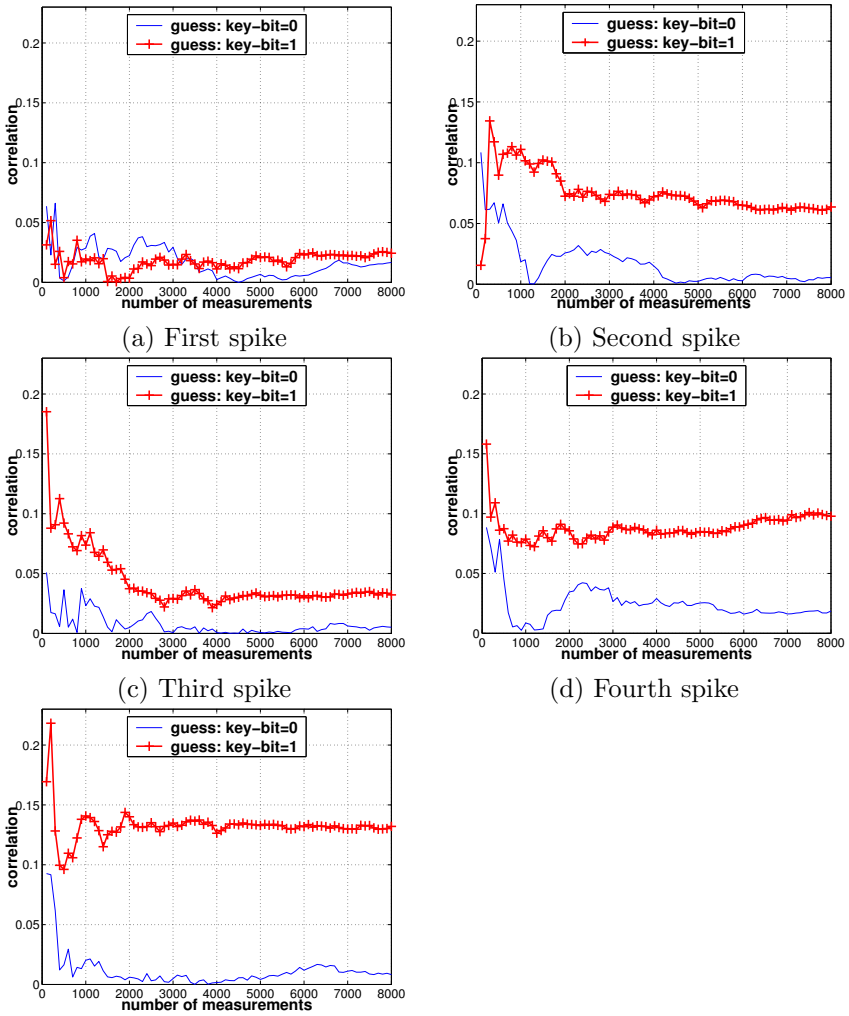


Figure 6.18: Electromagnetic radiation trace of the first measurement after taking the maximum value in every clock cycle (1st row of M_2)

measurements the decision of $k_{l-2} = 1$ can be made for the fourth and fifth spikes. For the second spike the correlation for the $k_{l-2} = 1$ guess becomes higher by using around 2000 measurements. The minimum necessary number of



(a) First spike

(b) Second spike

(c) Third spike

(d) Fourth spike

(e) Fifth spike

Figure 6.19: Changes in correlations between the electromagnetic radiation measurements and the predictions of the five spikes in Fig 6.18 according to the number of measurements

measurements in order to be successful with a DEMA attack is two times larger than the minimum necessary number of measurements for the DPA attack given in Section 6.2.2.1. This proves that the EM radiation measurements are more noisy than the power consumption measurements. But the information can be obtained with enough measurements.

We have also calculated the SNR of the electromagnetic radiation measure-

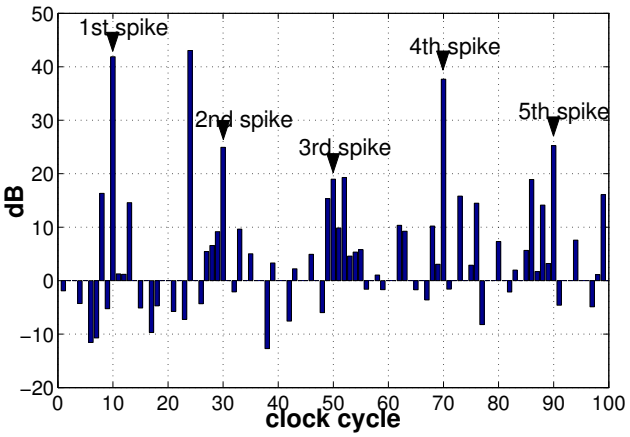


Figure 6.20: Signal to noise ratio of the electromagnetic radiation measurements

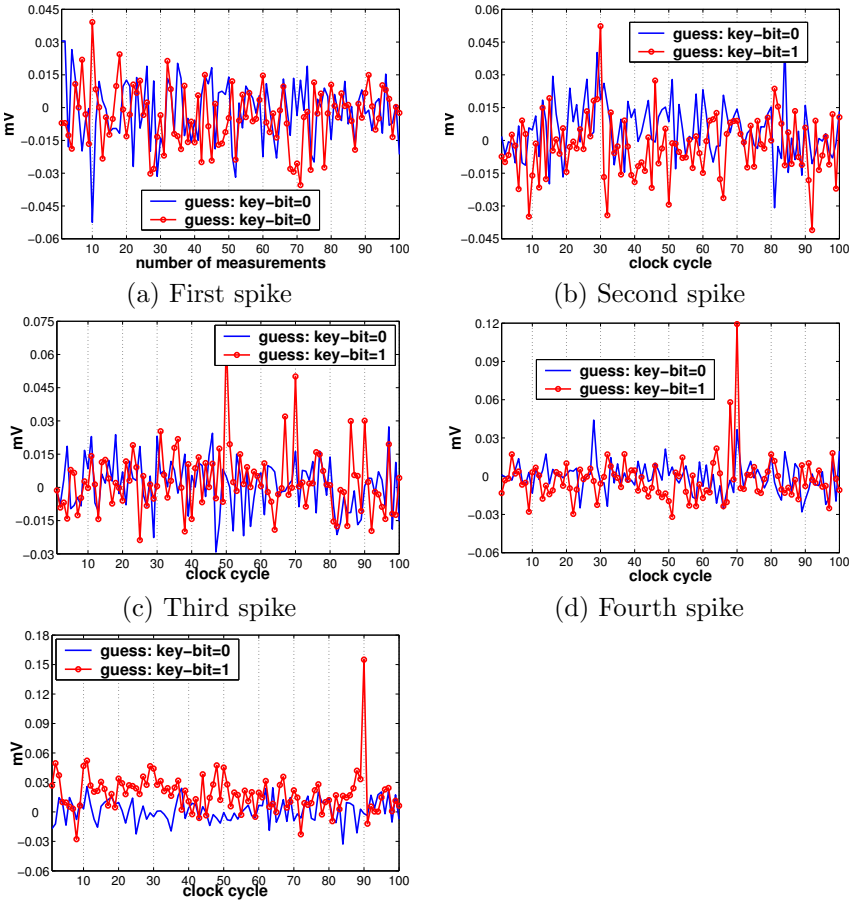
ments. The signal and the noise are calculated the same way as explained in Section 6.2.2.1. Figure 6.20 shows the SNR for all the clock cycles in Fig. 6.18. The SNRs at all the clock cycles which five spikes except the third spike occur are above 20dB. This means we can distinguish the signal from the noise.

When we compare Fig. 6.10 and Fig. 6.20 we see that the SNR of the electromagnetic radiation measurements are lower than the SNR of the power consumption measurements. This difference is the reason of the need for more measurements for the DEMA attack than DPA attack.

6.3.2.2 Distance of Mean Test

The EM radiation measurements are split into the sets with the same partitioning function as defined in Section 6.2.2.2. The electromagnetic bias signals for the five spikes and for guesses $k_{l-2} = 0$ and $k_{l-2} = 1$ are shown in Fig. 6.21. All the figures for all the spikes show high peaks on the expected spot on the line for the $k_{l-2} = 1$ guess in Fig. 6.21. Hence the decision for the right key bit is again as 1.

Again we find the number of measurements needed to find the right key bit for distance of mean test. Figure 6.22 shows the change in the amplitude of all the data points on the EM radiation bias signals for all the spikes for the $k_{l-2} = 1$ guess. As it is shown in Fig. 6.22.(a) 9000 measurements are not enough to distinguish the right clock cycle from the wrong ones. For the second, third, fourth and fifth spikes the right clock cycle is visible by using 5000, 7000, 2000 and 2000 measurements in Fig. 6.22.(b), (c), (d) and (e), respectively. When we



(a) First spike

(b) Second spike

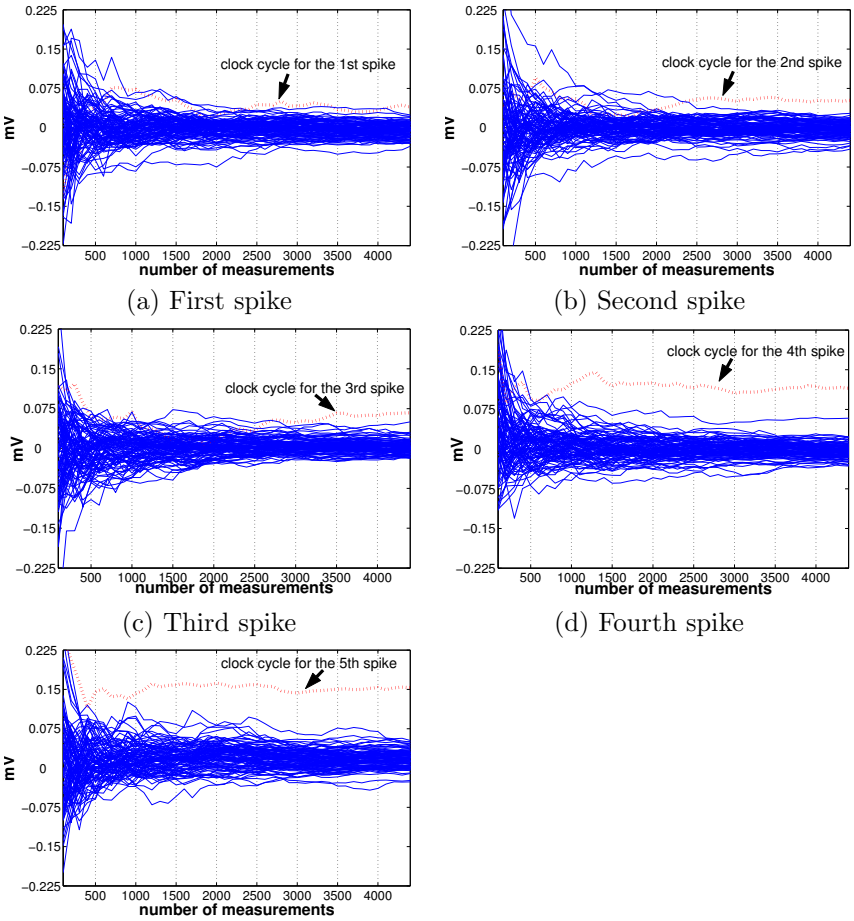
(c) Third spike

(d) Fourth spike

(e) Fifth spike

Figure 6.21: Electromagnetic radiation bias signals for the five spikes in Fig. 6.18 and the $k_{l-2} = 0$ and $k_{l-2} = 1$ guesses

compare the results shown in Fig. 6.19 and Fig. 6.22, we should use five times more measurements for the distance of mean test than for correlation analysis. The minimum necessary number of measurements in order to be successful with a DEMA attack is two times more than the minimum necessary number of measurements for DPA attack given in Section 6.2.2.2, which is the same difference between the correlation analysis for DPA and DEMA.



(a) First spike

(b) Second spike

(c) Third spike

(d) Fourth spike

(e) Fifth spike

Figure 6.22: Change in the amplitude of the electromagnetic radiation bias signal for the five spikes in Fig. 6.18, the $k_{l-2} = 1$ guess and for all clock cycles

6.4 Conclusions

We have implemented timing, power and electromagnetic analysis attacks on our FPGA implementations of elliptic curve cryptosystems over $GF(p)$. We conclude that our initial design was vulnerable to simple attacks and by using only the timing information it is possible to find the Hamming weight of the key. More drastically by using simple power and electromagnetic analysis attacks it was possible to find all the key bits. Next we improved our circuits such that they are resistant against timing and simple power and electromagnetic analysis attacks. We showed that this improved design is vulnerable to differential

attacks.

We conduct differential power and electromagnetic analysis attacks on our improved FPGA implementation of a elliptic curve processor. We use two well known techniques for DPA and DEMA; correlation analysis and a distance of mean test. We conclude that the correlation analysis reveals the right key bit by using six times less measurements than the distance of mean test for both DPA and DEMA. The number of required electromagnetic radiation measurements is twice the power consumption measurements in order to find the right key bit. This result has several reasons. We use a very simple handmade antenna shown in Fig. 6.13. Because the diameter of the antenna is large and it is not isolated from any effect by any protection, the antenna collects all the signals in the air. We use the same prediction data for both power consumption and electromagnetic radiation, which is the number of transitions in the target registers. This is a correct model for power consumption as it is related to the current flow to the load capacitances, but the electromagnetic radiation is also effected by the direction of the current internally. So the predictions for electromagnetic radiation should be improved by taking into account of the position of the antenna on the FPGA and the relative direction of the current flows.

A step would be to implement and test the countermeasures against differential power and electromagnetic analysis attacks. The suggestions for the countermeasures are given in Section 3.3.2.4; they have not been implemented in this thesis because of the time limitation.

Chapter 7

Power Analysis of Hardware Implementations of Block Ciphers

During the AES selection process, the security of Rijndael was evaluated with respect to all types of attacks. While AES is resistant to classical cryptanalytic methods, it turned out that side channel attacks are a serious threat against naive implementations of the Rijndael algorithm. To our knowledge, there exists no publication on practical implementations of power analysis attacks on dedicated hardware implementations of the AES. This chapter demonstrates the feasibility of power analysis attacks against hardware implementations of the AES. Our attacks target against an ASIC implementation of the AES developed by the ETH Zürich [90] and an FPGA implementation of the AES developed by the UCL Crypto group [246]. These results allow us to compare the alternative ways of implementing the same algorithm with respect to side-channel attacks.

In 1977, the Data Encryption Standard (DES) Algorithm [176] was adopted as a Federal Information Processing Standard for unclassified government communication. Although a new Advanced Encryption Standard (AES, [175]) was selected in October 2000, triple-DES is still widely used, particularly in the financial sector. In this chapter we describe a power analysis attack against an FPGA implementation of the DES developed by the UCL Crypto group [222].

For hardware design efficiency, clock frequency and area requirements are of primary importance. In this chapter, we demonstrate that these aspects also have a substantial impact on the feasibility of power analysis attacks.

We target a part of the key in our DPA attacks on the hardware implementations of the AES and the DES. It is important to note that more key bits may be found using exactly the same set of measurements. The attacker only has to modify the power consumption predictions and target different key bits. In order to produce the predictions we implemented the algorithms in C++. The C++ program runs for the same set of inputs as given to the FPGA during the measurements. During the executing of the program, the number of bit changes between two consecutive values of the target part of the target register at the attack point is used as the power consumption prediction for the FPGA at the attack point. Therefore, the full key can be recovered by computing the power consumption predictions for all the parts of the key and finding the correlation between these predictions and the measurements.

The results of this chapter are published in [200, 199, 244, 245]. The outline of this chapter is as follows. In Section 7.1 we present our DPA attack on an ASIC implementation of the AES. In Section 7.2 we present our DPA attack on the FPGA implementation of the AES. In Section 7.3 we present our DPA attack on the FPGA implementation of the DES.

7.1 Power Analysis of an ASIC Implementation of the AES

In this section we present our DPA attack on the ASIC implementation of the AES. First we summarize the ASIC implementation of the AES. Then we introduce our measurement setup. We implement our attack with simulated data in order to verify that our attack strategy is correct. Finally we present the DPA attack with real measurements on the ASIC implementation of the AES.

7.1.1 Fastcore

Fastcore is an efficient ASIC realization of the AES algorithm in a standard 0.25 μm CMOS process with en/decryption rates in excess of 2 Gb/s [90]. Fastcore contains two separate datapaths for the encryption and decryption operations. Figure 7.1 shows a simple block diagram highlighting the encryption datapath structure of Fastcore. The encryption operation is performed on 128-bit values in parallel internally, but the external chip interface is limited to 16 bits for plaintext and ciphertext. The input and output buffers are used to store plaintext and ciphertext values and transfer them to/from the chip respectively. Each encryption round requires a round key, that is generated from the encryption key using a key schedule algorithm. The key schedule routine is implemented in the *Key Expansion Unit*. The round keys in Fastcore are

generated on-the-fly, parallel to the encryption operation.

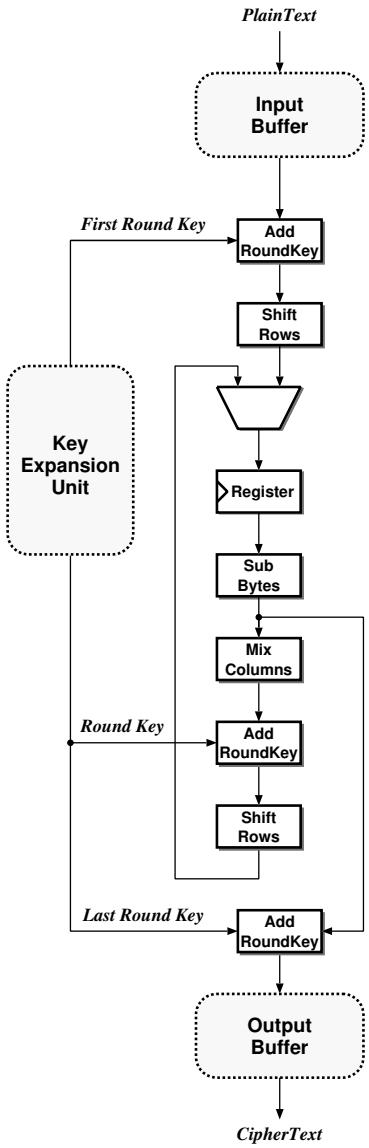


Figure 7.1: Block diagram of the encryption path of the AES crypto-chip, Fastcore

In Fastcore, the order of *SubBytes* and *ShiftRows* has been changed and the first *ShiftRows* operation has been moved to the initial *AddRoundKey* opera-

tion. The result is a functionally equivalent, but slightly different encryption round structure seen in Fig. 7.1. This transformation allows a more efficient implementation in hardware. The last encryption round shares the *SubBytes* operation with the standard encryption round followed by an additional *AddRoundKey* operation.

7.1.2 Measurements

The measurement setup consists of an HP83000 test system to provide the chip with the required inputs such as clock, data, key and control signals and a Tektronix 784C sampling oscilloscope with a Tektronix CT-1 current probe to measure the supply current. Fastcore uses two separate power supplies, a 3.3V supply for I/O and a 2.5V supply for the core cells. Only the core power supply has been measured. We neither took special measures to stabilize the power supply of the chip, nor did we try to characterize or minimize noise that might be induced by the measurement setup. Instead we have computed the average of the measurements to reduce noise. Using the averaging function of the sampling scope; we repeated each measurement 16 times. In the remainder of this section we will refer to the average measurements simply as measurements.

The HP83000 has been configured to perform an entire test run. Such a test run consists of initializing the crypto-chip, loading the encryption key, sending 10 000 plaintexts and comparing the results with the expected values. The sampling oscilloscope cannot sample the entire test run with the desired accuracy, hence the test system has been configured to generate a separate trigger signal at the beginning of each encryption operation. This signal has been used to sample and store the current multiple times for the clock cycles at and around the desired round of the encryption operation.

Fastcore contains a large number of functional blocks that can operate in parallel. Special care has been given to ensure that all unrelated blocks are either idle or compute the same results during the encryption operation. The decryption datapath is stalled and data I/O is not performed during encryption. The only other block that is active during an encryption operation is the key expansion unit. Since the same key is used throughout all the measurements, the key expansion unit calculates exactly the same intermediate results for the same encryption round. The test vectors ensure that of the 2758 flip-flops present in the Fastcore, only 128 flip-flops of the encryption round register have data dependent values and contribute to the difference in the power consumption.

If the functional blocks other than the encryption blocks are also active, the power consumption of them are added to the power consumption of the encryption block as noise. Because we do not have control on their activity and they work independently from the data we provide to the chip, we cannot predict the power consumption of the unused blocks during the encryption. Hence, in

this case we have to increase the number of measurements.

However, not all flip-flops within the encryption round register have the same drive strength (in total, there are 4 different drive strengths); they all are connected differently and the interconnection capacitance for each flip-flop is different. Furthermore, as a result of logic optimizations during the design phase, the number of gates driven by each flip-flop also differs slightly. Therefore, the contribution of each flip-flop to the overall dynamic power consumption is different. This can be clearly seen in Fig. 7.2, which shows the simulated dynamic power consumption of all the 128 flip-flops of the encryption round register separately.

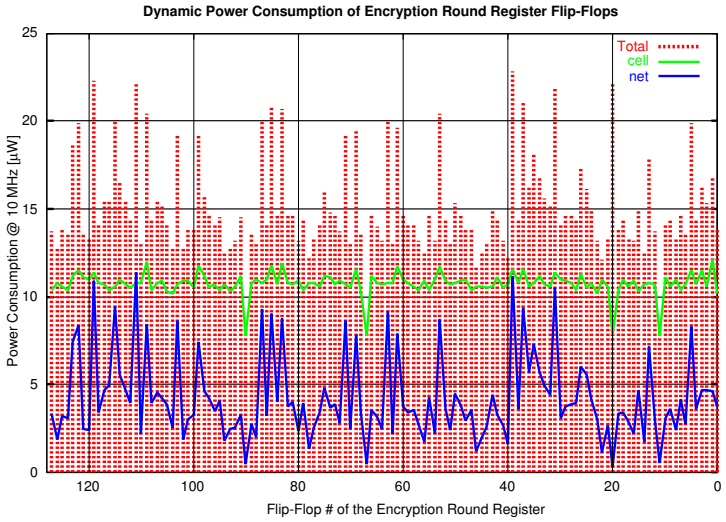


Figure 7.2: Power consumption characteristics of the Fastcore's registers

7.1.3 A Differential Power Analysis Using Simulated Data

The target for our DPA attack were the 8 MSBs of the `Register` in Fig. 7.1 after the initial key addition operation [200, 199]. Because the key used for this operation is the original key for encryption and the *ShiftRows* operation does not change the position of the 8 MSBs of the result of the *AddRoundKey* operation, we decided to predict the power consumption of the `Register` during the storage of these MSBs.

We have tested our attack with simulated data before making real measurements. This approach enabled us to estimate the difficulty of a real attack, *i.e.*, an attack using real measurements. In order to predict the dynamic power

consumption of the `Register`, behavioral HDL simulations of Fastcore were used. An advantage of this approach is that it allows to simulate attacks in an early stage of the design flow. Another reason for using a HDL simulation was that we did not reset the chip after each AES execution. At the beginning of an AES execution, the `Register` still contained some value which is related to the previous AES execution. Hence, without the HDL simulation, we could have only predicted the Hamming weight of the `Register`, but not the dynamic power consumption.

In the first step of this simulated attack, we have produced a so-called simulated power consumption file. For this purpose, we have chosen N random plaintexts and one fixed, but random key. After each encryption round (clock cycle), the simulator has written the total number of bit changes between the previous and the current values of the `Register` in Fig. 7.1 to this file. `Register` is used to store the state of the AES which is 128 bits (see Section 2.4.1). Fastcore computes one round of the AES in one clock cycle thus one encryption takes 10 clock cycles. Hence, the simulator has produced a file which contains an $N \times 10$ matrix ($N = 10\,000$), M_1, with values between 0 and 128.

In the second step, we have chosen the M MSBs of the `Register`. For the same plaintexts and key as in the first step, the simulator has calculated the total number of bit changes between the previous and the current values of these M MSBs of the `Register` for the initial key addition. This result was stored in a file as an $N \times 1$ matrix, M_2, which contains values between 0 and M. In this particular experiment, we have chosen $M = 8$.

Then, we have calculated the correlation between all the columns of M_1 and M_2 as follows:

$$c_i = C(M_1(1:N,i), M_2) \quad i = 1, \ldots, 10\,,$$

where $M_1(1:N,i)$ denotes the ith column vector of the matrix M_1.

In step 1 and step 2, the same plaintexts and the same key were used. The only difference between the two steps is the difference in the number of bits taken into account when counting the number of bit changes. Hence, the values generated in step 2 are a prediction for the values calculated in the initial key addition of step 1. If the calculations are correct, the correlation coefficient of M_2 and the first column of M_1 (which corresponds to the initial key addition) must be significantly higher than the correlation coefficients of M_2 and all the other columns of M_1. Figure 7.3 shows that this is indeed the case.

In the third step, we have repeated the second step with a different value for the key. Hence, we have produced an output file containing the matrix M_3.

As in step 2, we calculated the correlation coefficient of M_1 and M_3:

$$c_i = C(M_1(1:N,i), M_3) \quad i = 1, \ldots, 10\,.$$

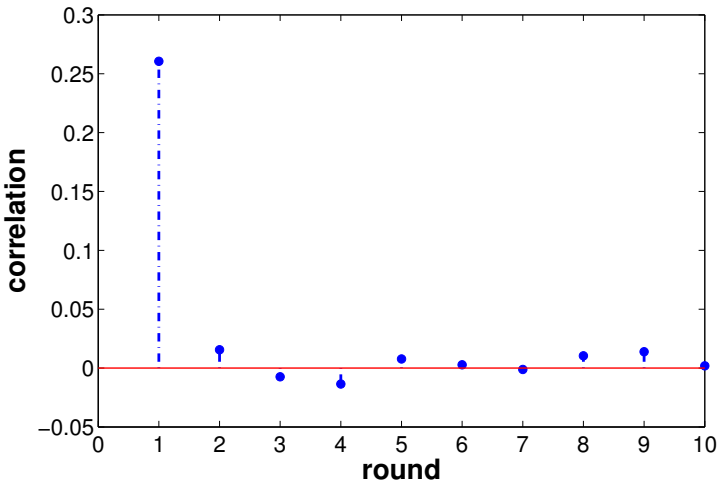


Figure 7.3: Correlation between all the columns of M_1 and M_2 with 10 000 plaintexts for the ASIC implementation of the AES

As we used another key to produce M_3, we expect no correlation between the columns of the matrices M_1 and M_3. Figure 7.4 shows that this is indeed the case. We conclude that our model, which consist of using predictions of the behavior HDL simulation, makes correct predictions for the simulated behavior of the Fastcore chip. From this result we expect that our model will make correct predictions for the real behavior also, but the real behavior will have more noise and we will have to use more plaintexts than the simulated attack in order to see the same results.

In the fourth and last step, we have extended the experiments performed in step 2 and step 3 in such a way that a full DPA attack on $L = 8$ bits was performed. Hence, we calculated an $N \times 2^L$ matrix M_4. Each column of the matrix M_4 contains the prediction for the bit changes in the `Register` for a particular guess of the L attacked key bits of the initial key addition. Eq. (7.1) shows how we can calculate the correlation coefficients between the predictions of all the possible keys and the first column of M_1:

$$c_i = C(M_1(1:N,1), M_4(1:N,i)) \quad i = 0,\ldots,2^L - 1\,. \tag{7.1}$$

From the previous steps we can expect that only one value, corresponding to the correct L key bits, leads to a high correlation coefficient. Figure 7.5 shows that this is indeed the case.

We have already demonstrated that our attack setup works well together with our model. The only question that remains is how many measurements, N, are needed to determine the correct key. In order to determine this minimum,

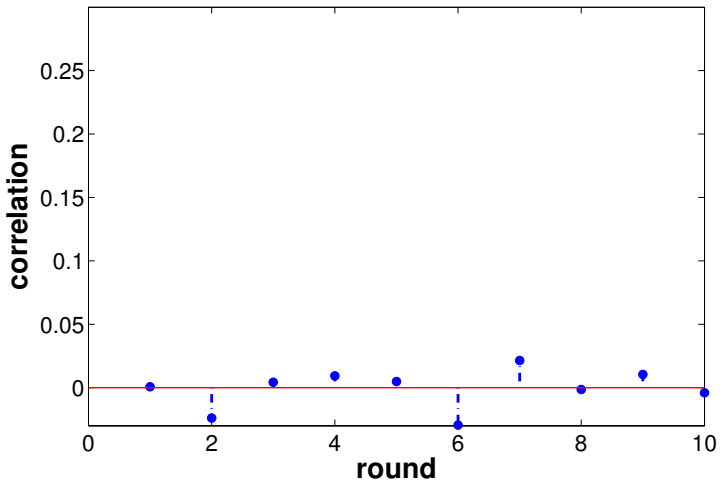


Figure 7.4: Correlation between all the columns of M_1 and M_3 with 10 000 plaintexts for the ASIC implementation of the AES

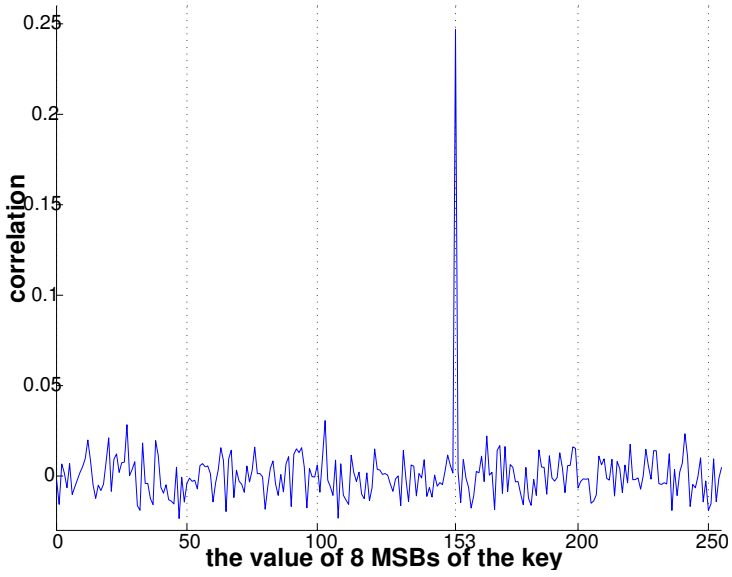


Figure 7.5: Correlation between the first column of M_1 and M_4 with 10 000 plaintexts for the ASIC implementation of the AES

we have calculated the correlation coefficient between M_1 and M_4 for different values of N:

$$c_{i,j} = C(M_1(1:i,1), M_4(1:i,j)) \quad i = 1, \ldots, 10\,000\,, j = 0, \ldots, 2^L - 1\,.$$

As shown in Fig. 7.6, after approximately 400 plaintexts the correct L MSBs can be distinguished from the wrong L MSBs. Hence, for the simulated attack, 400 measurements are sufficient to find the correct L MSBs of the key.

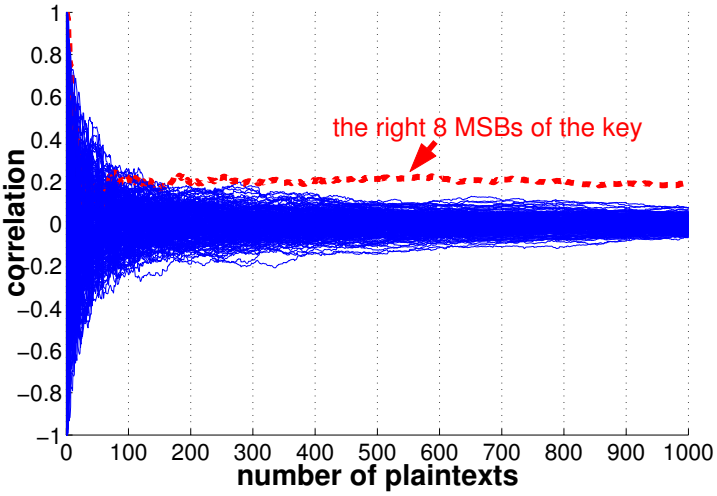


Figure 7.6: Correlation between the first column of M_1 and all the columns of M_4 as a function of the number of measurements for the ASIC implementation of the AES

7.1.4 A Differential Power Analysis Using Measured Data

In this section, we present the results of our DPA attacks on Fastcore using real, measured data. We have encrypted with Fastcore the same N plaintexts with the same key as used in the first step of Section 7.1.3. The initial key addition operation occurs during the first clock cycle. The result of this operation is written into the `Register` at the rising edge of the second clock cycle. Hence, we have measured the current consumption of Fastcore during the first two clock cycles of the encryption operation. The clock frequency applied to the chip was 2 MHz and the sampling frequency of the oscilloscope was 1 GHz. Hence, 500 samples were acquired per clock cycle. With these measurements, we have produced a $N \times 1000$ matrix, M_5. The power trace of one of these measurements, is shown in Fig. 7.7.

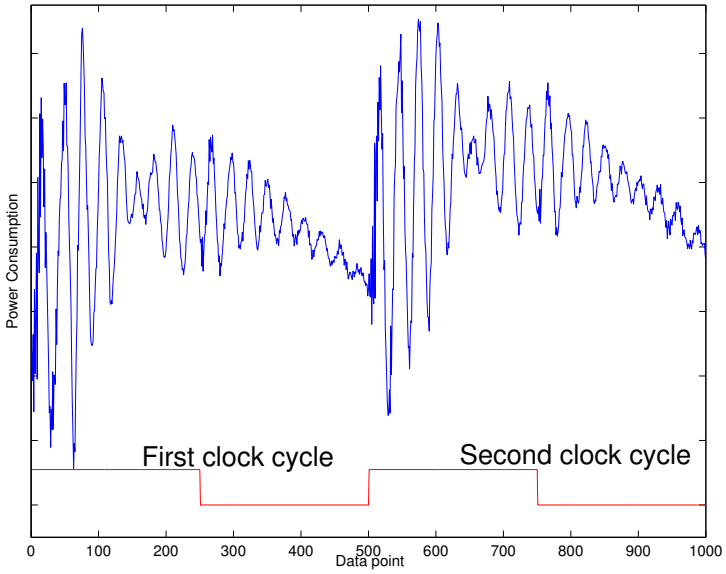


Figure 7.7: Current consumption trace of the 50th measurement of the ASIC implementation of the AES

In order to identify the correct L MSBs of the key we have used the correlation coefficient again. We have applied a pre-processing technique to reduce the noise in our measurements and to reduce the amount of measurement data. The pre-processing technique essentially consists of averaging. We have calculated the mean value of the measurement data in the second clock cycle as follows:

$$M_6(i) = E(M_5(i, D+1:2D)) \quad i = 1, \ldots, N\,,$$

where D is the number of data points measured during one clock cycle. $M_5(i, D+1:2D)$ is the vector which consists of the ith row and the columns between $D+1$ and $2D$ of M_5. We used these pre-processed measurements as input for our correlation analysis:

$$c_i = C(M_6, M_4(1:N, i)) \quad i = 0, \ldots, 2^L - 1\,.$$

As shown in Fig. 7.8 the highest correlation occurs at $i = 153$. This value corresponds to `0x99` which are the 8 MSBs of the key.

The critical path of Fastcore is around 7 ns. Thus, only the first data points of a measurement contain information which is directly related to the attacked operation. Hence, we decided to reduce the number of data points for the pre-processing step.

In order to determine the minimal number of data points that contain relevant

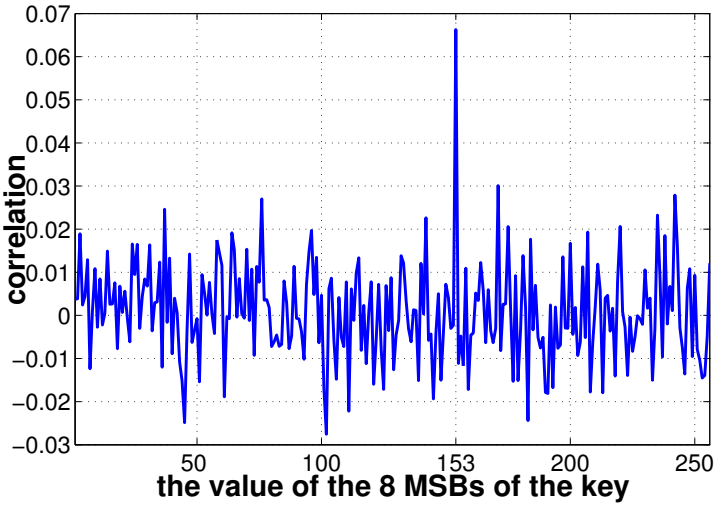


Figure 7.8: Correlation between all the columns of M_4 and M_6 with 10 000 measurements for the ASIC implementation of the AES

information, *i.e.*, information on the initial key addition, we have calculated the correlation coefficient between pre-processed measurement data (for a varying amount of data points in one clock cycle) and the column corresponding to the correct key of M_4:

$$M_7(i,j) = E(M_5(i, D+1 : D+j)) - E(M_5(i, D+1-j : D))$$

for $i = 1, \ldots, N$ and $j = 1, \ldots, D$.

$$c_i = C(M_7(1 : N, i), M_4(1 : N, 153)) \quad i = 1, \ldots, D\,.$$

Figure 7.9 shows that the highest correlation occurs when we use 50 data points around the rising edge of the second clock cycle.

Figure 7.10 depicts the correlation coefficients between all the columns of M_4 and the pre-processed data in column 50 of M_7. This figure shows clearly that the peak corresponding to the correct key becomes higher while the peaks corresponding to the incorrect key guesses stays constant compared to Fig. 7.8.

As in Section 7.1.3, N was taken as 10 000. However, we are interested in the smallest number of measurements that allow for a successful attack. In order to find the minimal number of measurements, we have calculated the following correlation coefficients:

$$c_{i,j} = C(M_7(1 : i, 50), M_4(1 : i, j))i = 1, \ldots, N\,, j = 0, \ldots, 2^L - 1\,.$$

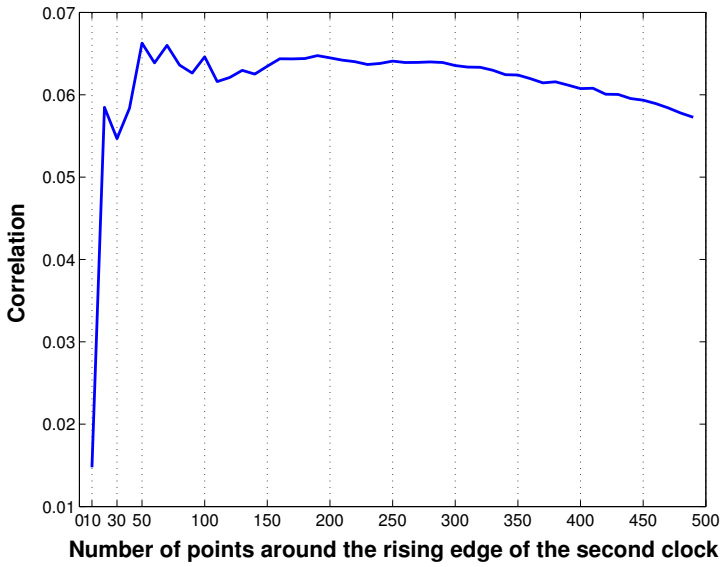


Figure 7.9: Correlation between $153th$ column of M_4 and all the columns of M_7 for the ASIC implementation of the AES

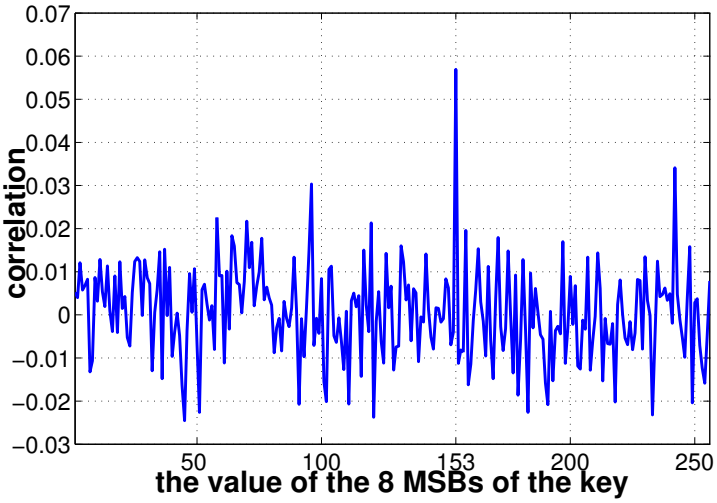


Figure 7.10: Correlation between all the columns of M_4 and the $50th$ column of M_7 with 10 000 measurements for the ASIC implementation of the AES

It is shown in Fig. 7.11 that after approximately 4000 measurements the correct and the wrong 8 MSBs of the key can be distinguished.

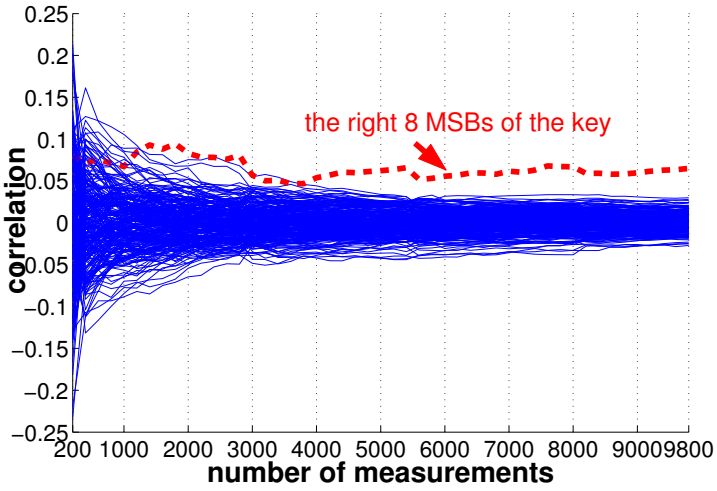


Figure 7.11: Correlation between all the columns of M_4 and the $50th$ column of M_7 as a function of the number of measurements for the ASIC implementation of the AES

The attack with simulated data in Section 7.1.3 needs about 400 measurements to deduce the correct key. Taking into account that the 4000 measurements are the averages of 64 000 real measurements (see Section 7.1.2) we conclude that we need 160 times more data to deduce the correct key.

7.2 Power Analysis of an FPGA Implementation of the AES

7.2.1 Hardware Description

The design used to investigate DPA attack against AES is described by Standaert *et al.* in [246]. The complete architecture is represented in Fig. 7.12, where all the registers contain 128 bits. It is a loop architecture with pipeline, designed for optimizing the ratio *Throughput (Mbits/s)/Area (slices)*. The resulting design implements the round (and key round) function in 5 clock cycles and the complete cipher in 52 clock cycles.

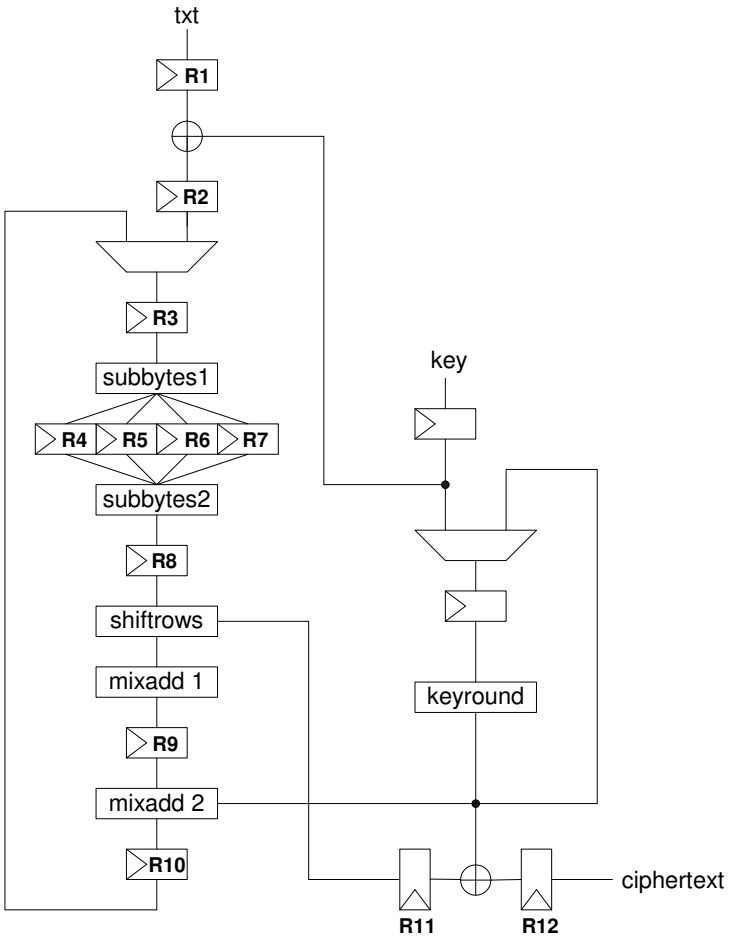


Figure 7.12: Architecture of an FPGA Implementation of AES

7.2.2 Predictions in a Pipeline Design

The problem we study in this section is whether pipelining has any influence on DPA attack resistance [244]. We also investigate a practical design that is the result of efficiency optimizations. Loop architecture is a relevant choice because it satisfies the usual area and throughput requirements for block cipher applications. However, unrolled architectures will also be explored in a further section.

Based on the hypothesis of Section 3.1.1, the first step in a power analysis attack is to make theoretical predictions on the power consumption. This can be done

using a selection function D that we define as follows. Let X_i and X_{i+1} be two consecutive values inside a target register. An estimation of the register's power consumption at the time of the transition is given by the function $D = H(X_i \oplus X_{i+1})$, where $H(X)$ is the Hamming weight of X. An attacker who has to predict the transitions inside the registers of an implementation therefore needs to answer two basic questions:

1. Which register transitions can we predict?
2. Which register transitions leak information?

Answering these questions determines which registers will be targeted during the attack. As an attacker can use the plaintexts (*resp.* ciphertexts) and predict transitions by partial encryption (*resp.* decryption), it is also important to evaluate both scenarios.

The *predictability* of a register is related to the number of key bits one should know to predict its transitions. For block ciphers, this depends on the size of the S-boxes and the diffusion layer. In practice, it is assumed that it is possible to guess up to 24 key bits and the diffusion layer usually prevents guessing of more than one block cipher round. In AES, S-boxes are 8 bits wide and their outputs are thus *predictable* after the first (*resp.* final) key addition. However, every *MixColumns* output bit depends on 32 key bits and is therefore computationally intensive to guess.

Definition 7.1 We denote a register as a *full* (*resp. empty*) register if its transitions leak (*resp.* do not leak) secret information. □

For example, it is obvious that an input (*resp.* output) register does not leak any secret information as it only contains the plaintext (*resp.* ciphertext).

A surprising consequence of the hypothesis introduced in Section 3.1.1 is that the registers following an initial (*resp.* final) key addition and which are not changed anywhere else in the algorithm do not leak information either. An example for this kind of register is `R2` in Fig. 7.12. To illustrate this statement, we use the following key addition:

AddKey

$result = input \oplus key$;

Let assume that the *result* is actually stored in the FPGA register $R2$ in Fig. 7.12. Let two consecutive inputs of the key addition be denoted as $input_1$ and $input_2$. Using the previously defined selection function, the register power consumption may be estimated by:

$$\begin{aligned} P_R \propto H(result_1 \oplus result_2) &= H(input_1 \oplus key \oplus input_2 \oplus key) \\ &= H(input_1 \oplus input_2)\,. \end{aligned} \tag{7.2}$$

Eq.(7.2) clearly specifies that register $R2$ is *empty*.

Note that this observation strongly depends on the hypothesis and selection functions used to perform the attack. Another surprising observation is that the register $R2$ may still leak secret information if reset signals are used. This is due to the constant state that reset signals introduce. Then, we have:

$$\begin{aligned} P_R \propto H(\text{“}all\ zeroes\text{”} \oplus result_1) &= H(\text{“}all\ zeroes\text{”} \oplus input_1 \oplus key) \\ &= H(input_1 \oplus key)\,, \end{aligned}$$

which makes the power consumption dependent on the key again. As a consequence, a secure hardware implementation should not apply reset signals to its inner registers in order to delete this additional information leakage. Note that a similar observation has been used to attack smart card implementations, where the constant state actually corresponds to a constant instruction address.

7.2.2.1 Predictions in AES

Figure 7.13 illustrates *predictable* and *full* registers when the AES design is filled with five different plaintexts, denoted $1, 2, \ldots, 5$, during the first eight clock cycles of an encryption. As an example, during the first cycle, register $R1$ contains plaintext 1 while all the other registers are undefined. During the second cycle, $R1$ contains plaintext 2, $R2$ contains the plaintext 1 and the other registers are undefined. Remark that in the eighth cycle, the multiplexer starts to loop and register $R3$ therefore contains data corresponding to plaintext 1 again.

Similarly, Fig. 7.14 illustrates *predictable* and *full* registers when the AES design is filled with five different plaintexts, denoted $1, 2, \ldots, 5$, during the last six clock cycles of an encryption. As an example, the register $R12$ contains the first ciphertext in the second cycle, ciphertext 2 in the third cycle and ciphertext 3 in the fourth cycle.

In the next section, we explain how theoretical predictions of the power consumption can be used to attack an FPGA implementation of the AES. Due to the size of the AES S-boxes, we predicted transitions in 8-bit registers for which the values depend on 8 key bits. In the following, predictions are consequently performed for 2^8 possible key guesses.

7.2.3 Description of a Correlation Analysis Attack

A correlation attack against an FPGA implementation of AES is divided into three steps [205, 30]. Let N be the number of plaintext/ciphertext pairs for which the power consumption measurements are accessible. Let K be the secret

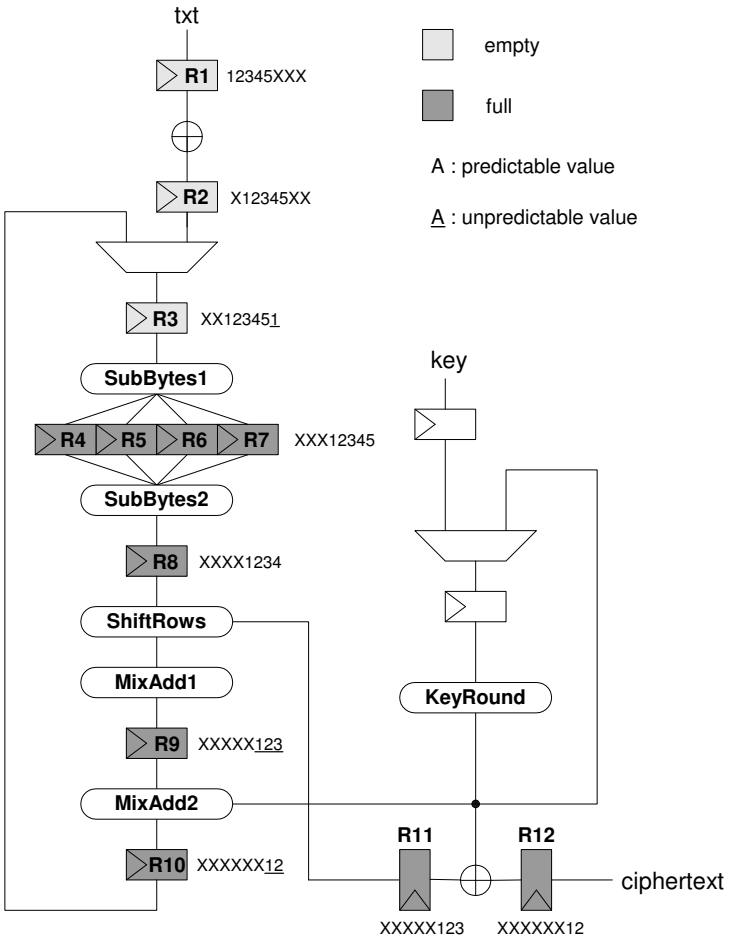


Figure 7.13: Status of the registers in the FPGA implementation of the AES during the first eight clock cycles

encryption key. When simulating the attacks, we assume that K is known to the attacker. For practical attacks, it is of course unknown.

7.2.3.1 Prediction Phase

For each of the N encrypted plaintexts, the attacker first selects the target registers and clock cycle for the previously defined selection function D. In Fig. 7.13, we see that between cycles 7 and 8, registers $R4, R5, R6, R7, R8, R11$ and $R12$ are *full* and have *predictable* and defined values. Similarly, in Fig. 7.14, we observe that between cycles 1 and 2, registers $R3, R4, R5, R6, R7$ and $R10$

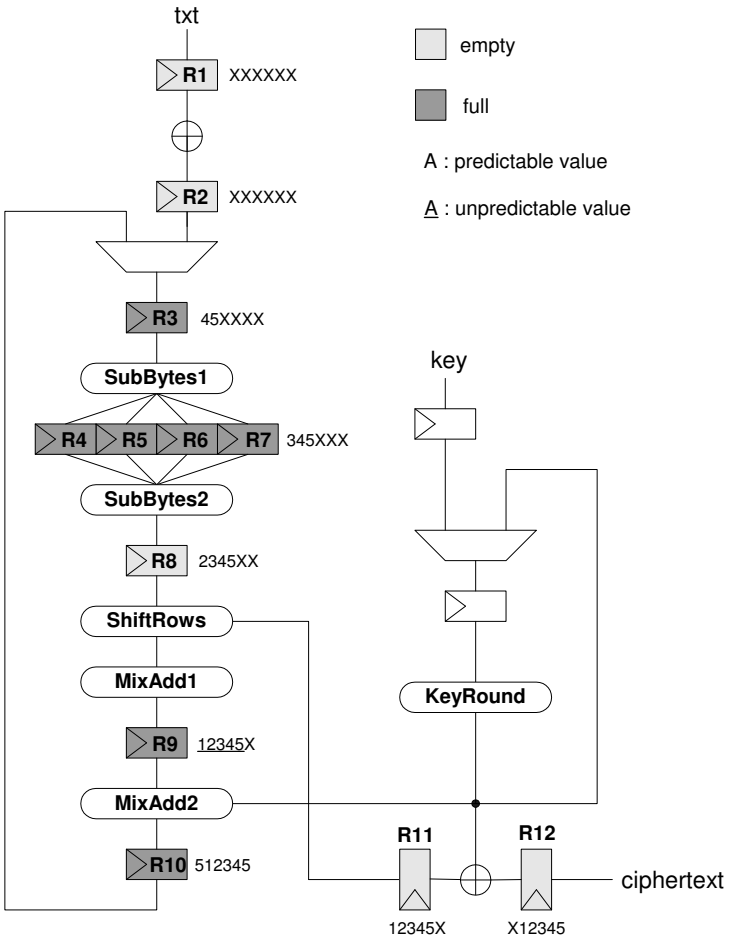


Figure 7.14: Status of the registers in the FPGA implementation of the AES during the last six clock cycles

are *full* and have *predictable* and defined values. Depending on the knowledge of the design, these registers can therefore be targeted. Due to the size of the AES S-box, the predictions are performed on 8 bits and may be repeated for every 8-bit part of a register R_i.

Let t be the number of 8-bit registers targeted by the attacker. Then, he predicts the value of D (*i.e.,* the number of bit switches inside the target registers in the targeted clock cycle) for the 2^8 possible key guesses and N plaintexts. The result of the prediction phase is an $N \times 2^8$ **selected prediction matrix**, containing integers between 0 and $8 \times t$. For simulation purposes, it

is also interesting to produce the **global prediction matrix** that contains the number of bit switches inside all the 12 registers of the design, for all the cycles. That is, if the encryption is performed in 52 clock cycles, we obtain a $N \times 52$ matrix, containing integers between 0 and $12 \times 128 = 1536$. This is only feasible if the key is known. In accordance with the hypothesis of Section 7.2.2, these matrices give estimations for the power consumption of the device. Since the same key is used for all the measurements, the power consumption of the key schedule is fixed and may be considered as a DC component that we can neglect as a first approximation.

7.2.3.2 Measurement Phase

During the measurement phase, we let the FPGA encrypt the same N plaintexts with the same key, as we did in the prediction phase. While the chip is operating, we measure the power consumption for the 52 consecutive clock cycles. Then, the power consumption trace of each encryption is averaged 10 times in order to remove the noise from our measurements; we store the maximum values of each encryption cycle so that we produce an $N \times 52$ matrix with the power consumption values for all the plaintexts and clock cycles. We denote it as the **global consumption matrix**.

7.2.3.3 Correlation Phase

In the correlation phase, we compute the correlation coefficient between a column of the global consumption matrix (corresponding to the cycle targeted by the prediction phase) and all the columns of the selected prediction matrix (corresponding to all the 2^8 key guesses). If the attack is successful, we expect that only one value, corresponding to the correct key guess, leads to a high correlation coefficient.

Finally, theoretical predictions of the attack can be performed by using the global prediction matrix instead of the global consumption matrix. As the global prediction matrix contains the number of bit switches inside all the registers, it represents a theoretical noise free measurement and may help to determine the minimum number of texts needed to mount a successful attack, *i.e.,* an attack where the correct key guess leads to the highest correlation coefficient. This is investigated in the next section.

7.2.4 A DPA Attack Using Simulated Data

In this section, we study the influence of the number of registers predicted on the efficiency of the attack. Several scenarios can be considered that correspond to different abilities of the attacker. In the most basic case, the attacker does not have any information about the design and has to make assumptions about

its implementation. A reasonable assumption is that the S-box outputs will be stored in registers. This is usually the case in the AES because S-boxes are the most time (and space) consuming parts of the algorithm. Therefore, the attacker will only predict the switching activity of 8 bits in $R8$ (in encryption) or $R3$ (in decryption). In the first step of the simulated attack, we produce the **selected prediction matrix** and **global prediction matrix** as defined in the previous section. Thereafter, we perform the correlation phase between these two matrixes. If the attack is successful, we expect that only one value, corresponding to the correct key guess, leads to a high correlation coefficient.

As the attacker is interested to determine the minimum number of plaintexts necessary to extract the correct key, we calculated this correlation coefficient for different values of $N : 1 \leq N \leq 4096$. As shown in Fig. 7.15, after approximately 1500 plaintexts the right 8 key bits can be distinguished from a wrong guess. We may therefore say that the attack is **theoretically successful** after about 1500 plaintexts.

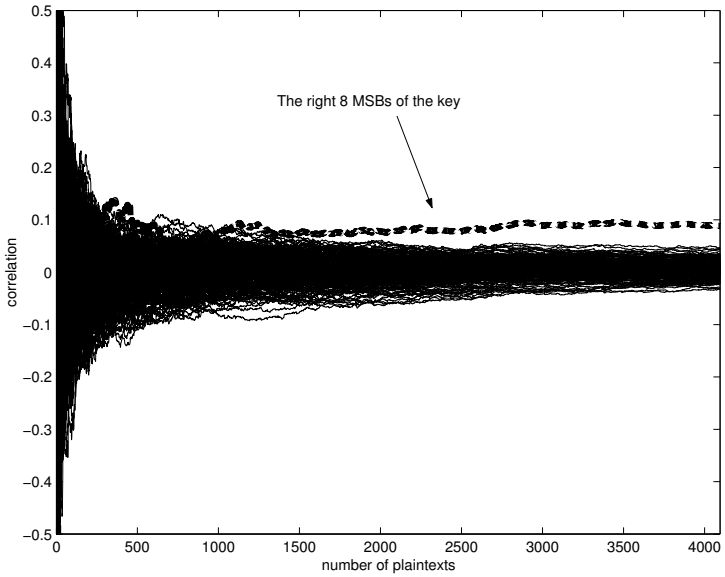


Figure 7.15: Change in the correlation coefficient as a function of the number of plaintexts for a simulated attack on the FPGA implementation of the AES by predicting only register $R8$ during the encryptions

As explained above an attacker who has no information about the implementation of the AES will only predict the switching activity of 8 bits in $R8$ (in encryption) or $R3$ (in decryption). In a more advanced scenario, the attacker has access to some implementation details (for example the scheme of Fig. 7.13).

Hence he knows that there are five pipeline stages in one round of the AES and twelve registers are used in the complete implementation of the AES. Then may determine the *predictable* and $full$ registers in the scheme of Fig. 7.13. Instead of just predicting the power consumption of $R8$, he can predict the power consumption of all twelve registers. Based on the complete predictions of Fig. 7.13, the correlation coefficient values for every key guess as a function of the number of plaintexts are represented in Fig. 7.16.

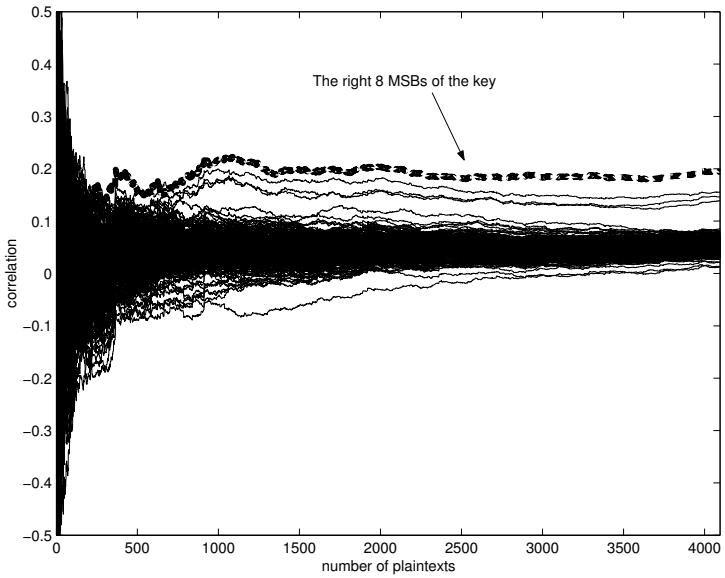


Figure 7.16: Change in the correlation coefficient as a function of the number of plaintexts for a simulated attack on the FPGA implementation of the AES by predicting all the *full* registers during the encryptions

We observe that the correct key guess is distinguishable after about 500 plaintexts, but stays closely correlated to three other candidates. As the S-box is a large multiplexer with two pipeline stage six input bits are actually used to select the values in registers $R4, R5, R6, R7$ [246]. Thereafter, two last bits select the final result of $R8$. As a consequence, if the key guess is such that the first six input bits of the S-box remain unchanged, the values stored in registers $R4, R5, R6, R7$ are the same. Only the S-box output in register $R8$ will differ. As there are four such key guesses, we will have four closely correlated candidates, including the correct one, what we can clearly observe in Fig. 7.16

A solution to this problem is to use the decryption predictions of Fig. 7.14. Then, even if only one bit differs at the output of the S-box (in $R8$), it will not result in the same intermediate register transitions. Based on these predictions,

the correlation coefficient values for every key guess and different number of traces are represented in Fig. 7.17, where the correct key candidate is clearly distinguishable after about 500 plaintexts.

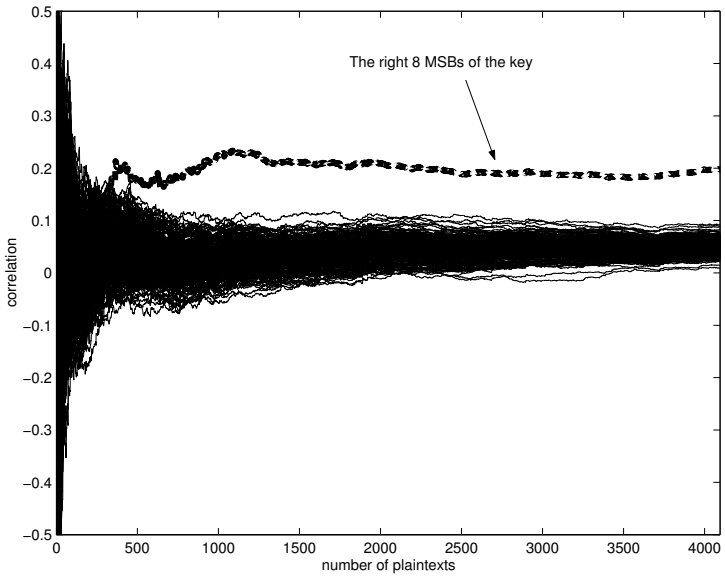


Figure 7.17: Change in the correlation coefficient as a function of the number of plaintexts for a simulated attack on the FPGA implementation of the AES by predicting all the *full* registers during the decryptions

7.2.5 A DPA Attack Using Practical Measurements

When we attack a device physically, the selected prediction matrix remains unchanged while we replace the global prediction matrix by the real measured global consumption matrix. Therefore, we encrypt 4096 plaintexts on the FPGA with the same key as we have done in the previous section and produce the matrix as described in Section 7.2.3.2.

In order to evaluate the quality of our theoretical predictions, we made a preliminary experiment and computed the correlation coefficient between one (in practice the 26th) column of the **global prediction matrix** and every column of the **global consumption matrix**. Figure 7.18 clearly illustrates that the highest correlation value appears for the predicted round, and therefore confirms that our predictions are correlated with real measurements.

In order to identify the correct 8 MSBs of the final round key, we used the

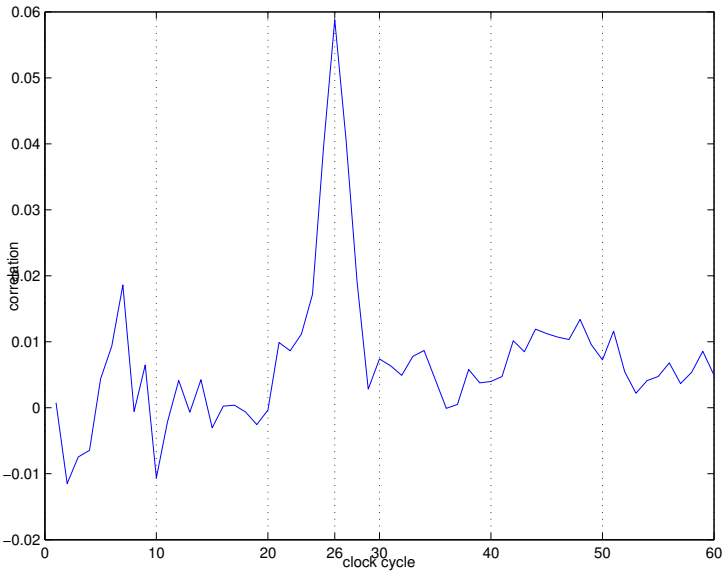


Figure 7.18: Correlation between global predictions for cycle 26 and measurements for the FPGA implementation of the AES ($N = 4096$)

correlation coefficient again. As it is shown in Fig. 7.19, the correct key guess is distinguishable after about 2500 traces. As a consequence, the attack is **practically successful**, *i.e.*, the selected prediction matrix is sufficiently correlated with the real measurements and we can extract the key information. Remark that comparing Fig. 7.17 and 7.19 allows us to evaluate the effect of the measurement phase. Compared with smart cards, the sampling process was made more difficult by the high clock frequency of the AES design (around 100 MHz).

Finally, it is important to note that more key bits may be found using exactly the same set of measurements. The attacker only has to modify the **selected prediction matrix** and target different key bits. The full key can therefore be recovered computing the correlation between the **global consumption matrix** and 16 predictions, each one revealing 8 key bits.

7.2.6 Hypothesis

Looking back at the hypothesis of Section 3.1.1, it is important to evaluate how the work presented in this section could be improved and how representative our results are. To the question "Are power analysis attacks realistic against efficient FPGA implementations of AES?"we may certainly answer "yes". While

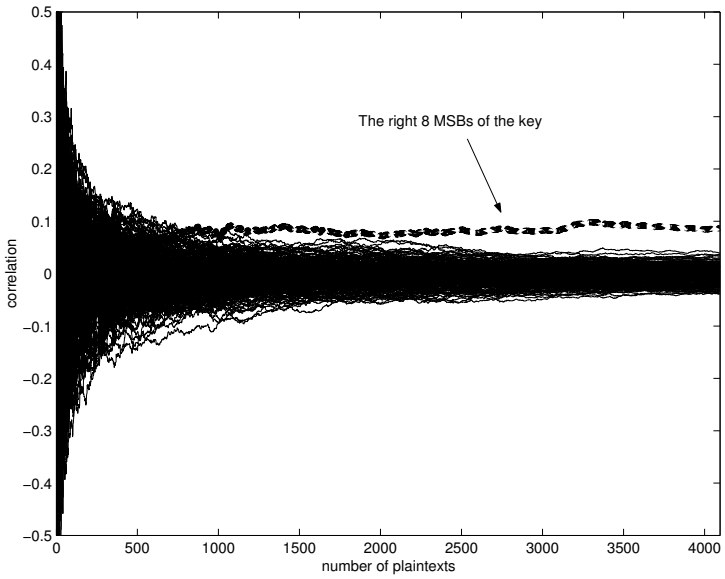


Figure 7.19: Change in the correlation coefficient as a function of the number of plaintexts for a practical attack on the FPGA implementation of the AES by predicting all the *full* registers during the decryptions

attackers usually investigate "toy"implementations for side-channel attacks, we took a real and optimized design with high clock frequencies and evaluated the significance of pipelining techniques in terms of DPA resistance. From an attacker's point of view, we have investigated the simplest possible hypothesis and built a practical attack based on these simple assumptions. However, the question "How to counteract these power analysis attacks?"is still open in different ways. When countermeasures are considered, it is important to note that our measurements can be improved in several ways; moreover, the attack model should be taken into account.

As an illustration, we limited the transition predictions to the registers of the AES design. However, it is clear that registers are not the only leaking parts in FPGAs and transitions in other components could be predicted in order to improve the attack. Similarly, looking back at Eq. (3.3), a more accurate prediction of the FPGA power consumption could be done by evaluating the load capacitance values. A notable feature of FPGAs is that they contain different resources (*e.g.* logic blocks, connections) the power consumption of which differs because of different effective load capacitances. As a consequence, the power consumption of FPGA designs does not only depend on their switching activity but also on the internal resources used. In practice, more accurate

estimations of the most consuming components of an FPGA design can be derived from the delay information that is generated by most implementation tools [152]. As an input delay represents the delay seen by a signal driving that input due to the capacitance along the wire, large (*resp.* small) delay values indicate that the wire has a large (*resp.* small) capacitance. Based on the reports that are automatically generated by the implementation tools, one may expect to recover very accurate information about the signals that are driving high capacitances. The knowledge of the implementation net lists with delay information is therefore relevant; it will allow an attacker to improve the attack.

7.3 Power Analysis of an FPGA Implementation of the DES

We have performed our experiments on the sequential DES implementation of Rouvroy *et al.* in [222] that takes one clock cycle to perform one round. It is represented in Fig. 2.3(a). We used again correlation analysis to implement a DPA attack on the FPGA Implementation of DES.

7.3.1 An Attack Using Simulated Data

This time our target is the four MSBs of the register L that are affected by the six MSBs of the round key 16 [245]. It corresponds to the output bits of S-box $S0$. For simulation purposes, we produce the **global prediction matrix** that contains the number of bit transitions inside all the registers of the design, for all the cycles (see Section 7.1.3). As explained in Section 2.3, there are two register of 32 bits, L and R. That is, if the encryption is performed in 16 clock cycles, we obtain an $N \times 16$ matrix, containing integers between 0 and 64 for N random plaintexts. The number N of measurements for this experiment was 4096.

Then for each of the N encrypted plaintexts, we predict the number of bit transitions inside our target register between rounds 15 and 16 for the 2^6 key guesses. The result of the prediction is an $N \times 2^6$ **selected prediction matrix** containing integers between 0 and 4.

According to the hypothesis of Sect. 3.1.1, these matrices give estimations for the power consumption of the device. Since the same key is used for all the measurements, the power consumption of the key schedule is fixed and may be considered as a DC component that we can neglect as a first approximation.

Thereafter, we compute the correlation coefficient between the $16th$ column of the global consumption matrix (corresponding to $16th$ round targeted) and all the columns of the selected prediction matrix. If the attack is successful, we

expect that only one value, corresponding to the correct key guess, leads to a high correlation coefficient. Figure 7.20 shows that this expectation is met and the correct 6 MSBs of the last round key guess are $\texttt{1E}_{hex} = 30_{dec}$.

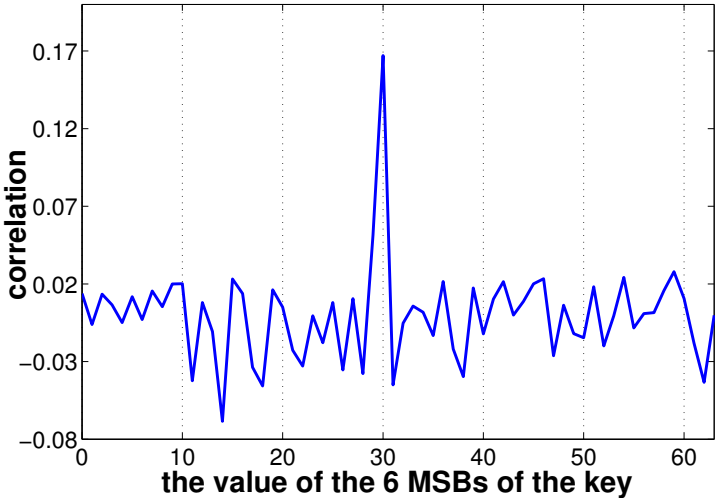


Figure 7.20: Correlation coefficient of all the 2^6 key guesses for simulated attack on the FPGA implementation of the DES ($N = 4096$)

As an attacker would like to learn the minimum number of plaintexts that are necessary to find the key, we have also calculated this correlation coefficient for different values of N : $0 \leq N \leq 2000$. As shown in Fig. 7.21, after approximately 400 plaintexts the right 6 key-bits can be distinguished from a wrong guess. We may therefore say that the attack is **theoretically successful** after about 400 texts.

7.3.2 Correlation Phase:

In the correlation phase, we compute the correlation coefficient between the 16th column of the global consumption matrix (corresponding to 16th round targeted by the prediction phase) and all the columns of the selected prediction matrix (corresponding to all the 2^6 key guesses). If the attack is successful, we expect that only one value, corresponding to the correct key guess, leads to a high correlation coefficient.

Finally, theoretical predictions of the attack can be performed by using the global prediction matrix instead of the global consumption matrix. As the global prediction matrix contains the number of bit switches inside all the registers, it represents a theoretical noise free measurement and may help to

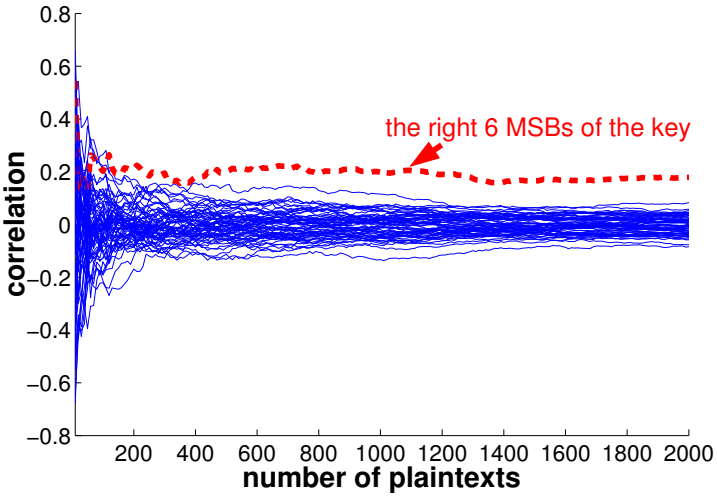


Figure 7.21: Change in correlation coefficient as a function of the number of measurements for simulated attack on the FPGA implementation of the DES

determine the minimum number of texts needed to mount a successful attack, *i.e.*, an attack where the correct key guess leads to the highest correlation coefficient. This is investigated in the next section.

7.3.3 An Attack Using Practical Measurements

When attacking a device in practice, the selected prediction matrix stays unchanged while we replace the global prediction matrix by the measured **global consumption matrix**. We encrypt the same N plaintexts on the FPGA with the same key, as we did in the prediction phase. While the chip is operating, we measure the power consumption for 16 consecutive clock cycles. Then, the power consumption trace of each encryption is averaged 10 times in order to remove the noise from our measurements and we store the maximum value of each encryption cycle so that we produce an $N \times 16$ matrix with the power consumption values for all the texts, cycles. We denote it as the **global consumption matrix**.

In order to identify the correct 6 MSBs of the final round key, we used the correlation coefficient again. As it is shown in Fig. 7.22, the highest correlation occurs when the key guess is $\mathtt{1E}_{hex} = 30_{dec}$. This value corresponds to the correct 6 MSBs of the round key 16. As a consequence, the attack is **practically successful**, *i.e.*, the selected prediction matrix is sufficiently correlated with the real measurements and we can extract the key information. Note that comparing Fig. 7.20 and Fig. 7.22 clearly allows to evaluate the effect of the

noise in our measurements.

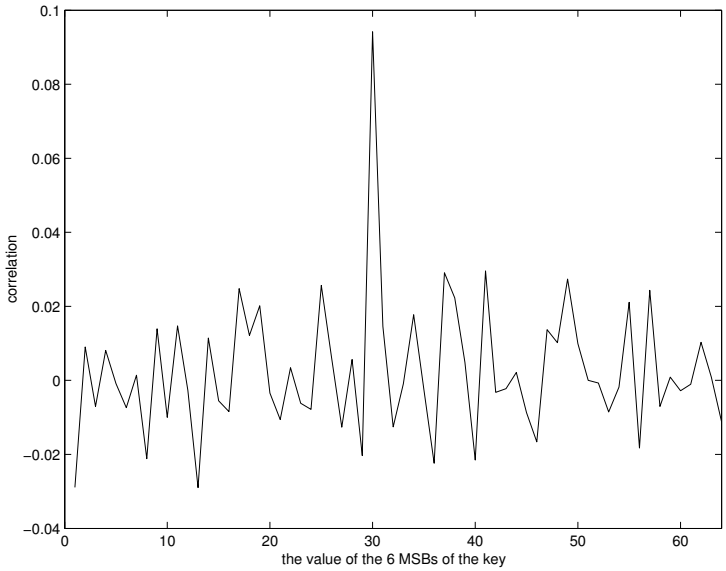


Figure 7.22: Correlation coefficient of all the 2^6 key guesses for the practical attack on the FPGA implementation of the DES ($N = 4096$)

It is important to note that more bits of the final subkey may be found using exactly the same set of measurements. The attacker only has to modify the selected prediction matrix in order to target different key bits. As every subkey consists of 48 bits and the master key of 56 bits, we can easily find the last 8 key bits by exhaustive search. Figure 7.23 shows the correlation coefficient for the predictions of 12 bits of the $16th$ round key. It proves that by using the same set of measurements and by just changing the predictions we can find all the key bits.

7.4 Conclusions

In Section 7.1, we have presented the first public implementation of a DPA attack on an ASIC implementation of the AES. We have shown how to build a reliable measurement setup and how to improve the correlation coefficients, *i.e.*, the signal to noise ratio for our measurements. Due to the results of the simulated attack and the real attack, we conclude that the chip tester, which we used in our measurement setup, introduces a considerable amount of noise in

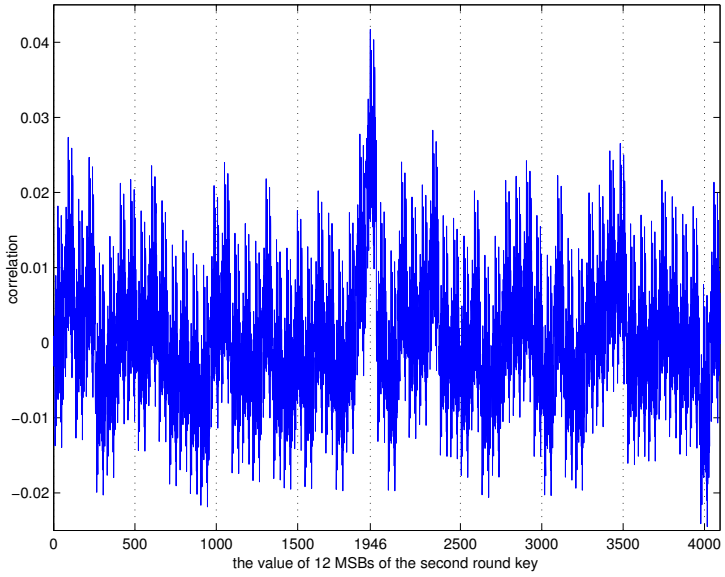


Figure 7.23: Correlation coefficient of all the 2^{12} key guesses for the practical attack on the FPGA implementation of the DES ($N = 4096$)

our measurements (we needed 160 times more measurements in the real attack than in the simulated attack).

Our approach forms a first step to link real and simulated power measurements. This is very important for designers of cryptographic hardware, as it allows them to estimate the vulnerability to power attacks in a very early stage of the design flow. This can bring important security and cost benefits.

In Sections 7.2 and 7.3, we have investigated the first power analysis attacks against FPGA implementations of the AES and the DES, respectively. We have exhibited the effect of pipelining and unrolling techniques in this context. It is first demonstrated that pipelining a loop implementation does not provide an effective countermeasure if an attacker has access to the design details because most of the registers in the pipeline remain predictable. We have also provided a theoretical model allowing the simulation and comparison of the attacks in different contexts. In practice, we have mounted the first successful attack against an efficient FPGA implementation of the AES. Finally, a clear discussion of the hypothesis used to perform power analysis is provided with some proposals for further improvements.

Chapter 8

Conclusions and Open Problems

8.1 Conclusions

In this thesis we focused on area and time efficient hardware design of elliptic curve cryptosystems and on side channel attacks on hardware implementations of cryptographic algorithms. We started the work with hardware implementation of elliptic curve cryptosystems. The efficiency of the implementation of the elliptic curve cryptosystems depends on the efficiency of the implementations of the finite field operations which are the basic operations. Hence first of all we have studied the efficient hardware implementation of the finite field operations over $GF\,(2^m)$ and $GF(p)$. Amongst all the finite field operations the most area and time consuming operation is the modular multiplication. Hence we give special attention to the choice of the most efficient algorithm and implementation it in most efficient way. In order to get rid of the modular reduction step, which is the most expensive operation of a modular multiplication, we use Montgomery modular multiplication (MMM).

We have presented efficient hardware implementations of the Montgomery modular multiplication algorithm over $GF\,(2^m)$ and $GF(p)$ in an FPGA. The designs use systolic array architectures to allow pipelining and to make the clock frequency independent of the operand bit-length. The systolic array consists of cells which are independent from m and p. The critical path of the complete Montgomery modular multiplier is the same as the critical path of a cell of the systolic array. In this way, the clock frequency does not change when m or the bit-length of p is enlarged for security reasons. For the Montgomery modular multiplication over $GF\,(2^m)$ the clock frequency only depends on the

word length ω, as $m = s\omega$, which determines the amount of logic in one cell of the systolic array, s is the digit-length of m. The word length is an input parameter for the implementation of the circuit. The MMM over $GF(2^m)$ is not restricted to field representations using irreducible trinomials: every irreducible polynomial of degree m can be used. In the design of MMM over $GF(p)$ the optimal bound is used which, with some savings in hardware, omits completely all the conditional reduction steps that are known to be vulnerable to the side-channel analysis attacks presented in Chapter 3.

We presented a hardware architecture of a processor for an EC cryptosystem over the finite field $GF(2^m)$ in Section 4.3. Our elliptic curve processor uses the Montgomery modular multiplier over $GF(2^m)$ proposed by Koç and Acar in [130]. The maximum clock frequency is again independent of the bit length m. The total latency of the ECP can be calculated as $(1900 + 14hw)\,T_{MMMC} + 316 + 2hw$, where hw is the Hamming Weight of the key and T_{MMMC} is the latency of one MMM. $T_{MMMC} = m$ for $w = 1$ and $T_{MMMC} = 3\frac{m}{w}$ for $w > 1$. One elliptic curve point multiplication (ECPM) requires 3.810 ms at 47 MHz.

Because there is no previous work which uses MMM and all other designs use different platforms, it is hard to compare the area of the current work with other implementations. The time needed for one ECPM with our design is approximately the same as the result reported in [5], smaller than the results reported in [249, 74, 75, 140, 63, 84, 116], but larger than the ones reported in [192] and [60].

We presented a FPGA implementation of an elliptic curve processor [202, 195, 197, 16, 198]. The processor consists of special operational blocks for Montgomery Modular Multiplication, modular addition/subtraction, EC Point doubling/addition, modular multiplicative inversion, EC point multiplication, projective to affine coordinates conversion and Montgomery to normal representation conversion. Our processor can be programmed by the host to execute these operations in any order. It is possible to use the proposed processor not only for ECC, but also for any system for which modular arithmetic operations are essential, such as the RSA cryptosystem.

The basic operations are MMM and MAS. The other blocks include FSMs, which control the execution of these operations. The critical path depends only on the critical path of the circuits for MMM and MAS. The architecture of these blocks is designed to ensure a short critical path to allow for high clock frequencies which are independent from the bit-length of the EC parameters.

The proposed processor has been implemented on a Xilinx V800-HQ-240-4 Virtex FPGA by taking the bit-length of EC parameters N and the bit-length of key, l as 160. According to the implementation results, the number of flip-flops and 4 input LUTs are equal to 11 192 and 14 393, respectively. This is equivalent to 175 952 gates. The minimum clock period is 40.550 ns (maximum

clock frequency: 24.661 MHz). LUTs are lookup-tables that are used as RAMs or 4-input gates.

The only existing previous work done on FPGA is from Orlando and Paar [194]. Orlando and Paar use a multiplier which is also based on the MMM algorithm but it is a generalized version with quotient pipelining introduced by Orup in [203]. We use the basic MMM algorithm from which we only exclude the modular reduction as a result of the bound adjustment. In this way no precomputation is required, which results in a substantial memory reduction. More importantly, this property also facilitates modular exponentiation as temporary results of multiplications can be fed back directly without any modular subtraction. Their multiplier has a semi-systolic architecture while the multiplier presented here is fully systolic. This results in an increased flexibility which is unrelated to any specific parameter choice. Orlando and Paar also used an adaptation of a fixed base exponentiation method as introduced by Brickell *et al.* in [29]. This algorithm is claimed to be four times faster than the standard double-and-add algorithm which we use. However, it involves a known point calculation which is a limiting factor with respect to various applications of ECC.

We have designed, implemented and simulated the circuits with commercial software tools, such as Synopsys, Cadence, etc.; it would be desirable to build and test real implementations. Because there are many choices for architectures that we would like to try and also we would like to test our circuits for debugging purposes before the final product comes out, we should have a prototyping environment. Because of the reconfigurability of the FPGAs, they are the best choice for this requirement. We need a development board which the FPGA we choose is on, we should be able to configure the FPGA and establish a communication between the FPGA and a PC via this board. Another issue designers should take care of is the resistance against side-channel attacks. The algorithms can be mathematically secure, but if they are implemented in a way that their side-channels give information about the secret data inside then the expected security level from the algorithm will never be achieved. The side-channels of an implementation that have been exploited so far are timing, power consumption, electromagnetic radiation and sound. As a hardware designer of cryptographic algorithms it is natural that we are interested in understanding the resistance of our implementations against side-channel analysis attacks. Again because the FPGAs give the possibility to try as many circuits as we wish, they serve as a good environment for attacking on our circuits and improving them in order to have more resistant circuits. The most popular side-channel information that the researchers use is the dynamic power consumption of the cryptographic hardware, while executing the cryptographic algorithm with a set of known inputs. There is a main problem in order to conduct a power analysis attack; we have to interrupt the power source wires of the FPGA. The commercial development boards provide all the

requirements mentioned above, but they do not give the possibility to reach the power sources of the FPGA. The only wire we can use to make the power consumption measurements is the power source wire of the board. If we use this wire then we will measure the power consumption of all the devices on the board, such as the microcontroller, memory, leds, etc. Hence we have designed and produce our own development board with special features to conduct the side-channel attacks easily.

We can prove that our FPGA leaks a significant amount of information about its internal computations through the power source wires. We have even provided evidence that the power consumption characteristics are comparable with the power consumption characteristics of ordinary application specific integrated circuits (ASICs). Therefore, it is possible to draw conclusions about the vulnerability of a certain circuit by performing power-analysis attacks on an FPGA-implementation. Since programming an FPGA is considerably less expensive than manufacturing an ASIC, assessing vulnerability of a device w.r.t. power-analysis attacks is less expensive on our platform. Consequently, our approach describes the first inexpensive and efficient way to conduct power-analysis attacks on a real implementation (*i.e.*, not on a software simulation) of a circuit in a very early stage of the design.

After developing the suitable environment for the side-channel analysis attacks, we have implemented timing, power and electromagnetic analysis attacks on our FPGA implementations of elliptic curve cryptosystems over $GF(p)$. We concluded that our initial design was vulnerable to simple attacks and by using only the timing information it is possible to find the Hamming weight of the key. More drastically by using simple power and electromagnetic analysis attacks it was possible to find all the key bits. Next we improved our circuits such that they are resistant against timing and simple power and electromagnetic analysis attacks. We showed that this improved design is still vulnerable to differential attacks.

We have conducted differential power and electromagnetic analysis attacks on our improved FPGA implementation of elliptic curve processor. We use two well known techniques for DPA and DEMA; correlation analysis and distance of mean test. We conclude that the correlation analysis reveals the right key bit by using six times less measurements than the distance of mean test for both DPA and DEMA. Our electromagnetic radiation measurements have to be doubled with respect to the power consumption measurements in order to find the right key bit. There are several reasons for this. We use a very simple handmade antenna shown in Fig. 6.13. Because the diameter of the antenna is large and it is not isolated from any effect by any protection, the antenna collects all the signals in the air. We use the same prediction data for both power consumption and electromagnetic radiation, which is the number of transitions in the target registers. This is a correct model for power consumption as it is related to the current flow to the load capacitances, but the electromagnetic radiation is

also affected by the internal direction of the current. Hence the predictions for electromagnetic radiation should be improved by taking into account of the position of the antenna on the FPGA and the relative direction of the current flows.

Having our custom development FPGA board gives us the opportunity to conduct power analysis attacks on the FPGA implementations of the AES and the DES from UCL Crypto Group. In order to compare the resistance against power analysis attack of different architectures and different technologies, we have conducted a power analysis attack on an ASIC implementation of the AES from ETH Zürich. We have presented the first public implementation of a DPA attack on an ASIC implementation of the AES. We have shown how to build a reliable measurement setup and how to improve the correlation coefficients, *i.e.*, the signal to noise ratio for our measurements. Due to the results of the simulated attack and the real attack, we conclude that the chip tester, which we used in our measurement setup, introduces a considerable amount of noise in our measurements (we needed 160 times more measurements in the real attack than in the simulated attack).

Our approach forms a first step to link real and simulated power measurements. This is very important for designers of cryptographic hardware, as it allows them to estimate the vulnerability to power attacks in a very early stage of the design flow. This can bring important security and cost benefits.

We have investigated the first power analysis attacks against FPGA implementations of AES and DES, respectively. We have studied the effect of pipelining and unrolling techniques in this context. First we have demonstrated that pipelining a loop implementation does not provide an effective countermeasure if an attacker has access to the design details because most of the registers in the pipeline remain predictable. We have also provided a theoretical model that allows to simulate and compare the attacks in different contexts. In practice, we have mounted the first successful attack against an efficient FPGA implementation of AES. Finally, a clear discussion of the hypothesis used to perform power analysis is provided with some proposals for further improvements.

8.2 Topics for New Research

- So far we use correlation analysis and distance of mean test for the DPA and DEMA attacks. Other methods [42] such as a maximum likelihood test should be used in order to decrease the needed number of measurements.
- Besides carefully exploring all available side-channel information an attacker can also focus on a combination of two or more side-channels. Agrawal *et al.* define these so-called multi-channel attacks in which the

side-channels are not necessarily of a different kind [8]. We can measure different side-channels of the FPGA simultaneously on our measurement setup, hence combining these information can decrease the needed number of measurements.

- A step would be to implement and test the countermeasures against differential power and electromagnetic analysis attacks. The suggestions for the countermeasures are given in Section 3.3.2.4.
- Fault attacks can be also conducted on our measurement setup by modifying the configuration file of the FPGA.
- We use a very simple handmade antenna shown in Fig. 6.13 for EMA attacks. Because the diameter of the antenna is large and it is not isolated from any effect by any protection, the antenna collects all the signals in the air. The position, dimension of the antenna should be analytically explored.
- We use the same prediction data for both power consumption and electromagnetic radiation, which is the number of transitions in the target registers. This is a correct model for power consumption as it is related to the current flow to the load capacitances, but the electromagnetic radiation is also effected by the direction of the current internally. So the predictions for electromagnetic radiation should be improved by taking into account of the position of the antenna on the FPGA and the relative direction of the current flows.
- It can be possible to decrease the number of needed measurements by combining the classical cryptanalysis and side-channel analysis attacks. There are a few publications on this work which are all implemented on software implementations [232, 233].

Bibliography

[1] G. B. Agnew, T. Beth, R. C. Mullin, and S. A. Vanstone. Arithmetic operations in $GF(2^m)$. *Journal of Cryptology*, 6:3–13, 1993.

[2] G. B. Agnew, R. C. Mullin, I. M. Onyszchuk, and S. A. Vanstone. An implementation for a fast public key cryptosystem. *Journal of Cryptology*, 3(2):63–79, 1991.

[3] G. B. Agnew, R. C. Mullin, and S. A. Vanstone. A fast elliptic curve cryptosystem. In J.-J. Quisquater and J. Vandewalle, editors, *Advances in Cryptology: EUROCRYPT'89*, volume 434 of *Lecture Notes in Computer Science*, pages 706–708. Springer-Verlag, 1989.

[4] G. B. Agnew, R. C. Mullin, and S. A. Vanstone. On the development of a fast elliptic curve cryptosystem. In R. A. Rueppel, editor, *Advances in Cryptology: EUROCRYPT'92*, volume 658 of *Lecture Notes in Computer Science*, pages 482–487. Springer-Verlag, 1992.

[5] G. B. Agnew, R. C. Mullin, and S. A. Vanstone. An implementation of elliptic curve cryptosystem over $F_{2^{155}}$. *IEEE Journal on Selected Areas in Communications*, 11(5):804–813, June 1993.

[6] D. Agrawal, B. Archambeault, S. Chari, J. R. Rao, and P. Rohatgi. Advances in side-channel cryptanalysis. *RSA Laboratories Cryptobytes*, 6(1):20–32, Spring 2003.

[7] D. Agrawal, B. Archambeault, J. R. Rao, and P. Rohatgi. The EM side-channel(s): Attacks and assessment methodologies. In B. S. Kaliski Jr., Ç. K. Koç, and C. Paar, editors, *Cryptographic Hardware and Embedded Systems (CHES)*, volume 2523 of *Lecture Notes in Computer Science*, pages 29–45. Springer-Verlag, 2002.

[8] D. Agrawal, J. R. Rao, and P. Rohatgi. Multi-channel attacks. In C. Walter, Ç. K. Koç, and C. Paar, editors, *Cryptographic Hardware and Embedded Systems (CHES)*, volume 2779 of *Lecture Notes in Computer Science*, pages 2–16. Springer-Verlag, 2003.

[9] G. C. Ahlquist, B. E. Nelson, and M. Rice. Optimal finite field multipliers for FPGAs. In P. Lysaght, J. Irvine, and R. W. Hartenstein, editors, *Field-Programmable Logic and Applications (FPL)*, volume 1673 of *Lecture Notes in Computer Science*, pages 51–60. Springer-Verlag, 1999.

[10] M.-L. Akkar and C. Giraud. An implementation of DES and AES, secure against some attacks. In Ç. K. Koç, D. Naccache, and C. Paar, editors, *Cryptographic Hardware and Embedded Systems (CHES)*, volume 2162 of *Lecture Notes in Computer Science*, pages 309–318. Springer-Verlag, 2001.

[11] R. Anderson and M. Kuhn. Tamper resistance – a cautionary note. In D. Tygar, editor, *Proceedings of the 2nd USENIX Workshop on Electronic Commerce*, pages 1–11, Oakland, CA, USA, November 18-21 1996.

[12] ANSI X9.62. *Public key cryptography for the finantial services industry: the Elliptic Curve Digital Signature Algorithm (ECDSA)*. American National Standard Institution, January 1999. `http:www.ansi.org`.

[13] ANSI X9.63. *Public key cryptography for the financial services industry: key agreement and key transport using elliptic curve cryptography*. American National Standard Institution, January 1999. `http:www.ansi.org`.

[14] Y. Asano, T. Itoh, and S. Tsujii. Generalized fast algorithm for computing multiplicative inverses in $GF(2^m)$. *Electronics Letters*, 25(10):664–665, March 1989.

[15] J. Axelson. *Parallel Port Complete: Programming, Interfacing, and Using the PC's Parallel Printer Port*. Lakeview Research, Madison, WI 53704, 2000.

[16] L. Batina, G. Bruin-Muurling, and S. B. Örs. Flexible hardware design for RSA and elliptic curve cryptosystems. In T. Okamoto, editor, *Topics in Cryptology – CT-RSA*, volume 2964 of *Lecture Notes in Computer Science*, pages 250–263. Springer-Verlag, 2004.

[17] L. Batina, N. Mentens, S. B. Örs, and B. Preneel. Serial multiplier architectures over $GF(2^n)$ for elliptic curve cryptosystems. In *Proceedings of the 12th IEEE Mediterranean Electrotechnical Conference (MELECON)*, 2004.

[18] L. Batina and G. Muurling. Montgomery in practice: How to do it more efficiently in hardware. In B. Preneel, editor, *Topics in Cryptology - CT-RSA*, volume 2271 of *Lecture Notes in Computer Science*, pages 40–52. Springer-Verlag, 2002.

[19] L. Batina, S. B. Örs, B. Preneel, and J. Vandewalle. Hardware architectures for public key cryptography. *Elsevier Science Integration the VLSI Journal*, 34(1-2):1–64, 2003.

[20] M. Bednara, M. Daldrup, J. Teich, J. von zur Gathen, and J. Shokrollahi. Tradeoff analysis of FPGA based elliptic curve cryptosystems. In *Proceedings of The IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 797–800, Scottsdale, Arizona, USA, May 26-29 2002.

[21] L. Benini, A. Macii, E. Macii, E. Omerbegovic, M. Poncino, and F. Pro. Energy-aware design techniques for differential power analysis protection. In *Proceedings of the 40th Design Automation Conference (DAC)*, Anaheim, CA, USA, June 2-6 2003.

[22] E. Biham and A. Shamir. Power analysis of the key scheduling of the AES candidates. In *Proceedings of the Second Advanced Encryption Standard (AES) Candidate Conference*, Rome, Italy, 1999.

[23] O. Billet and M. Joye. The Jacobi model of an elliptic curve and side-channel analysis. Cryptology ePrint Archive: Report 2002/125, 2002.

[24] I. Blake, G. Seroussi, and N. P. Smart. *Elliptic Curves in Cryptography*. London Mathematical Society Lecture Note Series. Cambridge University Press, 1999.

[25] T. Blum and C. Paar. High-radix Montgomery modular exponentiation on reconfigurable hardware. *IEEE Transactions on Computers*, 50(7):759–764, July 2001.

[26] D. Boneh, R. A. DeMillo, and R. J. Lipton. On the importance of checking cryptographic protocols for faults (extended abstract). In W. Fumy, editor, *Advances in Cryptology: EUROCRYPT'97*, volume 1233 of *Lecture Notes in Computer Science*, pages 37–51. Springer-Verlag, 1997.

[27] J. Borst. *Block Ciphers: Design, Analysis and Side-Channel Analysis*. PhD thesis, K.U.Leuven, September 2001.

[28] A. Bosselaers, R. Govaerts, and J. Vandewalle. Comparison of three modular reduction functions. In E. F. Brickell, editor, *Advances in Cryptology: CRYPTO'92*, volume 740 of *Lecture Notes in Computer Science*, pages 175–186. Springer-Verlag, 1993.

[29] E. F. Brickell, D. M. Gordon, K. S. McCurley, and D. B. Wilson. Fast exponentiation with precomputation: Algorithms and lower bound. In R. A. Rueppel, editor, *Advances in Cryptology: EUROCRYPT'92*, volume 658 of *Lecture Notes in Computer Science*, pages 200–207. Springer-Verlag, 1992.

[30] E. Brier, C. Clavier, and F. Olivier. Optimal statistical power analysis. IACR e-print archive 2003/152, 2003.

[31] E. Brier and M. Joye. Weierstrass elliptic curves and side-channel attacks. In D. Naccache and P. Paillier, editors, *Practice and Theory in Public Key Cryptosystems (PKC)*, volume 2274 of *Lecture Notes in Computer Science*, pages 335–345. Springer-Verlag, 2002.

[32] D. Brumley and D. Boneh. Remote timing attacks are practical. In *Proceedings of the 12th Usenix Security Symposium*, San Antonio, Texas, USA, June 9-14 2003.

[33] H. Brunner, A. Curiger, and M. Hofstetter. On computing multiplicative inverses in $GF(2^m)$. *IEEE Transactions on Computers*, 42(8):1010–1014, August 1993.

[34] P. Buysschaert and E. De Mulder. Elektromagnetische analyse (EMA) van een FPGA implementatie van een elliptische krommen cryptosysteem. Master's thesis, Katholieke Universiteit Leuven, Departement Elektrotechniek – ESAT, Kasteelpark Arenberg 10, B 3001 Heverlee, Belgium, May 2004.

[35] I. J. Calvo and M. Torres. Complexity of the inversion in $GF(2^m)$. *Electronics Letters*, 33(3):194–195, January 1997.

[36] V. Carlier, H. Chabanne, E. Dottax, and H. Pelletier. Electromagnetic side channels of an FPGA implementation of AES. Cryptology ePrint Archive-2004/145, 2004. `http://eprint.iacr.org/`.

[37] J. Cathalo, F. Koeune, and J.-J. Quisquater. A new type of timing attack: Application to GPS. In C. Walter, Ç. K. Koç, and C. Paar, editors, *Cryptographic Hardware and Embedded Systems (CHES)*, volume 2779 of *Lecture Notes in Computer Science*, pages 291–303. Springer-Verlag, 2003.

[38] S. Chari, C. S. Jutla, J. R. Rao, and P. Rohatgi. A cautionary note regarding evaluation of AES candidates on smart cards. In *Proceedings of the Second Advanced Encryption Standard (AES) Candidate Conference*, Rome, Italy, February 1999.

[39] S. Chari, C. S. Jutla, J. R. Rao, and P. Rohatgi. Towards sound approaches to counteract power-analysis attacks. In M. Wiener, editor, *Advances in Cryptology: CRYPTO'99*, volume 1666 of *Lecture Notes in Computer Science*, pages 398–412. Springer-Verlag, 1999.

[40] S. Chari, J. R. Rao, and P. Rohatgi. Template attacks. In B. S. Kaliski Jr., Ç. K. Koç, and C. Paar, editors, *Cryptographic Hardware and Embedded Systems (CHES)*, volume 2523 of *Lecture Notes in Computer Science*, pages 172–186. Springer-Verlag, 2002.

[41] M. Ciet, J.-J. Quisquater, and F. Sica. Preventing differential analysis in GLV elliptic curve scalar multiplication. In B. S. Kaliski Jr., Ç. K. Koç, and C. Paar, editors, *Cryptographic Hardware and Embedded Systems (CHES)*, volume 2535 of *Lecture Notes in Computer Science*, pages 540–550. Springer-Verlag, 2002.

[42] G. M. Clarke and D. Cooke. *A Basic Course in Statistics*. Arnold London, 4th edition, 1998.

[43] H. Cohen, A. Miyaji, and T. Ono. Efficient elliptic curve exponentiation using mixed coordinates. In K. Ohta and D. Pei, editors, *ASIACRYPT*, volume 1514 of *Lecture Notes in Computer Science*, pages 51–65. Springer-Verlag, 1998.

[44] J.-S. Coron. Resistance against differential power analysis for elliptic curve cryptosystems. In Ç. K. Koç and C. Paar, editors, *Cryptographic Hardware and Embedded Systems (CHES)*, volume 1717 of *Lecture Notes in Computer Science*, pages 292–302. Springer-Verlag, 1999.

[45] J.-S. Coron and L. Goubin. On Boolean and arithmetic masking against differential power analysis. In Ç. K. Koç and C. Paar, editors, *Cryptographic Hardware and Embedded Systems (CHES)*, volume 1965 of *Lecture Notes in Computer Science*, pages 231–237. Springer-Verlag, 2000.

[46] L. W. Couch. *Digital and Analog Communication Systems*. Prentice Hall, 1997.

[47] J. Daemen, M. Peeters, and Gilles Van Assche. Bitslice ciphers and power analysis attacks. In B. Schneier, editor, *Fast Software Encryption (FSE)*, volume 1978 of *Lecture Notes in Computer Science*, pages 134–149. Springer-Verlag, 2000.

[48] J. Daemen and V. Rijmen. The block cipher Rijndael. In J.-J. Quisquater and B. Schneier, editors, *Smart Card Research and Applications (CARDIS)*, volume 1820 of *Lecture Notes in Computer Science*, pages 288–296. Springer-Verlag, 1998.

[49] J. Daemen and V. Rijmen. Resistance against implementation attacks: A comparative study of the AES proposals. In *Proceedings of the Second Advanced Encryption Standard (AES) Candidate Conference*, March 1999. `http://csrc.nist.gov/CryptoToolkit/aes/round1/conf2/papers/daemen.pdf`.

[50] J. Daemen and V. Rijmen. Rijndael for AES. In *Proceedings of The Third Advanced Encryption Standard Candidate Conference*, pages 343–348, New York, NY, USA, April 13-14 2000. National Institute of Standards and Technology.

[51] J. Daemen and V. Rijmen. *The Design of Rijndael. AES – The Advanced Encryption Standard*. Springer-Verlag, 2001.

[52] J. Daemen and V. Rijmen. Rijndael, the advanced encryption standard. *Dr. Dobb's Journal*, 26(3):137–139, March 2001.

[53] E. De Win and B. Preneel. Elliptic curve public-key cryptosystems – an introduction. In B. Preneel and V. Rijmen, editors, *State of the Art in Applied Cryptography*, volume 1528 of *Lecture Notes in Computer Science*, pages 132–142. Springer-Verlag, 1997.

[54] B. den Boer, K. Lemke, and G. Wicke. A DPA attack against the modular reduction within a CRT implementation of RSA. In B. S. Kaliski Jr., Ç. K. Koç, and C. Paar, editors, *Cryptographic Hardware and Embedded Systems (CHES)*, volume 2535 of *Lecture Notes in Computer Science*, pages 228–243. Springer-Verlag, 2002.

[55] J. F. Dhem. *Design of an efficient public-key cryptographic library for RISC-based smart cards*. PhD thesis, Universite Catholiqué de Louvain, UCL Crypto Group, Laboratoire de microelectronique (DICE), May 1998.

[56] J. F. Dhem, F. Koeune, P.A. Leroux, P. Mestre, J.-J. Quisquater, and J. L. Willems. A practical implementation of the timing attack. Technical Report CG-1998/1, UCL Crypto Group, Universite Catholique de Louvain, Belgium, 1998.

[57] M. Diab and A. Poli. New bit-serial systolic multiplier for $GF(2^m)$ using irreducible trinomials. *Electronics Letters*, 27(13):1183–1184, June 1991.

[58] W. Diffie and M. E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22:644–654, 1976.

[59] G. Drolet. A new representation of elements of finite fields $GF(2^m)$ yielding small complexity arithmetic circuits. *IEEE Transactions on Computers*, 47(9):938–946, September 1998.

[60] H. Eberle, N. Gura, and S. Chang-Shantz. A cryptographic processor for arbitrary elliptic curves over $GF(2^m)$. In *IEEE 14th International Conference on Application-Specific Systems, Architectures and Processors (ASAP)*, pages 444–454, The Hague, The Netherlands, June 24-26 2003.

[61] T. ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In G. R. Blakley and D. Chaum, editors, *Advances in Cryptology: CRYPTO'84*, volume 196 of *Lecture Notes in Computer Science*, pages 10–18. Springer-Verlag, 1985.

[62] T. ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Transactions on Information Theory*, 31:469–472, 1985.

[63] M. Ernst, S. Klupsch, O. Hauck, and S. A. Huss. Rapid prototyping for hardware accelerated elliptic curve public-key cryptosystems. In *Proceedings of 12th IEEE Workshop on Rapid System Prototyping*, pages 24–31, Monterey, CA, June 2001.

[64] P. Fahn and P. Pearson. IPA: A new class of power attacks. In Ç. K. Koç and C. Paar, editors, *Cryptographic Hardware and Embedded Systems (CHES)*, volume 1717 of *Lecture Notes in Computer Science*, pages 173–186. Springer-Verlag, 1999.

[65] G.-L. Feng. A VLSI architecture for fast inversion in $GF(2^m)$. *IEEE Transactions on Computers*, 38(10):1383–1386, October 1989.

[66] S. T. J. Fenn, M. Benaissa, and D. Taylor. Fast normal basis inversion in $GF(2^m)$. *Electronics Letters*, 32(17):1566–1567, August 1996.

[67] S. T. J. Fenn, M. Benaissa, and D. Taylor. $GF(2^m)$ multiplication and division over the dual basis. *IEEE Transactions on Computers*, 45(3):319–327, March 1996.

[68] W. Fischer, C. Giraud, E. W. Knudsen, and J. P. Seifert. Parallel scalar multiplication on general elliptic curves over F_p hedged against non-differential side-channel attacks. Cryptology ePrint Archive: Report 2002/007, 2002.

[69] J. J. A. Fournier, S. Moore, H. Li, R. Mullins, and G. Taylor. Security evaluation of asynchronous circuits. In C. Walter, Ç. K. Koç, and C. Paar, editors, *Cryptographic Hardware and Embedded Systems (CHES)*, volume 2779 of *Lecture Notes in Computer Science*, pages 137–151. Springer-Verlag, 2003.

[70] G. Frey. How to disguise an elliptic curve. Talk at Waterloo workshop on the ECDLP, 1998. `http://www.cacr.math.uwaterloo.ca/conferences/1998/ecc98/slides.html`.

[71] S. D. Galbraith, F. Hess, and N. P. Smart. Extending the GHS weil descent attack. In L. Knudsen, editor, *Advances in Cryptology: EUROCRYPT'02*, volume 2332 of *Lecture Notes in Computer Science*, pages 29–44. Springer-Verlag, 2002.

[72] K. Gandolfi, C. Mourtel, and F. Olivier. Electromagnetic analysis: Concrete results. In Ç. K. Koç, D. Naccache, and C. Paar, editors, *Cryptographic Hardware and Embedded Systems (CHES)*, volume 2162 of *Lecture Notes in Computer Science*, pages 255–265. Springer-Verlag, 2001.

[73] L. Gao and K. K. Parhi. Custom VLSI design of efficient low latency and low power finite field multiplier for Reed-Solomon codec. In *Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS)*, volume 4, pages 574–577, 2001.

[74] L. Gao, S. Shrivastava, H. Lee, and G. E. Sobelman. A compact fast variable key size elliptic curve cryptosystem coprocessor. In *Proceedings of the Seventh Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pages 304–305, 1999.

[75] L. Gao, S. Shrivastava, and G. E. Sobelman. Elliptic curve scalar multiplier design using FPGAs. In Ç. K. Koç and C. Paar, editors, *Cryptographic Hardware and Embedded Systems (CHES)*, volume 1717 of *Lecture Notes in Computer Science*, pages 257–305. Springer-Verlag, 1999.

[76] L. Gao and G. E. Sobelman. Improved VLSI designs for multiplication and inversion in $GF(2^m)$. In *Proceedings of the 13th Annual Int. ASIC/SOC Conference*, pages 97–101, Washington DC., Sept. 13-16 2000.

[77] P. Gaudry, F. Hess, and N. P. Smart. Constructive and destructive facets of Weil descent on elliptic curves. *Journal of Cryptology*, 15:19–46, 2002.

[78] C. Gehrmann and M. Näslund. ECRYPT yearly report on algorithms and key sizes (rev. 2004). Technical report, ECRYPT NoE, January 13 2005.

[79] M. Girault. Self-certified public keys. In I. B. Damgård, editor, *Advances in Cryptology: EUROCRYPT'90*, volume 473 of *Lecture Notes in Computer Science*, page 490497. Springer-Verlag, 1990.

[80] J. D. Golic. DeKaRT: A new paradigm for key-dependent reversible circuits. In C. Walter, Ç. K. Koç, and C. Paar, editors, *Cryptographic Hardware and Embedded Systems (CHES)*, volume 2779 of *Lecture Notes in Computer Science*, pages 98–112. Springer-Verlag, 2003.

[81] J. D. Golic and C. Tymen. Multiplicative masking and power anaylsis of AES. In B. S. Kaliski Jr., Ç. K. Koç, and C. Paar, editors, *Cryptographic Hardware and Embedded Systems (CHES)*, volume 2535 of *Lecture Notes in Computer Science*, pages 198–212. Springer-Verlag, 2002.

[82] D. Gollman. Equally spaced polynomials, dual bases, and multiplication in F_{2^n}. *IEEE Transactions on Computers*, 51(5):588–591, May 2002.

[83] M. Gomulkiewicz and M. Kutylowski. Hamming weight attacks on cryptographic hardware - breaking masking defenses. In D. Gollmann, G. Karjoth, and M. Waidner, editors, *European Symposium on Research in Computer Security (ESORICS)*, volume 2502 of *Lecture Notes in Computer Science*, pages 90–103. Springer-Verlag, 2002.

[84] J. Goodman and A. P. Chandrakasan. An energy-efficient reconfigurable public-key cryptography processor. *IEEE Journal of Solid-State Circuits*, 36(11):1808–1820, November 2001.

[85] L. Goubin. A sound method for switching between Boolean and arithmetic masking. In Ç. K. Koç, D. Naccache, and C. Paar, editors, *Cryptographic Hardware and Embedded Systems (CHES)*, volume 2162 of *Lecture Notes in Computer Science*, pages 3–15. Springer-Verlag, 2001.

[86] L. Goubin and J. Patari. DES and differential power analysis the "duplication" method. In Ç. K. Koç and C. Paar, editors, *Cryptographic Hardware and Embedded Systems (CHES)*, volume 1717 of *Lecture Notes in Computer Science*, pages 158–172. Springer-Verlag, 1999.

[87] J. Grosschädl. A bit-serial unified multiplier architecture for finite fields $GF(p)$ and $GF(2^n)$. In Ç. K. Koç, D. Naccache, and C. Paar, editors, *Cryptographic Hardware and Embedded Systems (CHES)*, volume 2162 of *Lecture Notes in Computer Science*, pages 206–223. Springer-Verlag, 2001.

[88] J. Guajardo and C. Paar. Efficient algorithms for elliptic curve cryptosystems. In B. S. Kaliski Jr., editor, *Advances in Cryptology: CRYPTO'97*, volume 1294 of *Lecture Notes in Computer Science*, pages 342–356. Springer-Verlag, 1997.

[89] J.-H. Guo and C.-L. Wang. Systolic array implementation of Euclid's algorithm for inversion and division in $GF(2^m)$. *IEEE Transactions on Computers*, 47(10):1161–1167, October 1998.

[90] F. K. Gurkaynak, A. Burg, N. Felber, W. Fichtner, D. Gasser, F. Hug, and H. Kaeslin. A 2 GB/s balanced AES crypto-chip implementation. In D. Garrett, C. Zukowski, and J. Lach, editors, *Proceedings of the Great Lakes Symposium on VLSI (GLSVLSI)*, Boston, MA, USA, April 26-28 2004. ACM Press.

[91] A. Gutub, A. F. Tenca, and Ç. K. Koç. Scalable VLSI architecture for $GF(p)$ Montgomery modular inverse computation. In *Proceedings of IEEE Computer Society Annual symposium on VLSI*, pages 53–58, April 2002.

[92] J. C. Ha and S. J. Moon. Randomized signed-scalar multiplication of ECC to resist power attacks. In B. S. Kaliski Jr., Ç. K. Koç, and C. Paar, editors, *Cryptographic Hardware and Embedded Systems (CHES)*, volume 2523 of *Lecture Notes in Computer Science*, pages 129–143. Springer-Verlag, 2002.

[93] G. Hachez, F. Koeune, and J.-J. Quisquater. Timing attack: what can be achieved by a powerful adversary? In A. Barbé, E. C. van der Meulen, and P. Vanroose, editors, *Proceedings of the 20th symposium on Information Theory in the Benelux*, pages 63–70, May 1999.

[94] G. Hachez and J.-J. Quisquater. Montgomery exponentiation with no final subtractions: Improved results. In Ç. K. Koç and C. Paar, editors, *Cryptographic Hardware and Embedded Systems (CHES)*, volume 1965 of *Lecture Notes in Computer Science*, pages 293–301, 2000.

[95] A. Halbutogullari and Ç. K. Koç. Mastrovito multiplier for general irreducible polynomials. *IEEE Transactions on Computers*, 49(5):503–518, May 2000.

[96] H. Handschuh and H. M. Heys. A timing attack on RC5. In S. E. Tavares and H. Meijer, editors, *Selected Areas in Cryptography (SAC)*, volume 1556 of *Lecture Notes in Computer Science*, pages 306–318. Springer-Verlag, 1998.

[97] M. A. Hasan. Look-up table-based large finite field multiplication in memory constrained cryptosystems. *IEEE Transactions on Computers*, 49(7):749–758, July 2000.

[98] M. A. Hasan. Power analysis attacks and algorithmic approaches to their countermeasures for Koblitz curve cryptosystems. In Ç. K. Koç and C. Paar, editors, *Cryptograpic Hardware and Embedded Systems (CHES)*, volume 1965 of *Lecture Notes in Computer Science*, pages 93–108. Springer-Verlag, 2000.

[99] M. A. Hasan. Power analysis attacks and algorithmic approaches to their countermeasures for Koblitz curve cryptosystems. *IEEE Transactions on Computers*, 50(10):1071–1083, October 2001.

[100] M. A. Hasan and V. K. Bhargava. Bit-serial systolic divider and multiplier for finite fields $GF(2^m)$. *IEEE Transactions on Computers*, 41(8):972–980, August 1992.

[101] M. A. Hasan, M. Wang, and V. K. Bhargava. Modular construction of low complexity parallel multipliers for a class of finite fields $GF(2^m)$. *IEEE Transactions on Computers*, 41(8):962–971, August 1992.

[102] M. A. Hasan and A. G. Wassal. VLSI algorithms, architectures, and implementation of a versatile $GF(2^m)$ processor. *IEEE Transactions on Computers*, 49(10):1064–1072, October 2000.

[103] O. Hauck, A. Katoch, and S. A. Huss. VLSI system design using asynchronous wave pipelines: a 0.35 μm CMOS 1.5 GHz elliptic curve public key cryptosystem chip. In *Proceedings of Sixth International Symposium on Advanced Research in Asynchronous Circuits and Systems (ASYNC 2000)*, pages 188–196. IEEE, 2000.

[104] Helion Technology Limited, The Granary, Home End, Fulbourn, Cambridge CB1 5BS, UK. *DATASHEET - Compact Dual Hash Processor Core for Xilinx FPGA*. `www.heliontech.com`.

[105] A. Hevia and M. A. Kiwi. Strength of two data encryption standard implementations under timing attacks. In C. L. Lucchesi and A. V. Moura, editors, *Theoretical Informatics*, volume 1380 of *Lecture Notes in Computer Science*, pages 192–205. Springer-Verlag, 1998.

[106] P. Hofreiter and P. Laackmann. Electromagnetic espionage from smart cards attacks and countermeasures. Infineon Technologies AG, Technology Update, Smart Cards.

[107] I. S. Hsu, T. K. Truong, L. J. Deutsch, and I. S. Reed. A comparison of VLSI architecture of finite field multipliers using dual, normal, or standard bases. *IEEE Transactions on Computers*, 37(6):735–739, June 1988.

[108] IEEE1363-2000. Standard specifications for public key cryptography, 2000.

[109] IETF. IP Security Protocol (IPSEC). `http://www.ietf.org/html.charters/ipsec-charter.html`.

[110] K. Itoh, J. Yajima, M. Takenaka, and N. Torii. DPA countermeasures by improving the window method. In B. S. Kaliski Jr., Ç. K. Koç, and C. Paar, editors, *Cryptographic Hardware and Embedded Systems (CHES)*, volume 2523 of *Lecture Notes in Computer Science*, pages 303–317. Springer-Verlag, 2002.

[111] T. Itoh and S. Tsujii. Effective recursive algorithm for computing multiplicative inverses in $GF(2^m)$. *Electronics Letters*, 24(6):334–335, 1988.

[112] T. Izu and T. Takagi. A fast parallel elliptic curve multiplication resistant against side channel attacks. Technical Report CORR 2002-03, the Centre for Applied Cryptographic Research (CACR), University of Waterloo, 2002.

[113] T. Izu and T. Takagi. A fast parallel elliptic curve multiplication resistant against side channel attacks. In D. Naccache and P. Paillier, editors, *Practice and Theory in Public Key Cryptosystems (PKC)*, volume 2274 of *Lecture Notes in Computer Science*, pages 280–296. Springer-Verlag, 2002.

[114] S. K. Jain, L. Song, and K. K. Parhi. Efficient semisystolic architectures for finite-field arithmetic. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 6(1):101–113, March 1998.

[115] M. Janke and P. Laackmann. Power and timing analysis attacks against security controllers. Infineon Technologies AG, Technology Update, Smart Cards.

[116] S. Janssens, J. Thomas, W. Borremans, P. Gijsels, I. Verbauwhede, F. Vercauteren, B. Preneel, and J. Vandewalle. Hardware/software co-design of an elliptic curve public-key cryptosystem. In *Proceedings IEEE Workshop on of Signal Processing Systems*, pages 209–216, 2001.

[117] D. Johnson and A. Menezes. The elliptic curve digital signature algorithm (ECDSA). Technical Report CORR 99-34, Department of Combinatorics & Optimization, University of Waterloo, Canada, February 24 2000. `http://www.cacr.math.uwaterloo.ca`.

[118] Joye and J.-J. Quisquater. Hessian elliptic curves and side-channel attacks. In Ç. K. Koç, D. Naccache, and C. Paar, editors, *Cryptographic Hardware and Embedded Systems (CHES)*, volume 2162 of *Lecture Notes in Computer Science*, pages 402–410. Springer-Verlag, 2001.

[119] M. Joye, A. K. Lenstra, and J.-J. Quisquater. Chinese remaindering based cryptosystem in the presence of faults. *Journal of Cryptology*, 4(12):241–245, 1999.

[120] M. Joye and C. Tymen. Protections against differential analysis for elliptic curve cryptography. In Ç. K. Koç, D. Naccache, and C. Paar, editors, *Cryptographic Hardware and Embedded Systems (CHES)*, volume 2162 of *Lecture Notes in Computer Science*, pages 377–390. Springer-Verlag, 2001.

[121] B. Kaliski Jr. The Montgomery inverse and its applications. *IEEE Transactions on Computers*, 44(8):1064–1065, August 1995.

[122] S.-M. Kang and Y. Leblebici. *CMOS Digital Integrated Circuits: Analysis and Design*. McGraw Hill, 2002.

[123] J. Kelsey, B. Schneier, D. Wagner, and C. Hall. Side channel cryptanalysis of product ciphers. *Journal of Computer Security*, 8(2/3), 2000.

[124] V. Klima and T. Rosa. Further results and considerations on side channel attacks on RSA. In B. S. Kaliski Jr., Ç. K. Koç, and C. Paar, editors, *Cryptographic Hardware and Embedded Systems (CHES)*, volume 2523 of *Lecture Notes in Computer Science*, pages 244–259. Springer-Verlag, 2002.

[125] D. E. Knuth. *The Art of Computer Programming*, volume 2:Seminumerical Algorithms. Addison-Wesley, 3 edition, 1997.

[126] N. Koblitz. Elliptic curve cryptosystem. *Mathematics of Computation*, 48:203–209, 1987.

[127] N. Koblitz. *A Course in Number Theory and Cryptography*, volume 114 of *Graduate text in mathematics*. Springer-Verlag, Berlin, Germany, second edition, 1994.

[128] N. Koblitz, A. Menezes, and S. Vanstone. The state of elliptic curve cryptography. *Designs, Codes and Cryptography*, 19:173–193, 2000.

[129] C. K. Koc. RSA hardware implementation. Technical report, RSA Laboratories, RSA Data Security, Inc., Redwood City, CA, August 1995.

[130] C. K. Koc and T. Acar. Montgomery multiplication in $GF\left(2^k\right)$. *Designs, Codes and Cryptography*, 14:57–69, 1998.

[131] C. K. Koc and B. Sunar. Low-complexity bit-parallel canonical and normal basis multipliers for a class of finite fields. *IEEE Transactions on Computers*, 47:353–356, March 1998.

[132] P. Kocher. Timing attacks on implementations of Diffie-Hellman, RSA, DSS and other systems. In N. Koblitz, editor, *Advances in Cryptology: CRYPTO'96*, volume 1109 of *Lecture Notes in Computer Science*, pages 104–113. Springer-Verlag, 1996.

[133] P. Kocher, J. Jaffe, and B. Jun. Differential power analysis. In M. Wiener, editor, *Advances in Cryptology: CRYPTO'99*, volume 1666 of *Lecture Notes in Computer Science*, pages 388–397. Springer-Verlag, 1999.

[134] F. Koeune and J.-J. Quisquater. A timing attack against Rijndael. Technical Report CG-1999/1, UCL Crypto Group, Louvain-la-Neuve, 1999.

[135] O. Kommerling and M. G. Kuhn. Design principles for tamper resistant smartcard processors. In *Proceedings of the USENIX Workshop on Smartcard Technology*, Chicago, Illinois, USA, May 10-11 1999.

[136] X. Lai, J.L. Massey, and S. Murphy. Markov ciphers and differential cryptanalysis. In D. W. Davies, editor, *Advances in Cryptology: EUROCRYPT'91*, volume 547 of *Lecture Notes in Computer Science*, page 1738. Springer-Verlag, 1991.

[137] B. A. Laws and C. K. Rushforth. A cellular-array multiplier for $GF\left(2^m\right)$. *IEEE Transactions on Computers*, C-20:1573–1578, December 1971.

[138] C.-H. Lee and J.-I. Lim. A new aspect of dual basis for efficient field arithmetic. In H. Imai and Y. Zheng, editors, *Practice and Theory in Public Key Cryptography (PKC)*, volume 1560 of *Lecture Notes in Computer Science*, pages 12–28. Springer-Verlag, 1999.

[139] C.-Y. Lee, E.-H. Lu, and J.-Y. Lee. Bit-parallel systolic multipliers for $GF\left(2^m\right)$ fields defined by all-one and equally spaced polynomials. *IEEE Transactions on Computers*, 50(5):385–393, May 2001.

[140] K. H. Leung, K. W. Ma, W. K. Wong, and P. H. W. Leong. FPGA implementation of a microcoded elliptic curve cryptographic processor. In *Proceedings of Field-Programmable Custom Computing Machines (FCCM)*, pages 68–76, 2000.

[141] P. Y. Liardet and N. P. Smart. Preventing SPA/DPA in ECC systems using the Jacobi form. In Ç. K. Koç, D. Naccache, and C. Paar, editors, *Cryptographic Hardware and Embedded Systems (CHES)*, volume 2162 of *Lecture Notes in Computer Science*, pages 391–401. Springer-Verlag, 2001.

[142] R. Lidl and H. Niederreiter. *Introduction to Finite Fields and Their Applications*. Cambridge University Press, New York, NY, USA, 1994.

[143] J. Lopez and R. Dahab. Improved algorithms for elliptic curve arithmetic in $GF(2^n)$. Technical Report IC-98-39, Institute of Computing, State University of Campinas, Campinas, 13081-970 Sao Paulo, Brazil, October 1998.

[144] C.-C. Lu. A search of minimal key functions for normal basis multipliers. *IEEE Transactions on Computers*, 46(5):588–592, May 1997.

[145] S. Mangard. A simple power-analysis attack (SPA) attack on implementations of the AES key expansion. In P. J. Lee and C. H. Lim, editors, *Information Security and Crytography (ICISC)*, volume 2587 of *Lecture Notes in Computer Science*, pages 343–358. Springer-Verlag, 2002.

[146] S. Mangard. Exploiting radiated emissions – EM attacks on cryptographic ICs. In *Proceedings of Austrochip*, Linz, Austria, October 3 2003.

[147] S. Mangard. Hardware countermeasures against DPA – a statistical analysis of their effectiveness. In T. Okamoto, editor, *Topics in Cryptology – CT-RSA*, volume 2964 of *Lecture Notes in Computer Science*, pages 222–235. Springer-Verlag, 2004.

[148] M. M. Mano and C. R. Kime. *Logic and Computer Design Fundamentals*. Prentice Hall, New Jersey, second edition, 2001.

[149] J. L. Massey, G. H. Khachatrian, and M. K. Kuregian. Nomination of SAFER++ as a candidate algorithm for the New European Schemes for Signatures, Integrity and Encryption (NESSIE). Primitive submitted to NESSIE by Cylink Corp., September 2000.

[150] E. D. Mastrovito. *VLSI Architectures for Computations in Galois Fields*. PhD thesis, Department of Electrical Engineering, Linkøping University, Sweden, 1991.

[151] D. May, H. Muller, and N. Smart. Random register renaming to foil DPA. In Ç. K. Koç, D. Naccache, and C. Paar, editors, *Cryptographic Hardware and Embedded Systems (CHES)*, volume 2162 of *Lecture Notes in Computer Science*, pages 28–38. Springer-Verlag, 2001.

[152] L. T. Mc Daniel. An investigation of differential power analysis attacks on FPGA-based encryption systems. Master's thesis, Virginia Polytechnic Insitute, May 29 2003.

[153] R. J. McEliece. *Finite Fields for Computer Scientists and Engineers*. Kluwer Academic Publishers, Boston, MA, USA, 1987.

[154] B. Megarajan. Combinational power analysis on smart cards. Technical report, Department of Electrical & Computer Engineering, Oregon State University, Corvallis, Oregon, 2002.

[155] A. Menezes, P. van Oorschot, and S. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1997.

[156] A. J. Menezes. *Elliptic Curve Public Key Cryptosystems*. Kluwer Academic Publishers, 1993.

[157] N. Mentens, S. B. Örs, B. Preneel, and J. Vandewalle. An FPGA implementation of a Montgomery multiplier over $GF(2^m)$. In Z. Peng, E. Gramatová, S. Hellebrand, N. Frištacký, and M. Fischerová, editors, *Proceedings of the 7th IEEE Workshop on Design & Diagnostics of Electronic Circuits & Systems (DDECS)*, pages 121–128, Stará Lesná, Slovakia, April 18-21 2004.

[158] N. Mentens, S. B. Örs, B. Preneel, and J. Vandewalle. An FPGA implementation of a Montgomery multiplier over $GF(2^m)$. *Computing and Informatics, the Institute of Informatics Slovak Academy of Sciences, Slovak University of Technology, Comenius University and the Slovak Society for Computer Sciences*, special issue: Design and Test, 2004.

[159] N. Mentens, S. B. Örs, B. Preneel, and J. Vandewalle. An FPGA implementation of an elliptic curve processor over $GF(2^m)$. In D. Garrett, C. Zukowski, and J. Lach, editors, *Proceedings of the Great Lakes Symposium on VLSI (GLSVLSI)*, pages 454–457, Boston, MA, USA, April 26-28 2004. ACM Press.

[160] N. Mentens, P. Rommens, and M. Verhelst. Timing and power analysis attacks on the hardware implementation of elliptic curve cryptosystems over $GF(p)$ and $GF(2^m)$. Master's thesis, Katholieke Universiteit Leuven, Departement Elektrotechniek - ESAT, Kasteelpark Arenberg 10, B 3001 Heverlee, Belgium, May 2003.

[161] T. S. Messerges. Securing the AES finalists against power analysis attacks. In B. Schneier, editor, *Fast Software Encryption Workshop (FSE)*, volume 1978 of *Lecture Notes in Computer Science*. Springer-Verlag, 2000.

[162] T. S. Messerges. *Power Analysis Attacks and Countermeasures on Cryptographic Algorithms*. PhD thesis, University of Illinois, 2002.

[163] T. S. Messerges, E. A. Dabbish, and R. H. Sloan. Investigations of power analysis attacks on smartcards. In *Proceedings of the USENIX Workshop on Smartcard Technology*, Chicago, Illinois, USA, May 10-11 1999.

[164] T. S. Messerges, E. A. Dabbish, and R. H. Sloan. Power analysis attacks of modular exponentiation in smartcards. In Ç. K. Koç and C. Paar, editors, *Cryptographic Hardware and Embedded Systems (CHES)*, volume 1717 of *Lecture Notes in Computer Science*, pages 144–157. Springer-Verlag, 1999.

[165] V. Miller. Uses of elliptic curves in cryptography. In H. C. Williams, editor, *Advances in Cryptology: CRYPTO'85*, volume 218 of *Lecture Notes in Computer Science*, pages 417–426. Springer-Verlag, 1985.

[166] A. Miyaji, T. Ono, and H. Cohen. Efficient elliptic curve exponentiation. In Y. Han, T. Okamoto, and S. Qing, editors, *Proceedings of ICICS*, volume 1334 of *Lecture Notes in Computer Science*, pages 282–290. Springer-Verlag, 1997.

[167] B. Moller. Securing elliptic curve point multiplication against side-channel attacks. In G. I. Davida and Y. Frankel, editors, *Information Security (ISC)*, volume 2200 of *Lecture Notes in Computer Science*, pages 324–334. Springer-Verlag, 2001.

[168] P. Montgomery. Modular multiplication without trial division. *Mathematics of Computation*, Vol. 44:519–521, 1985.

[169] S. Moore, R. Anderson, R. Mullins, G. Taylor, and J. Fournier. Balanced self-checking asynchronous logic for smart card applications. *to appear in the Microprocessors and Microsystems Journal*, 2003.

[170] F. Morain and J. Olivos. Speeding up the computations on an elliptic curve using addition-subtraction chains. *Informatique Théorique et Applications*, 24:531–544, 1990.

[171] J. A. Muir. Techniques of side channel cryptanalysis. Master of mathematics in combinatorics and optimization, University of Waterloo, Waterloo, Ontario, Canada, 2001.

[172] R. C. Mullin. Multiple bit multiplier. United States Patent, Number 5,787,028, July 28 1998.

[173] R. C. Mullin, I. M. Onyszchuk, and S. A. Vanstone. Optimal normal bases in $GF(p^n)$. *Discrete Applied Mathemathics*, 22:149–161, 1989.

[174] National Institute of Standards and Technology (NIST). FIPS 186-2: Digital Signature Standard (DSS), January 2000.

[175] National Institute of Standards and Technology (NIST). FIPS 197: Advanced Encryption Standard, November 2001.

[176] National Institute of Standards and Technology (NIST). FIPS 46-3: Data Encryption Standard, October reaffirmed 1999.

[177] NESSIE consortium. NESSIE security report. Technical report, NESSIE, 2003. https://www.cosic.esat.kuleuven.ac.be/nessie/deliverables/D20-v2.pdf.

[178] P. Q. Nguyen and I. E. Shparlinski. The insecurity of the elliptic curve digital signature algorithm with partially known nonces. *Designs, Codes and Cryptography*, 30(2):201–217, September 2003.

[179] R. Novak. SPA-based adaptive chosen-ciphertext attack on RSA implementation. In D. Naccache and P. Paillier, editors, *Practice and Theory in Public Key Cryptosystems (PKC)*, volume 2274 of *Lecture Notes in Computer Science*, pages 252–262. Springer-Verlag, 2002.

[180] R. Novak. Side-channel attack on substitution blocks. In J. Zhou, M. Yung, and Y. Han, editors, *Applied Cryptography and Network Security (ACNS)*, volume 2846 of *Lecture Notes in Computer Science*, pages 307–318. Springer-Verlag, 2003.

[181] NSA. NSA TEMPEST Documents. http://www.cryptome.org/nsa-tempest.htm.

[182] K. Okeya, K. Miyazaki, and K. Sakurai. A fast scalar multiplication method with randomized projective coordinates on a Montgomery-form elliptic curve secure against side channel attacks. In K. Kim, editor, *Information Security and Cryptology (ICISC)*, volume 2288 of *Lecture Notes in Computer Science*, pages 428–439. Springer-Verlag, 2001.

[183] K. Okeya and K. Sakurai. Power analysis breaks elliptic curve cryptosystems even secure against the timing attack. In B. Roy and E. Okamoto, editors, *Cryptology in India (INDOCRYPT)*, volume 1977 of *Lecture Notes in Computer Science*, pages 178–190. Springer-Verlag, 2000.

[184] K. Okeya and K. Sakurai. On insecurity of the side channel attack countermeasure using addition-subtraction chains under distinguishability between addition and doubling. In L. M. Batten and J. Seberry, editors, *Information Security and Privacy (ACISP)*, volume 2384 of *Lecture Notes in Computer Science*, pages 420–435. Springer-Verlag, 2002.

[185] K. Okeya and T. Takagi. A more flexible countermeasure against side channel attacks using window method. In C. Walter, Ç. K. Koç, and C. Paar, editors, *Cryptographic Hardware and Embedded Systems (CHES)*, volume 2779 of *Lecture Notes in Computer Science*, pages 397–410. Springer-Verlag, 2003.

[186] K. Okeya and T. Takagi. The width-w NAF method provides small memory and fast elliptic scalar multiplications secure against side channel attacks. In M. Joye, editor, *Topics in Cryptology-CT-RSA*, volume 2612 of *Lecture Notes in Computer Science*, pages 328–342. Springer-Verlag, 2003.

[187] J. K. Omura and J. L. Massey. Computation method and apparatus for finite field arithmetic. United States Patent, Patent Number: 4.587.627, May 6, 1986.

[188] I. M. Onyszchuk, R. C. Mullin, and S. A. Vanstone. Computational method and apparatus for finite field multiplication. United States Patent, Number 4.745.568, May 17, 1988.

[189] Openssl. Openssl project. `http://www.openssl.org`.

[190] OptiMagic, Inc. Frequently-Asked Questions (FAQ) About Programmable Logic. `http://www.optimagic.com/faq.html\#FPGA`.

[191] G. Orlando and C. Paar. A super-serial Galois fields multiplier for FPGAs and its application to public-key algorithms. In *Proceedings of Seventh Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM'99)*, pages 232–239, 1999.

[192] G. Orlando and C. Paar. A high-performance reconfigurable elliptic curve processor for $GF(2^m)$. In Ç. K. Koç and C. Paar, editors, *Cryptograpic Hardware and Embedded Systems (CHES)*, volume 1965 of *Lecture Notes in Computer Science*, pages 41–56. Springer-Verlag, 2000.

[193] G. Orlando and C. Paar. Squaring architecture for $GF(2^m)$ and its applications in cryptographic systems. *Electronics Letters*, 36(13):1116–1117, June 2000.

[194] G. Orlando and C. Paar. A scalable $GF(p)$ elliptic curve processor architecture for programmable hardware. In Ç. K. Koç, D. Naccache, and C. Paar, editors, *Cryptograpic Hardware and Embedded Systems (CHES)*, volume 2162 of *Lecture Notes in Computer Science*, pages 356–371. Springer-Verlag, 2001.

[195] S. B. Ors, L. Batina, and B. Preneel. Hardware implementation of elliptic curve processor over $GF(p)$. New European Schemes for Signatures, Integrity and Encryption (NESSIE) working paper NES/DOC/-KUL/WP5/023, Department of Electrical Engineering, ESAT/COSIC, Katholieke Universiteit Leuven, 10 October 2002.

[196] S. B. Ors, L. Batina, B. Preneel, and J. Vandewalle. Hardware implementation of a Montgomery modular multiplier in a systolic array. In *The The 10th Reconfigurable Architectures Workshop (RAW)*, Nice, France, April 22 2003.

[197] S. B. Ors, L. Batina, B. Preneel, and J. Vandewalle. Hardware implementation of an elliptic curve processor over $GF(p)$. In *IEEE 14th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, pages 433–443, The Hague, The Netherlands, June 24-26 2003.

[198] S. B. Ors, L. Batina, B. Preneel, and J. Vandewalle. Hardware implementation of an elliptic curve processor over $GF(p)$ with Montgomery modular multiplier. *International Journal of Embedded Systems*, February 2005.

[199] S. B. Ors, F. K. Gürkaynak, E. Oswald, and B. Preneel. *Embedded Cryptographic Hardware: Design and Security*, chapter Power-Analysis Attack on an ASIC AES implementation. Nova Science, NY, USA, 2004.

[200] S. B. Ors, F. K. Gürkaynak, E. Oswald, and B. Preneel. Power-analysis attack on an ASIC AES implementation. In *Proceedings of the International Conference on Information Technology (ITCC)*, pages 546–552, Las Vegas, NV, USA, April 5-7 2004.

[201] S. B. Ors, E. Oswald, and B. Preneel. Power-analysis attacks on an FPGA – first experimental results. In C. Walter, Ç. K. Koç, and C. Paar, editors, *Cryptographic Hardware and Embedded Systems (CHES)*, volume 2779 of *Lecture Notes in Computer Science*, pages 35–50. Springer-Verlag, 2003.

[202] S. B. Ors and B. Preneel. Evaluation of elliptic curve digital signature algorithm. New European Schemes for Signatures, Integrity and Encryption (NESSIE) working paper NES/DOC/KUL/WP6/018/1, Department of Electrical Engineering, ESAT/COSIC, Katholieke Universiteit Leuven, 11 March 2002.

[203] H. Orup. Simplifying quotient determination in high-radix modular multiplication. In *Proceedings of the 12th Symposium on Computer Arithmetic*, pages 193–199. IEEE, 1995.

[204] E. Oswald. Enhancing simple power-analysis attacks on elliptic curve cryptosystems. In B. S. Kaliski Jr., Ç. K. Koç, and C. Paar, editors, *Cryptographic Hardware and Embedded Systems (CHES)*, volume 2523 of *Lecture Notes in Computer Science*, pages 82–97. Springer-Verlag, 2002.

[205] E. Oswald. *On Side-Channel Attacks and the Application of Algorithmic Countermeasures*. PhD thesis, Institute for Applied Information Processing and Communications (IAIK), TU Graz, Austria, June 2003.

[206] E. Oswald and M. Aigner. Randomized addition-subtraction chains as a countermeasure against power attacks. In Ç. K. Koç, D. Naccache, and C. Paar, editors, *Cryptograpic Hardware and Embedded Systems (CHES)*, volume 2162 of *Lecture Notes in Computer Science*, pages 39–50. Springer-Verlag, 2001.

[207] C. Paar. A new architecture for a parallel finite field multiplier with low complexity based on composite fields. *IEEE Transactions on Computers*, 45(7):856–860, July 1996.

[208] C. Paar. Fast arithmetic architectures for public-key algorithms over Galois fields $GF((2^n)^m)$. In W. Fumy, editor, *Advances in Cryptology: EUROCRYPT'97*, volume 1233 of *Lecture Notes in Computer Science*, pages 363–378. Springer-Verlag, 1997.

[209] C. Paar and N. Lange. A comparative VLSI synthesis of finite field multipliers. In *Proceedings of the 3rd International Symposium on Communication Theory & Applications*, Lake District, UK, July 10-14 1995.

[210] C. Paar and M. Rosner. Comparison of arithmetic architectures for Reed-Solomon decoders. In K. L. Pocek and J. Arnold, editors, *Proceedings of FCCM'97*, pages 219–225, Napa Valley, California, April 16-18 1997. IEEE Computer Society Press.

[211] B. Parhami. Generalized signed-digit number systems: A unified framework for redundant number representations. *IEEE Transactions on Computers*, 39(1):89–98, January 1990.

[212] B. Parhami. *Computer Arithmetic: Algorithms and Hardware Designs*. Oxford University Press, Inc., New York, 2000.

[213] M. G. Parker and M. Benaissa. Modular arithmetic using low order redundant bases. *IEEE Transactions on Computers*, 46(5):611–616, May 1997.

[214] J. Pollard. Monte Carlo method for index computation ($\mathrm{mod} p$). *Mathematics of Computation*, 32:918–924, 1978.

[215] J.-J. Quisquater and D. Samyde. Electromagnetic analysis (EMA): Measures and counter-measures for smard cards. In I. Attali and T. Jensen, editors, *Research in Smart Cards: Smart Card Programming and Security (E-smart)*, volume 2140 of *Lecture Notes in Computer Science*, pages 200–210. Springer-Verlag, 2001.

[216] A. Reyhani-Masoleh and M. A. Hasan. Efficient digit-serial normal basis multipliers over $GF(2^m)$. In *Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 781–784, 2002.

[217] A. Reyhani-Masoleh and M. A. Hasan. Efficient multiplication beyond optimal normal bases. Technical Report CORR 2002-12, The Centre for Applied Cryptographic Research (CACR), University of Waterloo, 2002.

[218] A. Reyhani-Masoleh and M. A. Hasan. A new construction of Massey-Omura parallel multiplier over $GF(2^m)$. *IEEE Transactions on Computers*, 51(5):511–520, May 2002.

[219] R. L. Rivest. The RC5 encryption algorithm. In B. Preneel, editor, *Fast Software Encryption (FSE)*, volume 1008 of *Lecture Notes in Computer Science*, pages 86–96. Springer-Verlag, 1994.

[220] 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.

[221] T. Romer and J.-P. Seifert. Information leakage attacks against smart card implementations of the elliptic curve digital signature algorithm. In I. Attali and T. P. Jensen, editors, *Research in Smart Cards, Smart Card Programming and Security (E-smart)*, volume 2140 of *Lecture Notes in Computer Science*, pages 211–219. Springer-Verlag, 2001.

[222] G. Rouvroy, F.-X. Standaert, J.-J. Quisquater, and J.D.Legat. Design strategies and modified descriptions to optimize cipher FPGA implementations: Fast and compact results for DES and triple-DES. In P. Y. K. Cheung, G. A. Constantinides, and J. T. de Sousa, editors, *Field-Programmable Logic and Applications (FPL)*, volume 2778 of *Lecture Notes in Computer Science*, pages 181–193. Springer-Verlag, 2003.

[223] RSA Labs. RSAREF 2.0.1996.03 - public- and private-key encryption library. `http://www.rsalabs.com/`.

[224] A. Satoh and K. Takano. A scalable dual-field elliptic curve cryptographic processor. *IEEE Transactions on Computers*, 52(4):449–460, April 2003.

[225] E. Savas and Ç. K. Koç. The Montgomery modular inverse revisited. *IEEE Transactions on Computers*, 49(7):763–766, July 2000.

[226] E. Savas, A. F. Tenca, and Ç. K. Koç. A scalable and unified multiplier architecture for finite fields $GF(p)$ and $GF\left(2^m\right)$. In C. Paar and Ç. K. Koç, editors, *Cryptographic Hardware and Embedded Systems (CHES)*, volume 1965 of *Lecture Notes in Computer Science*, pages 281–296. Springer-Verlag, 2000.

[227] P. Schaumont and I. Verbauwhede. A reconfiguration hierarchy for elliptic curve cryptography. In *Proceedings of 35th Asilomar Conference on Signals, Systems, and Computers*, November 4-7 2001.

[228] P. Schaumont, I. Verbauwhede, K. Keutzer, and M. Sarrafzadeh. A quick safari through the reconfiguration jungle. In *Proceedings of 38th Design Automation Conference (DAC'01)*, pages 172–177, Las Vegas, NV, USA, June 18-22 2001.

[229] W. Schindler. A timing attack against RSA with the Chinese remainder theorem. In C. Paar and Çetin Koç, editors, *Cryptographic Hardware and Embedded Systems (CHES)*, volume 1965 of *Lecture Notes in Computer Science*, pages 109–124. Springer-Verlag, 2000.

[230] W. Schindler, F. Koeune, and J.-J. Quisquater. Unleashing the full power of timing attack. Technical Report CG-2001/3, UCL Crypto Group, 2001.

[231] B. Schneier, J. Kelsey, D. Whiting, D. Wagner, C. Hall, and N. Ferguson. *The Twofish Encryption Algorithm*. John Wiley & Sons, 1996.

[232] K. Schramm, G. Leander, P. Felke, and C. Paar. A collision-attack on AES combining side channel- and differential-attack. In M. Joye and J.-J. Quisquater, editors, *Cryptographic Hardware and Embedded Systems (CHES)*, volume 3156 of *Lecture Notes in Computer Science*. Springer-Verlag, 2004.

[233] K. Schramm, T. Wollinger, and C. Paar. A new class of collision attacks and its application to DES. In T. Johansson, editor, *Fast Software Encryption (FSE)*, volume 2887 of *Lecture Notes in Computer Science*. Springer-Verlag, 2003.

[234] R. A. Serway. *Physics for Scientists and Engineers*. Saunders Golden sunburst series. Saunders college publishing, 1996.

[235] A. Shamir. Protecting smart cards from passive power analysis with detached power supplies. In C. Paar and Ç. Koç, editors, *Cryptographic Hardware and Embedded Systems (CHES)*, volume 1965 of *Lecture Notes in Computer Science*, pages 71–77. Springer-Verlag, 2000.

[236] A. Shamir and E. Tromer. Acoustic cryptanalysis. Preliminary proof-of-concept presentation, 2004. `http://www.wisdom.weizmann.ac.il/~tromer/acoustic/`.

[237] L. Shang, A. Kaviani, and K. Bathala. Dynamic power consumption in Virtex2 FPGA family. In *Proceedings of the tenth ACM International Symposium on Field-Programmable Gate Arrays (FPGA)*, pages 157–164, Monterey, CA, USA, February 24-26 2002. ACM.

[238] J. H. Silverman. Fast multiplication in finite field $GF\left(2^N\right)$. In Ç. K. Koç and C. Paar, editors, *Cryptographic Hardware and Embedded Systems (CHES)*, volume 1717 of *Lecture Notes in Computer Science*, pages 123–134. Springer-Verlag, 1999.

[239] G. J. Simmons, editor. *Contemporary Cryptology, The Science of Information Integrity*. IEEE Press, 1992.

[240] N. P. Smart. How secure are elliptic curves over composite extension fields? In B. Pfitzmann, editor, *Advances in Cryptology: EUROCRYPT'01*, volume 2045 of *Lecture Notes in Computer Science*, pages 30–39. Springer-Verlag, 2001.

[241] J. Solinas. An improved algorithm for arithmetic on a family of elliptic curves. In B. Kaliski Jr., editor, *Advances in Cryptology: CRYPTO'97*, volume 1294 of *Lecture Notes in Computer Science*, pages 357–371. Springer-Verlag, 1997.

[242] D. X. Song, D. Wagner, and X. Tian. Timing analysis of keystrokes and timing attacks on SSH. In *Proceedings of the tenth USENIX Security Symposium*, Washington, D.C., USA, August 13-17 2001.

[243] L. Song and K. K. Parhi. Low-complexity modified Mastrovito multipliers over finite fields $GF\left(2^M\right)$. In *Proceedings of the International Symposium on Circuits and Systems (ISCAS)*, pages 508–512. IEEE, 1999.

[244] F.-X. Standaert, S. B. Örs, and B. Preneel. Power analysis attack on an FPGA implementation of AES. In M. Joye and J.-J. Quisquater, editors, *Cryptograpic Hardware and Embedded Systems (CHES)*, volume 3156 of *Lecture Notes in Computer Science*, pages 30–44. Springer-Verlag, 2004.

[245] F.-X. Standaert, S. B. Örs, B. Preneel, and J.-J. Quisquater. Power analysis attacks against FPGA implementations of DES. In J. Becker, M. Platzner, and S. Vernalde, editors, *Field-Programmable Logic and its Applications (FPL)*, volume 3203 of *Lecture Notes in Computer Science*. Springer-Verlag, 2004.

[246] F.-X. Standaert, G. Rouvroy, J.-J. Quisquater, and J.-D. Legat. Efficient implementation of Rijndael encryption in reconfigurable hardware: Improvements and design tradeofs. In C. Walter, Ç. K. Koç, and C. Paar, editors, *Cryptograpic Hardware and Embedded Systems (CHES)*, volume 2779 of *Lecture Notes in Computer Science*, pages 334–350. Springer-Verlag, 2003.

[247] B. Sunar and Ç. K. Koç. Mastrovito multiplier for all trinomials. *IEEE Transactions on Computers*, 48(5):522–527, May 1999.

[248] B. Sunar and Ç. K. Koç. An efficient optimal normal basis type ii multiplier. *IEEE Transactions on Computers*, 50(1):83–87, January 2001.

[249] S. Sutikno, R. Effendi, and A. Surya. Design and implementation of arithmetic processor $GF\left(2^{155}\right)$ for elliptic curve cryptosystems. In *Proceedings of the IEEE Asia-Pacific Conference on Circuits and Systems (APCCAS)*, pages 647–650, 1998.

[250] S. Sutikno and A. Surya. An architecture of $F_{2^{2N}}$ multiplier for elliptic curves cryptosystem. In *Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS)*, volume 1, pages 279–282, Geneva, Switzerland, 2000.

[251] S. Sutikno, A. Surya, and R. Effendi. An implementation of ElGamal elliptic curve cryptosystem. In *Proceedings of the IEEE Asia Pacific Conference on Circuits and Systems*, pages 483–486, Chiangmai, Thailand, 1998.

[252] N. Takagi, J. Yoshiki, and K. Takagi. A fast algorithm for multiplicative inversion in $GF(2^m)$ using normal basis. *IEEE Transactions on Computers*, 50(5):394–398, May 2001.

[253] K. Tiri and I. Verbauwhede. Securing encryption algorithms against DPA at the logic level: Next generation smart card technology. In C. Walter, Ç. K. Koç, and C. Paar, editors, *Cryptographic Hardware and Embedded Systems (CHES)*, volume 2779 of *Lecture Notes in Computer Science*, pages 125–136. Springer-Verlag, 2003.

[254] E. Trichina. Combinational logic design for AES subbyte transformation on masked data. Cryptology ePrint Archive: Report 2003/236, 2003.

[255] E. Trichina and A. Bellezza. Implementation of elliptic curve cryptography with built-in counter measures against side channel attacks. In B. S. Kaliski Jr., Ç. K. Koç, and C. Paar, editors, *Cryptographic Hardware and Embedded Systems (CHES)*, volume 2535 of *Lecture Notes in Computer Science*, pages 98–113. Springer-Verlag, 2002.

[256] E. Trichina, D. De Seta, and L. Germani. Simplified adaptive multiplicative masking for AES. In B. S. Kaliski Jr., Ç. K. Koç, and C. Paar, editors, *Cryptographic Hardware and Embedded Systems (CHES)*, volume 2535 of *Lecture Notes in Computer Science*, pages 187–197. Springer-Verlag, 2002.

[257] W.-C. Tsai, C. B. Shung, and S.-J. Wang. A systolic architecture for elliptic curve cryptosystems. In *Proceedings of the ICSP*, pages 591–597, 2000.

[258] C. D. Walter. Montgomery exponentiation needs no final subtraction. *Electronic letters*, 35(21):1831–1832, October 1999.

[259] C. D. Walter. Montgomery's multiplication technique: How to make it smaller and faster. In Ç. K. Koç and C. Paar, editors, *Cryptographic Hardware and Embedded Systems (CHES)*, volume 1717 of *Lecture Notes in Computer Science*, pages 80–93. Springer-Verlag, 1999.

[260] C. D. Walter. Sliding windows succumbs to big mac attack. In Ç. K. Koç, D. Naccache, and C. Paar, editors, *Cryptographic Hardware and Embedded Systems (CHES)*, volume 2162 of *Lecture Notes in Computer Science*, pages 286–299. Springer-Verlag, 2001.

[261] C. D. Walter. MIST: An efficient, randomized exponentiation algorithm for resisting power analysis. In B. Preneel, editor, *Topics in Cryptology – CT-RSA*, volume 2271 of *Lecture Notes in Computer Science*, pages 53–66. Springer Verlag, 2002.

[262] C. D. Walter. Precise bounds for Montgomery modular multiplication and some potentially insecure RSA moduli. In B. Preneel, editor, *Topics in Cryptology – CT-RSA*, volume 2271 of *Lecture Notes in Computer Science*, pages 30–39. Springer-Verlag, 2002.

[263] C. D. Walter. Some security aspects of the MIST randomized exponentiation algorithm. In B. S. Kaliski Jr., Ç. K. Koç, and C. Paar, editors, *Cryptographic Hardware and Embedded Systems (CHES)*, volume 2523 of *Lecture Notes in Computer Science*. Springer-Verlag, 2002.

[264] C. D. Walter and S. Thompson. Distinguishing exponent digits by observing modular subtractions. In D. Naccache, editor, *Topics in Cryptology – CT-RSA*, volume 2020 of *Lecture Notes in Computer Science*, pages 192–207. Springer-Verlag, 2001.

[265] C. A. Wang, T. K. Truong, H. M. Shao, L. J. Deutsch, J. K. Omura, and I. S. Reed. VLSI architectures for computing multiplications and inverses in $GF\,(2^m)$. *IEEE Transactions on Computers*, 34(8):709–717, August 1985.

[266] C. C. Wang. Algorithm to design finite-field normal-basis multipliers. Technical Report NPO-17109, NASA's Jet Propulsion Laboratory, Pasadena, California, 1986.

[267] C. C. Wang. An algorithm to design finite field multipliers using a self-dual normal bases. *IEEE Transactions on Computers*, 38(10):1457–1460, October 1989.

[268] C.-L. Wang and J.-H. Guo. New systolic arrays for $C + AB^2$, inversion, and division in $GF\,(2^m)$. *IEEE Transactions on Computers*, 49(10):1120–1125, October 2000.

[269] S.-W. Wei. VLSI architectures for computing exponentiations, multiplicative inverses, and divisions in $GF\,(2^m)$. *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, 44(10):847–855, October 1997.

[270] M. J. Wiener. Performance comparison of public-key cryptosystems. *RSA CryptoBytes*, 4(1):1–5, 1998.

[271] J. Wolkerstorfer. Dual-field arithmetic unit for $GF(p)$ and $GF\,(2^m)$. In B. S. Kaliski Jr., Ç. Koç, and C. Paar, editors, *Cryptographic Hardware and Embedded Systems (CHES)*, volume 2523 of *Lecture Notes in Computer Science*. Springer-Verlag, 2002.

[272] H. Wu. Low complexity bit-parallel finite field arithmetic using polynomial basis. In Ç. Koç and C. Paar, editors, *Cryptographic Hardware and Embedded Systems (CHES)*, volume 1717 of *Lecture Notes in Computer Science*, pages 280–291. Springer-Verlag, 1999.

[273] H. Wu. Montgomery multiplier and squarer in $GF(2^m)$. Technical Report CORR 2000-28, The Centre for Applied Cryptographic Research (CACR), University of Waterloo, Canada, 2000.

[274] H. Wu. On complexity of squaring using polynomial basis in $GF(2^m)$. Technical Report CORR 2000-28, The Centre for Applied Cryptographic Research (CACR), University of Waterloo, Canada, 2000.

[275] H. Wu. Montgomery multiplier and squarer for a class of finite fields. *IEEE Transactions on Computers*, 51(5):521–529, May 2002.

[276] H. Wu and M. A. Hasan. Low complexity bit-parallel multipliers for a class of finite fields. *IEEE Transactions on Computers*, 47(8):883–887, August 1998.

[277] H. Wu, M. A. Hasan, and I. F. Blake. New low-complexity bit-parallel finite field multipliers using weakly dual nases. *IEEE Transactions on Computers*, 47(11):1223–1234, November 1998.

[278] H. Wu, M. A. Hasan, and I. F. Blake. Highly regular architectures for finite field computation using redundant basis. In Ç. Koç and C. Paar, editors, *Cryptographic Hardware and Embedded Systems (CHES)*, volume 1717 of *Lecture Notes in Computer Science*, pages 280–291. Springer-Verlag, 1999.

[279] Xilinx, Inc. *Virtex 2.5 V Field Programmable Gate Arrays*, April 2 2001. `http://direct.xilinx.com/bvdocs/publications/ds003.pdf`.

[280] Xilinx, Inc. *Powering Xilinx FPGAs*, August 5 2002. `http://support.xilinx.com/xapp/xapp158.pdf`.

[281] Z. Yan and D. V. Sarwate. Systolic architectures for finite field inversion and division. In *Proceedings of The IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 789–792, 2002.

[282] C.-S. Yeh, I. S. Reed, and T. K. Truong. Systolic multipliers for finite fields $GF(2^m)$. *IEEE Transactions on Computers*, C-33(4):357–360, April 1984.

[283] Z. Yu. Low power finite field multiplication and division in re-configurable Reed-Solomon codes. In *Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 785–788, 2002.

[284] T. Zhang and K. K. Parhi. Systematic design of original and modified Mastrovito multipliers for general irreducible polynomials. *IEEE Transactions on Computers*, 50(7):734–748, July 2001.

Biography

Sıddıka Berna Örs Yalçın was born in Ankara, Turkey, in 1973. She received the B.Sc. degree in Electronics and Communications Engineering from Istanbul Technical University, Electrical and Electronic Engineering Faculty, Turkey in 1995 and the M.Sc degree in Electronics and Communications Engineering from Istanbul Technical University, Institute of Science and Technology, Turkey in 1998. She worked as a research and teaching assistant at Istanbul Technical University, Electrical and Electronics Faculty between 1995-1999.