Silesian University of Technology
Faculty of Automatic Control, Electronics and Computer Science
Institute of Electronics
University of Rennes 1
IRISA

A DISSERTATION

Arithmetic operators on $GF(2^m)$ for cryptographic applications: performance - power consumption - security tradeoffs

Author:

Danuta Pamuła

Supervisors:
dr hab. inż. Edward Hrynkiewicz,
prof. nzw. w Politechnice Śląskiej (PL)
Arnaud Tisserand,
CNRS researcher, HDR (FR)

Submitted in total fulfillment of
the requirements of the degree of
Doctor of Philosophy
under a Cotutelle agreement with
Silesian University of Technology (PL)
and University of Rennes 1 (FR).

Gliwice 2012

Acknowledgements

I would like to thank Professor Edward Hrynkiewicz and Arnaud Tisserand, my research supervisors, for their patient guidance, enthusiastic encouragement and useful critiques of this research work.

Contents

List of Figures

Nomenclature

$[k]P$	scalar point multiplication
$\mathbb{F}_q$	finite field
AT	efficiency factor
$f(x)$	irreducible polynomial, field generator
$GF(2^m), \mathbb{F}_{2^m}$	binary extension fields
$GF(p), \mathbb{F}_p$	prime field
m	field size
ASIC	Application Specific Integrated Circuits
DLP	Discrete logarithm problem
DPA	Differential Power Analysis
ECC	Elliptic curve cryptography
ECDLP	Elliptic curve discrete logarithm problem
FFT	Fast Fourier Transform
FPGA	Field Programmable Gate Arrays
FSM	finite state machine
HDL	Hardware Description Language
LFSR	Linear feedback shift register
LUT	lookup table
MSB	most significant bit

NIST National Institute of Standards and Technology

NP-hard non-deterministic polynomial-time hard

PKC Public Key Cryptography

RSA Rivest-Shamir-Adleman

SCA Side Channel Attack

SECG Standards for Efficient Cryptography Group

SFM Spectral Flatness Measure

SPA Simple Power Analysis

1. Introduction

Digital systems and Internet are nowadays spanning most domains of our lives. They are responsible for communication between people, institutions, for controlling airport systems, transport systems, managing medical systems, etc. Digital systems start to appear everywhere and are responsible for more and more important and confidential processes. We are flooded with digital data, which are not always easy to authenticate, manage and secure. Generally majority of common users of digital systems do not care much about authentication, confidentiality, integrity and security of their data. They are still little aware of possibilities of stealing, tampering or using their digital data or what is worse their digital identity (identity fraud is a serious threat [90]). They are even less aware of consequences resulting from such abuses or negligence of security matters [90, 63].

Fortunately security awareness slowly increases mainly due to rapid development and increase of number of services performed in a digital way. People start to perceive the meaning (necessity) of securing data. Everyone wants to securely perform banking transactions, safely sign important documents, protect confidential data (tax, medical, etc.) or just safely shop online. On the other hand, nobody wants to be bothered about securing data and nobody wants that this process will in any way interrupt normal work of a system. Luckily most system developers have information security awareness and tend to equip digital systems and communication channels with more and more efficient security issues, depending on application and requirements. The security of a system has to be very often verified because although users start to take precautions and new ways for securing data are developed, new ways of stealing and tampering data also appear.

The science, which provides us with means to secure data, is called cryptography [66, 99]. Cryptography dates back to ancient times [43]. It was used to cipher messages to prevent adversaries from reading it. First ciphers were very naive but usually sufficient due to the fact that most people were illiterate. As centuries passed and elementary education became a standard, ciphers had to become more and more sophisticated. Nowadays, cryptography has to exploit properties of NP-hard mathematical problems (see [6] on computational complexity) to provide us with new means of data security. The mathematicians working on encryption algorithms constantly adapt them to arising needs and computer scientists create new information security systems employing them (for more details see [91]). With development of new technologies designers tend to create faster and more efficient cryptographic systems. Unfortunately as the technical and theoretical possibilities of securing data increase, the number of ways of tampering communication and recovering secret and hidden data also increases. In fact cryptography, treating about concealing the secret, is just one branch of a wider science: cryptology. The other branch of cryptology, evolving simultaneously is cryptanalysis, which concerns breaking the ciphers and data security (see [100, 105]). Due to developments in cryptanalysis, modern cryptographic systems suffer from more threats than their predecessors. They have not only to be mathematically secure but also physically secure.

At first it was sufficient to employ a simple, secure, mathematically unbreakable cryptographic algorithm. Then it occurred that with development of computational power of computer systems and new means of communication (Internet, wireless communication), the mathematical security of most algorithms should be revised, and either new algorithms should be developed or their parameters have to be changed [99]. After managing the problems of algorithms' mathematical security, it was proven that there exist other ways of extracting secrets from cryptographic systems. Cryptanalysts came up with idea to eavesdrop work of digital cryptographic systems developed to secure data [68]. They propose to analyse power trace, current signa-

tures, execution time and other leaking information, concerned useless, in order to correlate them directly with the secret or with operations executed on secret in the cryptographic system.

Unfortunately their approach for recovering the secret was successful [53] and nowadays it is not only sufficient to employ mathematically secure cryptographic algorithms but also to secure their implementations as well as systems and devices performing cryptographic operations against adversaries. It implies that safe and mathematically unbreakable algorithm is not enough to secure the data; one needs also to secure hardware or software solutions against information leakage. It is proven that it is possible to record power trace, current trace or electromagnetic emissions, or observe execution time and by analysis of obtained information deduce secret data. Such approach is called side-channel analysis or side-channel attack (SCA), see [103]. Until very recently, information leaking from the device during its work was concerned as useless noise and designers did not especially bothered to decrease or control it. Fortunately, now security systems developers/researchers are aware that every information "leaking" from the cryptographic device can be useful to the attacker. To avoid loss of secret data developers analyse the behaviour of their devices in order to make them secure against eavesdropping. New ways of securing data and cryptographic processors are being developed making attackers job harder. Simultaneously methods for secret data retrieving also develop, decreasing the strength of added security issues (countermeasures) [105].

There are few families of possible side-channel analysis attacks [88] depending on which side channel the attacker is exploiting. To retrieve secret the attacker analyses timings of the operation, power consumed by the device or the character of electromagnetic radiations. The side-channel analysis attacks are so called passive attacks, they are based on the information eavesdropped during circuit work, they do not interfere with the device. There exist also active attacks in which the attacker manipulates cryptographic device and/or its environment, see [7]. Usually the attacker tries to

insert errors (fault-injection attacks) in device work, tries to force unnormal behaviour of the device or manipulates clocks to observe changes in device behaviour which may give information about secret.

In our researches we are motivated by the possibility of ensuring more powerful physical security of cryptographic systems especially against power analysis attacks. There are still many ideas for countermeasures to verify and there are still units of cryptographic systems, which were not considered during security level evaluation, i.e. for which no countermeasures against SCA were yet provided. We aim at fulfilling parts of those security gaps.

Adding countermeasures against SCA is not a trivial task [35]. Some may overload cryptographic device and degrade its performance. Some countermeasures may protect against one type of SCA but may make the other type more feasible to succesfully perform [26]. The ideal countermeasures are such that do not decrease the overall performance, efficiency and do not increase the cost of the cryptographic system too much. The cryptographic systems are already complex circuits due to the fact that they employ a lot of arithmetic computations on large numbers. Thus overloading them with useless subcircuits generating additional activity may cause serious decrease of efficiency, especially in terms of area. Moreover adding noise to blind the operations performed is speculative because there exist effective denoising methods in signal processing, see [87]. Additionally the noise adding countermeasures are insufficient as an autonomous countermeasures. They can serve as an additional protection element [60].

Thus we are motivated by a possibility of increasing the security of cryptographic system in such a way that it will not result in degradation of its efficiency and overall cost increase. What is more we want to increase the overall efficiency of cryptographic system and decrese its cost. To be able to achieve our goals we first have to propose very efficient computation units dedicated to work in cryptographic systems and then try to insert the countermeasures in such a way that elaborated efficiency of our units will not decrease. That way we presume we may improve overall cryptographic

system performance (by increasing efficiency of its basic units) and cryptographic system security (by inserting necessary countermeasures against eavesdropping).

Utilising reconfigurable circuits, for instance Field Programmable Gate Arrays (FPGAs) [46, 37], as a target platforms for our cryptographic devices seems to provide a lot of possibilities in our field of research. Such circuits allow for quick evaluation of proposed solutions and inserted countermeasures. They are relatively cheap, flexible and provide a great mean for prototyping circuits before implementation in more expensive Application Specific Integrated Circuits (ASIC). Another advantage of FPGA solution is that it is much harder to successfully attack them than a solution implemented on microprocessors, due to for example sequential and predictable nature of operation of a microprocessor.

Cryptographic systems rely on arithmetic operations and complex mathematics, they exploit certain mathematical problems, which are infeasible to solve. There exist two types of modern cryptographic systems, utilising: secret-key cryptography or public-key cryptography (PKC). Our work concerns the second type, the public-key cryptography. There are three most widely used types of PKC systems. They are divided regarding the mathematical problem their security is based on. The most commonly exploited problems are [36]: integer factorisation problem (e.g. RSA system), discrete logarithm problem (e.g. ElGamal system) and elliptic curve discrete logarithm problem (Elliptic Curve Cryptography system). We have decided to consider in our research security and efficiency of cryptographic systems based on elliptic curve discrete logarithm problem; that is Elliptic Curve Cryptography (ECC) [36] systems. The ECC is very advantageous especially due to the fact that it operates on much smaller numbers than for example RSA , in order to provide the same level of security. This fact should create the possibility to propose much more efficient cryptographic hardware solutions.

The elliptic curve cryptography concerns/exploits mathematical properties of elliptic curves defined over finite fields. Main ECC protocols operations are performed on points of such elliptic curves. To perform those operations (curve-level operations) one needs to perform operations on the coordinates of elliptic curve points, i.e. on the elements of the underlying finite field. Due to this, the operations on the elements of finite fields are the ones on which really the work of any ECC protocol depends. The efficiency of finite-field computation units is crucial for the efficiency of ECC systems.

There exist many ways of protecting the operations performed on points of elliptic curves (curve-level operations) or operations performed by ECC cryptographic protocols, see [26]. However there are not yet known any means for securing the operations in the underlying finite field (field-level operations). According to the fact that efficiency and work of ECC systems depend on the performance of the operations performed in finite fields [97], we find that security of whole system may also depends on the finite field arithmetic units security. The motivation for our research is the possibility to increase the security and efficiency of whole ECC system via securing and improving finite-field arithmetic operators responsible for performing vital computations in ECC systems.

In elliptic curve cryptography, many SCAs [80] have been proposed. To protect circuits against those attacks researchers propose various countermeasures, or protections, see [39]. Moreover, specific protections at the arithmetic level (curve-level operations arithmetic) have been proposed. For instance, addition chains allow performing only one type of operation, point addition, during scalar multiplications [14]. In [15] randomisation techniques are used. But these protections are at the curve-level not the finite-field one. At the moment the means and effects of protecting finite-field arithmetic operators are not yet exploited. It seems that if except just securing curve level operations of the ECC processor we will secure also arithmetic operators, which efficiency is crucial for curve-level opera-

tions, we can make our cryptographic system more difficult to break (to attack successfully). We presume that leaking information are much harder to analyse and to correlate with a secret when the basic arithmetic units operations are secured against eavesdropping. Our objective is to protect cryptographic devices as much as possible against some SCAs. Usually the only thing, which stops cryptanalysts from recovering secret data (breaking the device), is insufficient computational power of available computer systems. The more countermeasures and protections the more computational power needed to break the system.

Summing up, we recognise the following problems to analyse and to solve. First problem concerns the efficiency of ECC systems. Its efficiency strongly depends on the efficiency of finite-field arithmetic operators. Thus we need to perform research, which will allow us to come up with very efficient hardware finite-field arithmetic units. In order to provide solution to this problem and elaborate our own efficient algorithm easily translatable to hardware it is necessary to analyse as many existing algorithms as possible.

There are two types of finite fields over which elliptic curves are defined to serve cryptographic purposes. Prime fields $GF(p)$ and binary extension fields $GF(2^m)$ [36]. Binary extension fields $GF(2^m)$ allow for carry-free operations. Thus we may avoid taking care of long carry chains. According to many sources $GF(2^m)$ fields are more suitable for hardware solutions, i.e. [111, 47]. Thus we have decided to focus on $GF(2^m)$ rather than $GF(p)$ arithmetic operators. Generally there are two operators defined in a field: addition and multiplication. All other operations (i.e. squaring, inversion) can be implemented by means of addition and multiplication. Addition in a binary field is very simple, it is a bitwise `XOR` operation. However managing large operands even during such a simple operation may yield problems. ECC applications require performing operations on operands of size 150-600 bits [32]. Multiplication is more complex and furthermore it is a modular operation (modulo specific irreducible polynomial generating

the field). It means that we need not only to perform multiplication but also reduce obtained result. There are many multiplication algorithms and their improvements presented in literature, however most are just theoretically evaluated. This means that proposed mathematical improvements might not give desired enhancements when implemented in hardware. In our work we are motivated by a possibility of finding such modifications of algorithms, which may yield real hardware improvements, i.e. energy and area savings, design acceleration (speed-up). Our goal is to provide such algorithms, which will be suitable for efficient implementation in hardware.

Second problem, which influences the structures of elaborated algorithms, is the need to secure algorithms' implementations against physical attacks (here we consider SCA). As stated by Micali and Reyzin in [68], when they first defined group of physical attacks, *"computation and only computation leaks information"*, thus our goal is that our computations leak as small amount of information useful to an adversary as possible. In fact we are not able to prevent electronic device from leaking information, however we may make the leaking information as useless as possible by controlling the behaviour of our devices to a feasible extent. We want that our solutions are as robust as possible to side channel attacks. We focus on preventing successful power analysis attacks due to the fact that they are the most popular types of SCA attacks, i.e. they receive a lot of attention from researchers and cryptanalysts [61]. Moreover according to [61] they are very powerful and can be conducted relatively easy.

The thorough analysis of finite field operations algorithms should reveal the possibilities of securing them. It should reveal their features, advantages and potentialities for inserting countermeasures. In order to counteract to possible attacks, we have to propose modifications at algorithm level as well as at the architecture level. The goal is to propose them in such a way that resulting overhead will be sensible and that they will be transferable to other hardware architectures (ASICs).

As mentioned developers usually add protections in ECC systems at curve operations level and as proven such protections usually secure only against certain families of physical attacks [53]. For example the device strongly secured against timing attacks can be very weak against power attacks and otherwise [79]. We are strongly motivated by the presumption that securing all computations performed in ECC system (finite-field operations, curve-level operations, protocol operations) allows creating a system strongly secure against most families of side-channel attacks.

Third problem, which needs to be investigated, is the trade-off between security issues and efficiency. On one hand we want the device to be very secure but on the other it still has to be very efficient. If we overload operators with security issues (countermeasures) their speed may drastically decrease and their size/cost may dramatically increase. However if we insert not enough countermeasures, cryptographic system might be easily attacked. The elaborated efficient hardware arithmetic operators units should allow for inserting countermeasures without adding much overheads to the solution (without degrading performance of the solution and increasing its cost). The impact of added countermeasures on the parameters and behaviour of the solution should be very carefully evaluated. If a countermeasure degrades speed too much or causes an explosion of its size, it should be either avoided and substituted by other or thoroughly reconsidered (and possibly improved).

The alongside problem, having impact on all the others, is the size of data to be manipulated by the operators. As they need to serve ECC purposes they need to operate on numbers of size approximately 150-600 bits [32]. Large binary vectors are not easy to handle and what is more sometimes they may cause synchronisation and routing problems, i.e. be the cause of hazards or strange delays. Usually with growth of operands size, the operator solutions grow and their speed decrease, so our objective is to provide

very efficient solutions for arithmetic operators working on vectors of large sizes.

In the following sections some cryptography basics, necessary to understand the purpose of our researches, will be presented.

1.1. Modern cryptology - basics, goals, applications and threats

In this section a short introduction to cryptography is presented. We provide brief overview of most popular cryptographic techniques and more detailed description of the techniques to which our researches will apply.

We introduce also cryptanalysis and describe briefly code breaking techniques. The short introduction to those topics is necessary to understand the objectives of our researches. More detailed introduction to some attacks is presented in Chapter 4.

1.1.1. Cryptology basics

Cryptology comprises cryptography and cryptanalysis. To introduce reader to our problem we present briefly both branches. We give here classical definitions. Presently cryptology domain concerns not only mathematics but also computer science. This is due to the fact that the modern cryptology deals with digital data and digital systems. Nowadays to use cryptographic techniques it is necessary not only to know a secure mathematical algorithm but also to efficiently implement it in a digital system.

Cryptography
Cryptography is a branch of cryptology treating about information security. It provides means for securing communication and information exchange in

presence of adversaries.

Definition 1.1.1. *(according to [66, 99])* ***Cryptography*** *is a study of mathematical* techniques related to aspects of information security such as confidentiality, data integrity, entity authentication and data origin authentication. Cryptography treats about prevention, detection of tampering and other malicious activity of which physical and digital data can suffer.*

**modern cryptography as mentioned above concerns also computer science discipline*

Modern cryptography concerns the following security objectives [66, 99]:

- confidentiality (privacy) - no information can be extracted by unauthorised entity from messages send over unsecured channel or data stored on unsecured media (in unsecured area/zones);
- authentication - a process by which one may ascertain for example data origin; comprises entity authentication and data origin authentication;
- data integrity - ensures that a message has not been tampered with (altered in unauthorised way);
- non-repudiation - the message is bound to the sender, i.e. the receiver can be sure that it comes from the sender and the sender cannot deny sending it;

The most popular cryptographic tools for providing information security are symmetric cryptography and asymmetric cryptography. Both comprise algorithms, which security bases on intractability of underlying mathematical problems and on security of a secret key. Short explanation of some basics of those algorithms is presented in next subsections.

Cryptanalysis

The second, equally interesting, branch of cryptology is cryptanalysis.

Definition 1.1.2. *(according to [66, 99])* ***Cryptanalysis*** *is a study of mathematical* techniques related to analysis of secured communication, ciphers and cryptographic systems in order to discover their weaknesses, which may allow retrieving secret data. Modern cryptanalysis treats about breaking mathematical systems as well as physical devices implementing them. It

validates cryptographic system security and points out the features, which need to be improved.
**modern cryptography as mentioned above concerns also computer science discipline*

Cryptanalysts study breaking codes, breaking cryptographic systems and recovering the secret. We may say that their task is to validate a cryptographic system. To prove that it is breakable in any way or to confirm its security level. Before popularisation of digital systems, the aim of cryptanalysts was just to find a way to solve an intractable mathematical problem. Nowadays when underlying mathematical problems are really hard to solve and the ability to solve them usually depends on available computing power, the cryptanalysts seek for other, easier, complementary and less expensive, ways of recovering the secret. Due to the fact that physical documents are being replaced by digital ones, to secure and handle them researchers/designers tend to provide efficient digital systems, either software or hardware, implementing cryptographic algorithms. Hence the cryptanalysts turn their interest to observation of designed devices and systems implementations in order to find cheaper and more effective ways of recovering secrets. Unfortunately for system designers it occurred that by observation of the behaviour of a device implementing cryptographic system: power consumption, time of execution, electromagnetic emissions, it is possible to break the system [68, 5]. It was proven that plenty of information leaking from the system might be useful to a cryptanalyst (an adversary, eavesdropper). Thus in order to create secure cryptographic system, it is necessary not only to find secure algorithm but also to be aware of possible information leakage advantageous to an adversary [4, 61].

Communication model
Figure 1.1 shows a typical communication model. In this model entity A communicates with entity B. Entity E tries to tamper the communication either by stealing exchanged messages, altering them or destroying them. The goal of cryptography is to secure communication between A and B

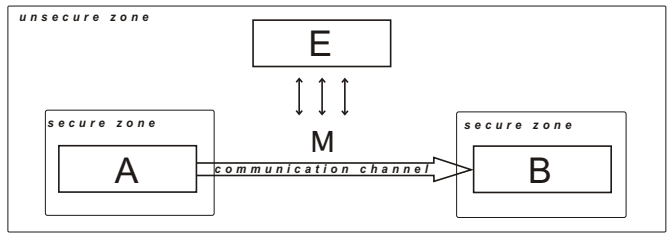


Figure 1.1.: Typical plain (not secured) communication model

against actions of E. The goal of cryptanalysis is to find a way to tamper the secured communication or to retrieve secret data (M, key).

Secure communication model

The model is illustrated on Figure 1.2. In secure communication model

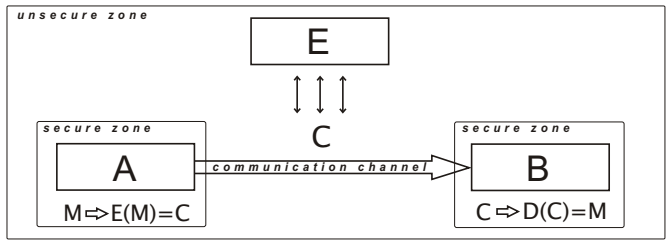


Figure 1.2.: Secure communication model

entity A, before transmitting the message to B, enciphers it. Upon receiving the ciphered message (ciphertext) entity B must decipher it to be able to read it. It should be infeasible for E to tamper the communication or to decipher message sent by A. This infeasibility should be ascertained by proper cryptographic techniques. Nowadays the most popular techniques for securing communication are key-based techniques. Key-based means that their security depends on secrecy of the key.

The cryptographic problem in this model (Figure 1.2) is how to effectively encipher the message (plaintext) to have it deciphered by B but not by E.

The idea of key-based algorithm is to rely entirely on the secrecy of a key. In such algorithms the encryption/decryption process is done in the following way:

$$plaintext \xrightarrow{K} ciphertext \xrightarrow{K} plaintext.$$

Let $K_i \in keyspace$, plaintext be denoted by M, ciphertext (ciphered plaintext) by C. Let us also denote encryption by E and decryption by D. Thus (see [66]):

$$E_{K_e}(M) = C$$
$$D_{K_d}(C) = M,$$

where E_{K_e} denotes encryption with key K_e and D_{K_d} decryption with key K_d.
Entity A transforms plaintext M (message) into a ciphertext, using encryption key K_e and transmits the ciphertext to B. Entity B receives ciphertext C and transforms it back to plaintext M, again using a key, this time decryption key K_d (somehow correlated with K_e). Depending on how we define, correlate and distribute the pair of keys we may distinguish two different key-based cryptographic techniques: symmetric cryptography and asymmetric cryptography.

1.1.2. Symmetric cryptography (Secret-Key Cryptography)

In *symmetric-key* cryptography, called also *secret*, *single*, *one-key* [66], we perform (see also Figure 1.3):

1. *Key exchange / key distribution*
2. $E_{K_e}(M) = C$
   $D_{K_d}(C) = M,$

where K_e can be calculated from K_d and otherwise [66]. In fact in this cryptographic scheme usually $K_e = K_d = K$.

In symmetric-key cryptography, before starting to communicate, A and B

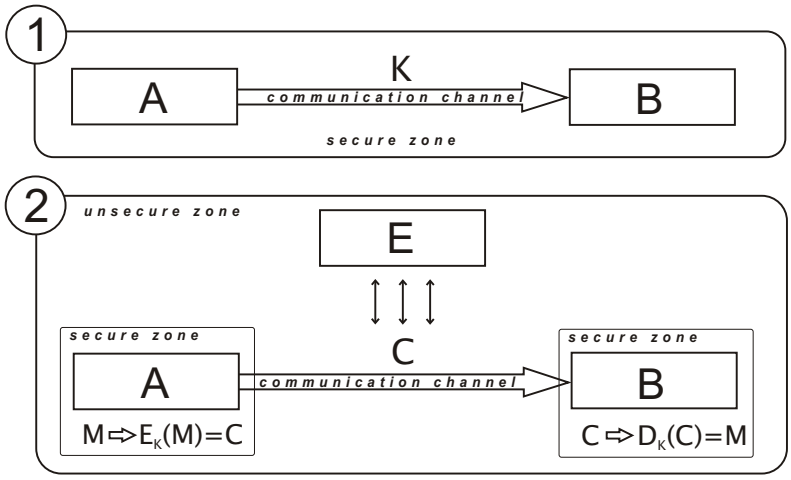


Figure 1.3.: Secret-key cryptography communication model

have to exchange secret key via some secured channel, see step 1 on Figure 1.3. The key must remain secret as long as communication has to remain secret. The problem of secure key distribution and management is crucial for symmetric key cryptography. It leads to many other problems and although secret-key cryptography is very efficient, due to key management problems it cannot be safely used in all communication schemes, especially in secure communication over the Internet. What is more secret-key cryptography does not fully implement all abovementioned cryptographic objectives (i.e. authentication, non-repudiation) [66].

The most popular symmetric-key cryptography algorithms are [36]:

- Data Encryption Standard (DES), Triple DES,
- Advanced Encryption Standard (AES),
- RC4 stream cipher (Rivest Cipher 4),
- Message Authentication Codes (MAC/HMAC).

Even though secret-key cryptographic techniques are characterised by high efficiency they cannot be used before the key is safely exchanged. To overcome this problem public-key cryptography was proposed [22, 67].

The secret-key cryptography is out of scope of our researches thus we do not present the algorithms in more details. For further reading we suggest NIST (National Institute of Standards and Technology) standards or [99, 66].

1.1.3. Asymmetric cryptography (Public-Key Cryptography)

Public-key cryptography (PKC) was introduced in 1975 by Diffie, Hellman [22] and Merkle [67] as an attempt to solve problems arising in secret-key cryptography. Definition according to Diffie and Hellman [22] is presented below (see also Figure 1.4):

Definition 1.1.3. *[22] A **Public-Key Cryptosystem** is a pair of families* $\{E_K\}_{K\in\{K\}}$ *and* $\{D_K\}_{K\in\{K\}}$ *of algorithms representing invertible transformations,*

$$E_K : \{M\} \rightarrow \{M\}$$

$$D_K : \{M\} \rightarrow \{M\}$$

on a finite message space M*, such that*

- *for every* $K \in \{K\}$*,* E_K *is the inverse of* D_K
- *for every* $K \in \{K\}$ *and* $M \in \{M\}$*, the algorithms* E_K *and* D_K *are easy to compute,*
- *for almost every* $K \in \{K\}$*, each easily computed algorithm equivalent to* D_K *is computationally infeasible to derive from* E_K*,*
- *for every* $K \in \{K\}$*, it is feasible to compute inverse pairs* E_K *and* D_K *from* K*.*

In public-key communication model, communicating entities avoid exchanging secret key. Instead of one secret key, which is hard to distribute (transmit) securely, the entities A, B use a pair of keys. One, which is pri-

vate (secret) and not transmitted; and the other, which is public and can be distributed freely. Each entity has its own pair of keys (K_e, K_d). The public-key communication scheme is as follows:

1. *Key distribution*
2. $E_{K_e}(M) = C$
   $D_{K_d}(C) = M$

where $K_e \neq K_d$ and K_e (public key) can be calculated from K_d (secret key) but K_d cannot be calculated from K_e. Depending on which entity wants to communicate, this entity distributes its public key.

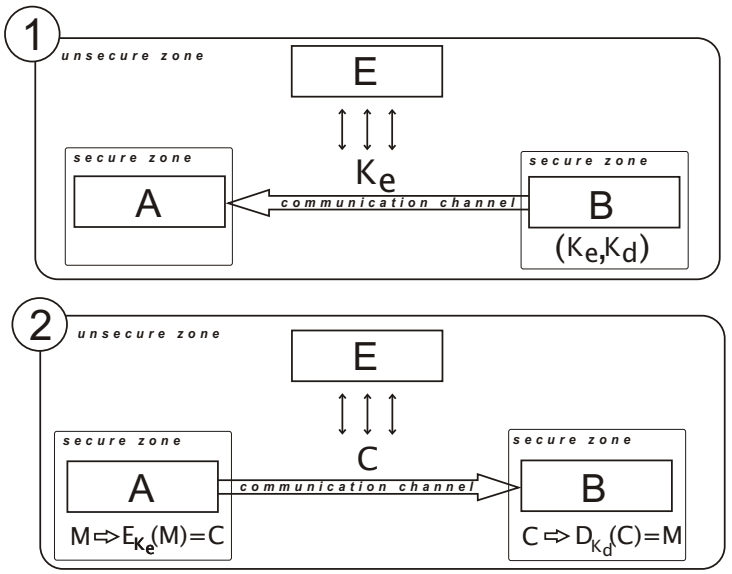


Figure 1.4.: PKC communication model

Everyone can encipher messages using key K_e but only the owner of paired key K_d is able to decrypt and read them. According to Figure 1.4 the communication is conducted as follows. If entity B wants to securely

communicate with entity A, it generates the pair of keys (K_e, K_d). It keeps K_d for itself and sends K_e to entity A. Upon receiving K_e from B, A is able to send encrypted messages to B. In order to send the message to B, A encrypts it using K_e. Entity B, receives encrypted message sent by A and in order to read it, decrypts it using key K_d. That way no one except B can read message encrypted with K_e key. In case of digital signature public key K_e is used by entity A for verification of B's signature (B's document received).

The property and simultaneously the requirement for PKC key security states that it should be computationally infeasible to compute the private key K_d from public key K_e and otherwise. The public key is used to encrypt messages and only private key can be used to decrypt them. Thus if it would be feasible to compute K_d knowing K_e it would be possible to break the system and make communication unsecure.

Although the public-key cryptography solves the problem of key management and distribution, it is slower and much harder to implement efficiently than secret-key cryptography (see Table 1.1 for comparison). Thus it is popular to use PKC for secret key exchange and later proceed with communication secured with symmetric cryptography techniques. The key pair

Table 1.1.: Comparison of secret- and public-key cryptography

Secret-Key Cryptography		Public-Key Cryptography	
Advantages	Disadvantages	Advantages	Disadvantages
high efficiency	key distribution, key management problem	solves key distribution problems	lower efficiency, higher cost due to more complex computations and longer keys
lower cost	cannot fully implement authentication and non-repudiation	fully implements all cryptographic objectives	

generation is a crucial point of asymmetric cryptography. The pair should be generated in such a way that it is infeasible to inverse the process. The private key is believed to be safe as long as a mathematical problem involved in its derivation is believed to be intractable. The following mathematical problems, infeasible to solve for certain sizes of arguments, form bases for security of private key:

- Integer factorisation problem
- Discrete logarithm problem (DLP)
- Elliptic curve discrete logarithm problem (ECDLP)

Regarding the underlying mathematical problems one can distinguish three groups of algorithms. The most popular algorithms based on integer factorisation problem are RSA public-key encryption and signature schemes [54]. ElGamal cryptographic schemes [23] exploit discrete logarithm problem. The last group of algorithms based on elliptic curves exploiting ECDLP [36] problem is of most concern to us. Thus in Chapter 2 we provide more detailed description of elliptic curve cryptographic techniques and schemes.

In our researches we have decided to focus on ECC because it is proven that it can be more efficient than RSA [109, 56, 36], which is the most popular PKC scheme. In key-based cryptography where security depends on a key the infeasibility of computing it from publicly known data is crucial. It is recognised that the abovementioned mathematical problems are feasible to solve for some arguments (usually small but also for certain types of arguments). To make the problems infeasible to solve the mathematicians proposed the arguments to be primes of specific sizes. For too small primes the accessible computational power is enough to solve the problems in reasonable time. The safe, suitable for cryptographic purposes, argument (key) sizes are given in cryptographic standards (e.g. NIST , SECG). The standards are often verified by cryptanalysts and updated if the computational power, which continuously grows, becomes enough to break the cryptographic algorithm secured with a key of a certain size or if new type

of attack, which makes retrieving the secret feasible, appears. The key sizes, for which RSA achieves the same security level as ECC, are much bigger than the ones required for ECC. For example, RSA key size of 3072 bits gives equivalent security level as ECC key of size 256 bits [109]. More detailed comparison of different key-based techniques and their security levels depending on the key size is presented below.

Comparison of security strength of different cryptographic key-based techniques

Table 1.2 (according to [109]) confirms and explains the abovementioned advantages and disadvantages of all presented types of cryptographic techniques. We can clearly see why one technique is more efficient than the other. The key sizes for symmetric encryption algorithms are much smaller than the ones used in asymmetric encryption schemes. It is especially visible when we compare key sizes of RSA with symmetric key sizes. The difference between key sizes providing equivalent security strength for ECC and symmetric algorithms is much smaller. That feature makes ECC very attractive. With smaller keys the computations are simpler and faster, thus also the computational devices are smaller and less demanding.

Table 1.2.: Comparison of key sizes [109]

security (bits)	symmetric encryption algorithm	minimum size (bits) of Public-Key		
		DSA/DH	RSA	ECC
80	Skipjack	1024	1024	160
112	3DES	2048	2048	224
128	AES-128	3072	3072	256
192	AES-192	7680	7680	384
256	AES-256	15360	15360	512

1.1.4. Modern cryptosystems - application, requirements, security (robustness)

Definition 1.1.4. *(according to [66, 99])* ***Cryptosystem*** *is a set of cryptographic algorithms with all possible ciphertexts, plaintexts, keys and key management processes. It is a set of cryptographic techniques (primitives) used to provide security services for communication over unsecured channel.*

Nowadays we perceive cryptosystem as an embedded digital system implementing cryptographic primitives in order to provide information security. Before digital information era, security of information depended on the manner in which we have sealed our document, on type of media we have used to record and pass the message, and usually on communication channel (messenger, furnisher). Due to digitalisation of data and popularity of digital techniques and networks high percent of confidential transactions became digital. The electronic cash transaction, electronic confidential documents exchange (tax data, health data), communication with banks and important offices, it all becomes more and more popular. With growth of popularity of digital data exchange, grows the need to secure such communications. The digital documents exchange is usually done over Internet, which is a very demanding, unsecured communication channel. The cryptographic techniques evolve to fulfill the arising requirements and their implementations adapt to new conditions.

Applications The applications of digital cryptosystems spread many domains. The first and the most popular is securing data exchange in communication over Internet. The number of services possible to do over Internet still grows. The most popular ones are: messages exchange (e-mail), banking transactions (electronic credit and debit card transactions, bank account management), all transactions involving electronic cash, e-commerce, digital signatures, business transactions, communications with offices (e.g. tax office) and many many more.

Other applications involve not only data exchange but also data storing.

The digital data importance grows. Many people and companies start to rely mostly on digital documents and data, instead of keeping many useless paper copies. Many jobs now are performed using computers and many people's job depends on the security of data stored either on hard drives or somewhere over Internet. We start to deposit our data on external servers thus they can be more vulnerable to unauthorised actions. The so called "cloud computing" service providing computing power and storage capacity, becomes very popular. Therefore our data should be secured/encrypted before transmitting/depositing it somewhere over the Internet. The loss or unauthorised alteration of such data may cause huge problems to a company and similarly to a common user.

What is more, many offices and institutions tend to digitalise their databases, e.g. to ease the access to it. In hospitals and clinics the vital medical data have to be secured properly to avoid stealing or tampering. The same applies to tax offices, the tax data need to be secured properly to avoid embezzelments.

Another problem to which cryptosystems can be applied is a wireless communication. Number of wireless applications communicating grows rapidly thus also the demands for its quality, i.e. speed, range, security. Wireless communication is especially easy to eavesdrop or tamper. To do this the adversary does not even need to have direct access to the communicating entities [74].

What is more, nowadays, with modernisation of healing techniques, there arise a need to secure medical appliances. Besides usual medical apparel hard to disturb without direct access to them, there were developed a microchip devices delivering drugs [42], which can be used instead of regular injections. Such a device is implanted in a patient and is responsible for oozing out the right dose of a drug in proper time intervals. If the microchip work would be disturbed due to external malicious actions, it could cause irreversible damage of one's health.

Finally, the most obvious application: military application. Cryptosys-

tems apply to almost all areas in military domain. They are responsible for securing information exchange between governments, for distribution of confidential orders, etc. They provide means for securing remote controls of military equipment (for example: rocket launcher), for securing flow of information between units in order to avoid being eavesdropped or discovered and many, many more.

Requirements Depending on the application the requirements vary. However the digital cryptosystem should always fulfill the following objectives in order to serve any application.
The proper cryptosystems should be:

- very efficient (fast, small, not very demanding when it comes to power consumption)
- mathematically robust (they should use up-to-date specifications of cryptographic systems)
- physically robust (they should be secure against eavesdropping and tampering)
- adaptable (they should properly work in given environment - depends on application)

Characteristics of a good cryptosystem:

- *theoretical/mathematical security* - hardness of underlying mathematical problem,
- *key length* - the smaller the key the easier the computation,
- *speed-efficiency* of encryption/decryption process,
- *implementation* - efficiency of implementation,
- *scalability* - "the unit can be reused or replicated in order to generate long precision result independently of the data path precision for which the unit was originally designed" [8]
- *interoperability* - ability to exchange information with external sources.
- *physical security* - security against side channel attacks, security of a device

Robustness Security strength of an cryptographic algorithm depends on quality of the algorithm and underlying mathematical problem, length of the key and nowadays also on quality of the implementation of the algorithm or we may say robustness of the cryptographic device (device performing cryptographic operations). The cryptanalysts describe the security of the system using the notion of *level of security*. Level of security is usually given in terms of the amount of work (number of operations), using the best methods currently known, needed to be performed to break the system [66].

Figure 1.5 presents different layers of a cryptosystem. Each of these

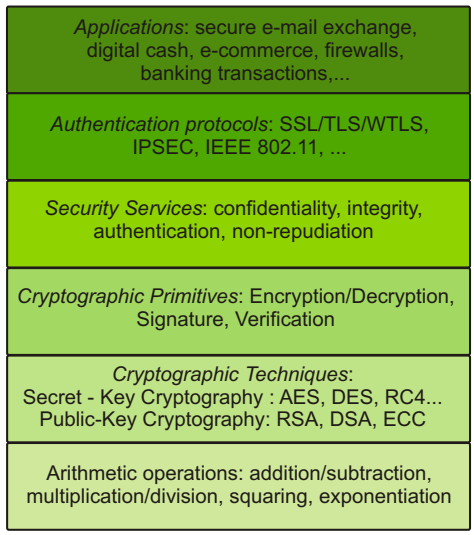


Figure 1.5.: Security layer model [8, 98]

layer should be somehow secured in order to obtain a secure communication scheme. For us the most interesting is the bottom layer. For ECC it can be divided further, see Figure 1.6. It can be divided into three parts (sub-layers):

- $[k]P$ sub-layer - multiplication of the base point of the curve by a large scalar $[k]$ (key, secret),

- $2P$, $P + Q$ (doubling, addition) - operations on points of the curve,
- arithmetic operations in $GF(2^m)$ - operations on coordinates of the points, on elements of the underlying field.

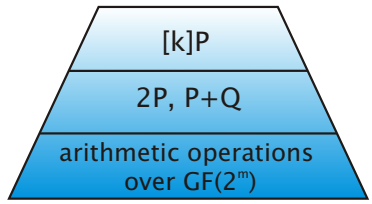


Figure 1.6.: ECC cryptosystem layers

There are already known techniques for securing the first two sub-layers. On some we were working together in IRISA laboratory (Lannion, France) with other PhD student Thomas Chabrier [15]. However there are not yet known any propositions for securing at the arithmetic level the operations performed on the elements of the underlying field.

1.2. Dissertation overview

In the next chapter, we will provide a short introduction to elliptic curves for use in cryptography and elliptic curve cryptography techniques. Then we will explain the arithmetic in finite fields and provide more details about binary extension fields $GF(2^m)$. Finally we will formulate the main thesis we want to prove with our researches. Third chapter contains detailed description of hardware arithmetic operators elaborated during the researches. Followingly the subsequent chapter introduces the side channel attacks, especially the power analysis attacks and presents our ideas for securing the previously described hardware arithmetic operators against them. Eventually we summarise our work, draw conclusions and present future prospects.

2. Elliptic curves over finite fields - application to cryptography (overview)

In this chapter we present brief overview of the most important, from cryptographic point of view, properties of elliptic curves and finite fields. We present their application to modern cryptography, which is of most interest to us. We give a short overview of the application of finite fields to elliptic curve cryptography. We will try to show what is the impact of finite-field arithmetic operators on ECC system, how important those operators are for the computations performed by the ECC system.

All presented here elliptic curve theory is based on [49, 102, 55, 10, 65]. Finite field description is written according to [59, 58, 64, 96, 48]. Those sets of references contain complete knowledge about elliptic curves and finite fields.

2.1. Elliptic curves and cryptography

Elliptic curves were studied long before they were introduced to cryptography. In 1985, independently Neal Koblitz [49] and Victor Miller [69] proposed to use them in public-key cryptographic systems due to their specific properties. It occurs that the problem on which the security of most popular public-key techniques depends, i.e. the discrete logarithm problem (DLP), defined for elliptic curves (ECDLP) is more complex than in usual

case (in case of DLP). Elliptic curve cryptography techniques were popularised in 90's. Their use in security applications have been approved and recommended by many. Their attractiveness lies especially in fact that to achieve the same security level as RSA, they require much smaller keys i.e. they operate on much smaller numbers, see Table 1.2 on page 20 for comparison. The smaller are the numbers on which the arithmetic units operate the simplest (the smallest, the fastest) the final cryptographic device.
In the following sections we briefly introduce elliptic curve arithmetic, then present their application to security schemes. The ECDLP problem, guarding security of ECC protocols, will also be explained along with the description of few ECC security schemes.

Understanding elliptic curve arithmetic is not necessary to be able to provide efficient $GF(2^m)$ arithmetic units. However it is crucial when we want to add protections against SCA to those units. We ought to be conscious, which operations need to be secured and in what way they can be insecure or vulnerable to attacks.

2.1.1. Elliptic curves

Definition 2.1.1. *(according to [24]) An* ***elliptic curve*** *E over a field K can be defined by* ***Weierstrass equation*** *of the form:*

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

where $a_1, a_3, a_2, a_4, a_6 \in K$.

The following quantities are related to E:

$$\begin{aligned} \Delta &= -b_2^2b_8 - 8b_4^3 - 27b_6^2 + 9b_2b_4b_6 \\ j &= \frac{c_4^3}{\Delta} \qquad for \quad \Delta \neq 0 \\ b_2 &= a_1^2 + 4a_2 \end{aligned}$$

$$b_4 = 2a_4 + a_1a_3$$
$$b_6 = a_3^2 + 4a_6$$
$$b_8 = a_1^2a_6 + 4a_2a_6 - a_1a_3a_4 + a_2a_3^2 - a_4^2$$
$$c_4 = b_2^2 - 24b_4.$$

Element Δ is called discriminant of E and determines whether the Weierstrass equation is singular or not, j is its j-invariant. The quantities b_i and c_i are defined to simplify the definition of Δ. K is called the underlying field and can be the field $\mathbb{R}$ of real numbers, $\mathbb{Q}$ rational numbers, $\mathbb{C}$ complex numbers or $\mathbb{F}_q$ finite field. If E is defined over K then it is defined over any extension of K. An elliptic curve E defined over a field K can be also denoted as E/K.

The set of points of an elliptic curve E defined over any extension L of field K forms an abelian group and is defined in the following way:

$$E(L) = \Big\{(x,y) \in L \times L : y^2 + a_1xy + a_3y - x^3 - a_2x^2 - a_4x - a_6 = 0 \cup \{\infty\}\Big\},$$

where ∞ is a point at infinity. The *elliptic curve over* K is the set of points (x, y) satisfying a Weierstrass equation. Depending on the underlying field K, the equation 2.1 can be simplified. During our researches, we focus on elliptic curves defined over finite fields of characteristic 2 ($GF(2^m)$). For $GF(2^m)$ the basic Weierstrass equation defining elliptic curve may be simplified as follows from Definition 2.1.1.

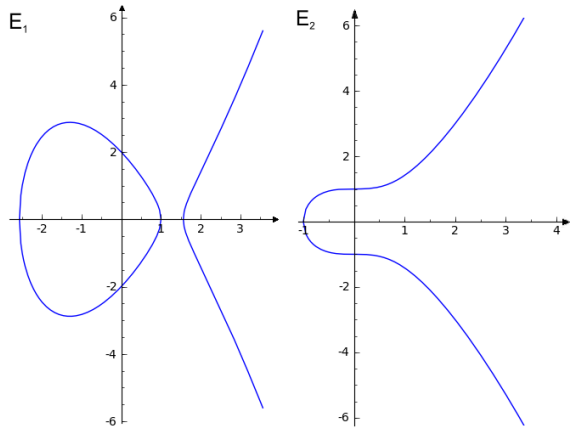
Definition 2.1.2. *If K is a finite field of characteristic 2 ($K = GF(2^m)$) then E/K can be defined by:*

$$E_1 : y^2 + cy = x^3 + ax + b, \quad for\ a = 0, \Delta = c^4 \neq 0 \quad \textit{(supersingular)} \tag{2.2}$$

or

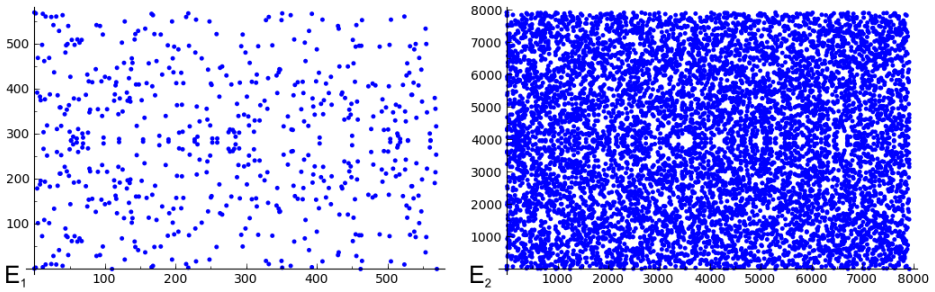
$$E_2 : y^2 + xy = x^3 + ax^2 + b, \quad for\ a \neq 0, \Delta = b \neq 0 \quad \textit{(non-supersingular)} \tag{2.3}$$

All the arithmetic principles of elliptic curves are best visualised geometrically on elliptic curves defined over $\mathbb{R}$. Below we present graphs of curves defined over $\mathbb{R}$ (Figure 2.1) as well as curves defined over prime finite fields (Figure 2.2). The exemplary curves were plotted using SAGE.



$E_1 : y^2 = x^3 - 5x + 4$ $\qquad$ $E_2 : y^2 = x^3 + 1$

Figure 2.1.: Elliptic curves over $\mathbb{R}$.



$E_1(\mathbb{F}_{571}) : y^2 = x^3 + 1$ $\qquad$ $E_2(\mathbb{F}_{7919}) : y^2 = x^3 + 7914x + 4$

Figure 2.2.: Elliptic curves over $\mathbb{F}_p$.

Group Law (according to [36]) The basic operation on elliptic curve group is point addition. It is best explained geometrically with *chord-and-tangent rule* for elliptic curves defined over $\mathbb{R}$. Let $P(x_1, y_1)$, $Q(x_2, y_2)$, $R(x_3, y_3)$

be three distinct points on $E(K)$ ($x_i, y_i \in K$) such that Equations 2.2/ 2.3 hold. Then

Additive identity
If P is the point at infinity, i.e. $P = \infty$, then $-P = \infty$ and $P + Q = Q$. Point ∞ (zero element) serves as additive identity of the group of points

Negatives
The negative $-P$ is on the curve whenever P is. The point $-P$ has the same x-coordinate as P but negative y-coordinate, i.e. $-(x_1, y_1) = (x_1, -y_1)$. The addition $P + (-P)$ gives as a result point at the infinity.

Addition of two distinct points P, Q (see Figure 2.3 left part)
Let $R \in E(K)$ be the result of $\mathbf{P} + \mathbf{Q}$. To obtain R we draw a line through P and Q. The third point, at which this line intersects $E(K)$ is the reflection about x-axis of the sum R.

Point addition algebraic formula for non-supersingular $E(\mathbb{F}_{2^m}) : y^2 + xy = x^3 + ax^2 + b$

$$x_3 = \lambda^2 + \lambda + x_1 + x_2 + a \qquad y_3 = \lambda(x_1 + x_3) + x_3 + y_1,$$

$$\text{where } \lambda = \frac{(y_1+y_2)}{(x_1+x_2)}$$

Point addition algebraic formula for supersingular $E(\mathbb{F}_{2^m})$: $y^2 + cy = x^3 + ax + b$

$$x_3 = \lambda^2 + x_1 + x_2 \qquad y_3 = \lambda(x_1 + x_3) + y_1$$

$$\text{where } \lambda = \frac{(y_1+y_2)}{(x_1+x_2)}$$

Doubling P (see Figure 2.3 right part)
Let $Q \in E(K)$ be the result of **2P** operation. To obtain Q we draw a line tangent to elliptic curve at P. The point, at which this line intersects $E(K)$ is the reflection about x-axis of the resulting point Q.

Point doubling algebraic formula for non-supersingular $E(\mathbb{F}_{2^m}): \; y^2 + xy = x^3 + ax^2 + b$

$$x_3 = \lambda^2 + \lambda + a = x_1^2 + \frac{b}{x_1^2} \qquad y_3 = x_1^2 + \lambda x_3 + x_3$$

$$\text{where } \lambda = x_1 + \frac{y_1}{x_1}$$

Point doubling algebraic formula for supersingular $E(\mathbb{F}_{2^m})$: $y^2 + cy = x^3 + ax + b$

$$x_3 = \lambda^2 \qquad y_3 = \lambda(x_1 + x_3) + y_1 + c$$

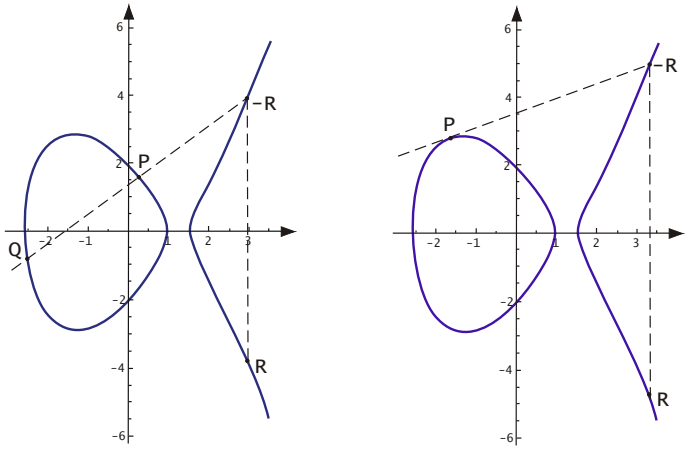
$$\text{where } \lambda = (\frac{x_1^2 + a}{c})$$

Point on elliptic curve can be represented using different types of coordinates. Each type has his advantages and disadvantages. For instance, projective coordinates does not require inversion when performing operations on elliptic curve points [36]. All the above formulas are derived for curves described by affine coordinates. For other types of coordinates: projective, Jacobian, mixed, etc., those formulas are different [36].

For more details about elliptic curves we suggest reading [49, 102, 55, 10, 65].

2.1.2. Elliptic Curve Cryptography

The elliptic curve cryptographic techniques exploit properties of elliptic curves defined over finite fields $\mathbb{F}_q$. The elliptic curve cryptography schemes depend on the hardness of elliptic curve discrete logarithm problem (below



Addition $P + Q = R$ Doubling $2P = R$

Figure 2.3.: Addition and Doubling of a point on $E(K)$

we present definition from [36]).

Definition 2.1.3. ***Elliptic Curve Discrete Logarithm Problem (ECDLP) [36]*** *Given an elliptic curve E defined over finite field $\mathbb{F}_q$, a point $P \in E(\mathbb{F}_q)$ of order n, and a point $Q \in \langle P \rangle$, find the integer $l \in [0, n-1]$ such that $Q = lP$. The integer l is called the* discrete logarithm of Q to the base P, *denoted* $\mathbf{l = \log_P Q}$.

Elliptic curve domain parameters D: q - field order; FR - field representation; S - seed; $a, b \in \mathbb{F}_q$, which define the equation of elliptic curve E; point $P(x_p, y_p) \in \mathbb{F}_q$; *order* n of P; *cofactor* $h = \#E(\mathbb{F}_q)$, to be used in cryptography are usually defined in standards (NIST [32], SECG [92, 93]). Only for specific values of those parameters the cryptographic schemes resist all known mathematical attacks on ECDLP.

Exemplary ECC security schemes The ECC is used in many cryptographic schemes. We will provide some details of how some schemes work

and give exemplary algorithms. Our goal is to point out the operations in elliptic curve based security schemes, which are the attackers target.

The most important algorithm used in all types of public-key schemes is the key pair (Q, d) generation, where Q is a public key and d is the corresponding private key. On the secrecy of the key d depends the security of cryptographic techniques/schemes.

Algorithm 1 Key pair generation [36]

Input: Domain parameters $D = \{q, FR, S, a, b, P, n, h\}$.
Output: Public key Q, private key d.
1: Select $d \in_{\mathbb{R}} [1, n-1]$
2: Compute $Q = dP$
3: Return (Q, d)

The computation of d having Q and P is the elliptic curve discrete logarithm problem. As the problem for properly chosen domain parameters D is intractable the security of d is ensured.

Signature scheme

Signature schemes are used to sign digital documents in the same way as handwritten signatures are used to sign paper documents. With them we can provide the following security services: authentication, data integrity and non-repudiation.

The signature scheme consists of the following steps [36]:

1. *Domain parameter generation* - to perform any of the next steps, we need set $D = \{q, FR, S, a, b, P, n, h\}$; for cryptographic purposes those sets are defined in standards: NIST [32], SECG [92, 93];
2. *Key pair generation* - generation of key pair $\{Q, d\}$, see Algorithm 1;
3. *Signature generation* - generation of a signature Σ of message m, using set D and private key d (see Algorithm 2);
4. *Signature verification* - signature is verified, using set D, public key Q, and received signature Σ, in order to reject or accept incoming message m, see Algorithm 3.

One of the most popular scheme is Elliptic Curve Digital Signature Algorithm (ECDSA).

Algorithm 2 ECDSA signature generation [36]

Input: Domain parameters $D = \{q, FR, S, a, b, P, n, h\}$, private key d, message m

Output: Signature (r, s)

1: Select $k \in_{\mathbb{R}} [1, n-1]$
2: Compute $kP = (x_1, y_1)$ and convert x_1 to an integer $\overline{x_1}$
3: Compute $r = \overline{x_1} \bmod n$. If $(r = 0)$ go to step 1.
4: Compute $e = H(m)$ // H is a hash function //
5: Compute $s = k^{-1}(e + dr) \bmod n$. If $(s = 0)$ go to step 1.
6: Return (r, s)

Algorithm 3 ECDSA signature verification [36]

Input: Domain parameters $D = \{q, FR, S, a, b, P, n, h\}$, public key Q, message m, signature (r, s)

Output: Acceptance or rejection of the signature

1: Verify that r, s are integers in the interval $[1, n-1]$. If *verification=fail* then return "reject the signature"
2: Compute $e = H(m)$ // H is a hash function //
3: Compute $w = s^{-1} \bmod n$
4: Compute $u_1 = ew \bmod n$ and $u_2 = rw \bmod n$
5: Compute $X = u_1P + u_2Q$. If $(X = \infty)$ then return 'reject the signature"
6: Convert the x-coordinate x_1 of X to an integer $\overline{x_1}$; Compute $v = \overline{x_1} \bmod n$
7: If $v = r$ return (r, s)

The other popular elliptic curve signature scheme is Elliptic Curve Korean Certificate-based Digital Signature Algorithm (EC-KCDSA). For more details see standards: ANSI X9.62 see [2], FIPS 186-3 see [32], IEEE 1363-2000 see [3], ISO/IEC 15946-2 see [1].

Public-key encryption schemes

Public-key encryption schemes provide confidentiality service. It comprises the following steps [36]:

1. *Domain parameter generation* - to perform the scheme, we need set $D = \{q, FR, S, a, b, P, n, h\}$; for cryptographic purposes those sets are defined in standards: NIST [32], SECG [92, 93];
2. *Key pair generation* - generation of key pair $\{Q, d\}$, see Algorithm 1;
3. *Encryption* - encryption of a message m, using set D and public key Q, preparation of ciphertext c, see Algorithm 4;
4. *Decryption* - either rejects the ciphertext as invalid or produces plaintext m using domain parameters D, private key d, and received ciphertext c, see Algorithm 5; it is assumed that D and Q are valid. The decryption algorithm always accepts (D, d, c) and outputs m if c was indeed generated by the encryption algorithm on input (D, Q, m).

As an example we will provide algorithms used in elliptic curve analogue of ElGamal encryption scheme (see Algorithms 4, 5). Other popular elliptic curve based public key encryption schemes are Elliptic Curve Integrated Encryption Scheme (ECIES), see [101], and Provably Secure Encryption Curve Scheme (PSEC), see [78].

Algorithm 4 Basic ElGamal elliptic curve encryption [36]

Input: Domain parameters $D = \{q, FR, S, a, b, P, n, h\}$, public key Q, message m

Output: Ciphertext (C_1, C_2)

1: Represent the message m as a point M in $E(\mathbb{F}q)$.
2: Select $k \in_{\mathbb{R}} [1, n-1]$
3: Compute $C_1 = kP$
4: Compute $C_2 = M + kQ$
5: Return (C_1, C_2)

Observing the structures of Algorithms 1, 2, 4, one can spot that if the values d, k will be known to an adversary the cryptographic schemes will not serve their purpose anymore. Knowing the algorithm and those values an adversary will be able to negatively affect the communication.
Finding those values mathematically is equivalent to solving ECDLP prob-

Algorithm 5 Basic ElGamal elliptic curve decryption [36]

Input: Domain parameters $D = \{q, FR, S, a, b, P, n, h\}$, public key Q, ciphertext (C_1, C_2)

Output: Message m

1: Compute $M = C_2 - dC_1$

2: Return m

lem, which is intractable for certain sets of elliptic curve domain parameters. Unfortunately except theoretical attacks, there exist physical attacks. By analysis of the behaviour of the device performing cryptographic operations it is possible to discover the secret values, in ECC case, the values such as the private key d or k (see algorithms in this section). Thus it is necessary to secure all operations involving values d and k.

2.2. Finite Fields

The general theory of finite fields starts in the beginning of 19th century with works of Carl Friedrich Gauss (1777–1855) and Evariste Galois (1811–1832). We will introduce the most important algebraic theories. For a complete introduction to finite fields we suggest reading [59, 58, 64, 96, 48]. The contents of this section are based on those references.

Groups [59]

Definition 2.2.1. *A **group** is a set G together with binary operation * on G such that following properties hold:*

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

*If for all $a, b \in G$, $a * b = b * a$, then the group is called **abelian** (commuta-*

tive).

Definition 2.2.2. *A multiplicative group G is called **cyclic** if there is an element $a \in G$ such that for any $b \in G$ there is some integer j with $b = a^j$. The element a is called a **generator of the cyclic group**, and we note $G =< a >$. Every cyclic group is commutative.*

Definition 2.2.3. *A group is called **finite** (resp. infinite) if it contains finitely (resp. infinitely) many elements. The **number of elements** in a finite group is called its **order**. We shall write: $|G|$ for the order of the finite group G.*

Rings [59]

Definition 2.2.4. *A **ring** $(R, +, \cdot)$ is a set R, together with two binary operations, denoted by $+$ and $\cdot$, such that:*

1. *R is an abelian group with respect to $+$*
2. *$\cdot$ is associative - that is, $(a \cdot b) \cdot c = a \cdot (b \cdot c)$ for all $a, b, c \in R$.*
3. *The distributive laws hold: that is, for all $a, b, c \in R$ we have $a \cdot (b+c) = a \cdot b + a \cdot c$ and $(b + c) \cdot a = b \cdot a + c \cdot a$.*

Element 0 (the zero element) is the identity element of the abelian group R with respect to addition. Element $-a$ is the additive inverse of a. Rings can be classified as follows:

Definition 2.2.5. *Rings classification*

1. *A ring is called a **ring with identity** if the ring has a multiplicative identity - that is, if there is an element e such that $a \cdot e = e \cdot a = a$ for all $a \in R$.*
2. *A ring is called **commutative** if $\cdot$ is commutative.*
3. *A ring is called an **integral domain** if it is a commutative ring with identity $e \neq 0$ in which $ab = 0$ implies $a = 0$ or $b = 0$.*
4. *A ring is called a **division ring** (or skew field) if the nonzero elements of R form a group under $\cdot$ operation.*

5. ***A commutative division ring is called a field.***

Fields [59]

Definition 2.2.6. *A **field** is a set F with two binary operations, addition and multiplication, containing two distinguished elements 0 (zero element) and e (identity element) with $0 \neq e$.*
A field F is an abelian group with respect to addition having 0 as the identity element. The elements of F that are $\neq 0$ form an abelian group with respect to multiplication with e as the multiplicative identity element, usually denoted by 1. Addition and multiplication are characterised by the following distributive laws $a \cdot (b + c) = a \cdot b + a \cdot c$, $(b + c) \cdot a = b \cdot a + c \cdot a$.

Definition 2.2.7. *Extension field*
*Let F be a field. A subset K of F that is itself a field under the operations of F will be called a **subfield** of F. Then, F is called an **extension (field)** of K. If $K \neq F$, then K is a proper subfield of F. If K is a subfield of the finite field F_p, p prime, then K must contain the elements 0 and 1, and all other elements of F_p by the closure of K under addition. It follows that F_p contains no proper subfields.*

Definition 2.2.8. *Field as a vector space*
*If L is an extension field of K, then L may be viewed as a **vector space** over K. The elements of L ("vectors") form an abelian group under addition. Moreover, each "vector" $\alpha \in L$ can be multiplied by a "scalar" $r \in K$ so that $r\alpha$ is again in L ($r\alpha$ is simply the product of the field elements r and α of L) and the laws for multiplication by scalars are satisfied: $r(\alpha + \beta) = r\alpha + r\beta$, $(r + s)\alpha = r\alpha + s\alpha$, $(rs)\alpha = r(s\alpha)$, and $1\alpha = \alpha$, where $r, s \in K$ and $\alpha, \beta \in L$.*

Definition 2.2.9. *Existence and uniqueness*
*The **order** of a finite field is the **number of elements** in the field. There exists a finite field F of order q if and only if q is a prime power, i.e.,*

$q = p^n$. If $n = 1$, then F is called a prime field. If $n = 2$, then F is called an extension field. For any prime power q, there is essentially only one finite field of order q; informally, this means that any two finite fields of order q are structurally the same except that the labeling used to represent the field elements may be different. We say that any two finite fields of order q are isomorphic and denote such a field by F_q.

Number of elements of a field. [59]

Theorem 2.2.1. *Let F be a finite field. Then F has p^n elements, where the prime p is the **characteristic of** F and n is the degree of F over its prime subfield.*
***Proof**. Since F is finite, its characteristic is a prime p according. Therefore the prime subfield K of F is isomorphic to F_p and thus contains p elements.*

Constructing finite fields. [59]
Starting from the prime fields F_p, we can construct other finite fields by the process of root adjunction. If $f \in F_p[x]$ is an irreducible polynomial over F_p of degree n, then by adjoining a root of f to F_p we get a finite field with p^n elements.

Bases of the finite field. [59, 36]

Definition 2.2.10. *We regard a finite extension $F = F_{q^m}$ of the finite field $K = F_q$ as a vector space over K. Then F has **dimension** m over K, and if $\{\alpha_1, ..., \alpha_m\}$ is a **basis** of F over K, each element $\alpha \in F$ can be uniquely represented in the form*
$\alpha = c_1\alpha_1 + \cdots + c_m\alpha_m$ with $c_j \in K$ for $1 \leq j \leq m$.

Definition 2.2.11. *Let K be a finite field and F a finite extension of K. Then two bases $\{\alpha_1, ..., \alpha_m\}$ and $\{\beta_1, ..., \beta_m\}$ of F over K are said to be **dual (or complementary) bases** if for $1 \leq i, j \leq m$ we have:*

$$Tr_{F/K}(\alpha_i, \beta_j) = \begin{cases} 0 & \text{for } i \neq j \\ 1 & \text{for } i = j \end{cases}$$

Trace
The trace function $Tr_{F/K}$ serves for a description of all linear transformations from F into K. For more detailed information we suggest Chapter 3 of [58].

Definition 2.2.12. *Let $K = F_q$ and $F = F_{q^m}$. Then a basis of F over K of the form $\{\alpha, \alpha^q, ..., \alpha^{q^m}\}$ consisting of a suitable element $\alpha \in F$ and its conjugates with respect to K, is called a* ***normal basis*** *of F over K.*

2.2.1. Binary finite field extensions $GF(2^m)$

The two most popular finite fields used in cryptography are prime fields $GF(p)$, where p is prime, and finite fields of characteristic 2 (binary extension field) $GF(2^m)$, denoted also by $\mathbb{F}_{2^m}$. The described research work in this thesis, concerns the second type of fields that is binary extension fields $GF(2^m)$. Binary extension fields $GF(2^m)$ are considered advantageous for hardware solutions because their elements are represented by polynomials (binary polynomials) instead of integers and polynomial addition and modular reduction are regarded as simpler than operations on integers [113].

To construct a binary finite field extension an irreducible polynomial $f(x)$ over $GF(2)$ of degree m is used, it is assumed that α is its root, i.e. $f(\alpha) = 0$. The irreducible polynomial is of the form :

$$f(x) = x^m + f_{m-1}x^{m-1} + \cdots + f_2x^2 + f_1x + 1, \tag{2.4}$$

where $f_i \in GF(2)$.

The field can be viewed as a vector space, which elements are represented with a use of a specific basis. The most commonly used basis, of the form $\{1, \alpha, \alpha^2, \ldots, \alpha^{m-1}\}$, is called polynomial (canonical) basis of the field. The

elements of the finite field $GF(2^m)$ represented in this basis are as follows:

$$A = \sum_{i=0}^{m-1} a_i \alpha^i = a_{m-1}\alpha^{m-1} + a_{m-2}\alpha^{m-2} + \cdots + a_2\alpha^2 + a_1\alpha^1 + a_0\alpha^0, \quad (2.5)$$

where $a_i \in GF(2) = \{0, 1\}$ and α is a root of $f(x)$. Thus we may say that all elements of $GF(2^m)$ are polynomials of degree at most $(m-1)$.

2.3. Problem definition

This section will provide the reader with problems of our interest existing in described domain, which for us seems worth solving. There are two main objectives when creating a cryptographic system:

1. The system has to be very **efficient** in terms of speed, size (implementation cost), power consumption, performance.
2. The system should be **secure** against attempts of stealing the secret data.

Efficiency of cryptographic system Modern cryptographic systems have numerous applications. In order not to negatively impact the performance of larger electronic systems they need to be integrated with, they should be very efficient. By efficiency we mean that they are fast, relatively small and effective. That they protect our data without slowing down other operations and disturbing our work.

As the data size on which cryptographic systems operate constantly grows, we need to constantly adapt existing cryptographic systems to arising needs. In ECC systems all operations depend on the operations performed by finite-field arithmetic operators. If their performance is inferior they negatively impact the work of the whole cryptosystem. Even if highly efficient higher-level solution will be provided (see Figure 1.6 for what is a higher-level

operation), the system will fail due to poor performance of the most important modules: arithmetic operators.

It is not easy to create a really efficient solution. We have to take into account many things, such as finite field elements representation, arithmetic operation algorithm, size of operands (for ECC solutions they are of size 150–600 bits), if it is worth and possible to parallelise the algorithm. It is necessary to find a trade-off between size and speed of a solution. It is possible to create extremely fast solutions but huge and otherwise. The problem is to balance those two parameters so that the overall efficiency/performance and cost will be satisfying.

Security of cryptographic system If we will implement the elliptic curve cryptographic system according to requirements stated in the newest standards we can be sure that our cryptographic system is safe against theoretical (mathematical) attacks. However as mentioned in the introduction, there were developed attacks called side channel analysis attacks. SCA attacks exploit the correlation between behaviour of the cryptographic device, such as power consumption, emitted electromagnetic field, timing of the operation, and the secret data, such as private keys, on which the device operates. The SCA is a very serious threat for a cryptographic system. The weak implementations of cryptographic techniques allow even not well skilled adversary to discover a private key. With the discover/reveal of private key the whole communication system becomes insecure.

Recently there have been proposed many methods for securing ECC systems, however they focus on the top layers of the system; that is on operation $[k]P$ (multiplication of the point on a curve P by a scalar k) and primitive operations on points of elliptic curves ($2P$, $P + Q$). We presume that securing just curve-level operations of ECC system is not enough. For simple side channel analysis, where just one power trace sample is analysed, it is sufficient. However in case of differential power analysis (where hundreds of samples are analysed) such countermeasures may be too weak. We do

not know any sources discussing countermeasures for finite field arithmetic operators. The topic seems not yet exploited.

We presume it is necessary to secure all layers of a cryptographic system to make it really secure. Because even if the upper layers of a cryptographic system are very secure, the information leaking in lowest layer may significantly decrease the level of security of the whole system. We find that implementing SCA countermeasures on each layer of ECC system should increase much the level of the security of the cryptographic device.

Trade-off between objectives One may say that presented objectives for a cryptographic system exclude one another. That, if we want to have efficient cryptosystem it cannot be loaded with countermeasures because they would limit much its efficiency. On the other hand very efficient cryptosystem without any countermeasure is useless nowadays. The problem is to find a trade-off between efficiency of the system and its security, i.e. to find a way to secure the system without degrading its elaborated efficiency.

2.4. Thesis formulation and research objectives

According to the formulated problems we may define the research objectives. The two main ones are: to create efficient $GF(2^m)$ arithmetic operators and to make the operators secure against some types of SCAs.

In order to create efficient $GF(2^m)$ basic arithmetic operators we have to familiarise ourselves with details of operations on finite fields. Then it is necessary to perform a vast research and choose most suitable basis to represent finite-field elements. Later we have to carefully study existing algorithms for operations defined over $GF(2^m)$. Then we have to translate the most promising ones to hardware and analyse meticulously their work in order to be able to create our own solutions as efficient as possible.

The second objective is to secure the created arithmetic operators. There are many types of SCAs, we have decided to secure our operators against

family of power analysis attacks. The power analysis attacks exploit the correlation between power consumed by the cryptosystem and operations performed inside. We want to find effective algorithmic and architectural countermeasures against those type of attacks.

Working on those two objectives we have to remember to verify the countermeasures impact on the overall performance of arithmetic operators. If the countermeasure degrades much the work of the solution then we have to rethink the countermeasure and either upgrade it or abandon the idea. We have summarised all objectives of our research in the following plan:

1. Study of elliptic curve cryptography, finite fields, arithmetic operators, side channel attacks and countermeasures.
2. Study of parameters of $GF(2^m)$ arithmetic operators - number representation, algorithms, requirements.
3. Design and development of hardware $GF(2^m)$ arithmetic operators solutions.
4. Theoretical and practical evaluation of designed operators' performances in FPGA circuit - the operators' efficiency is evaluated and if necessary improved until the results are satisfying, and final versions of efficient operators are provided
5. Design and development of test environment for evaluating security of elaborated operators - experiments using simulators, specific FPGA boards, FPGA dedicated internal signals analysers, probes and oscilloscope.
6. Evaluation of security of designed arithmetic operators, elaboration of countermeasures.
7. Insertion and evaluation of countermeasures - evaluations of their efficiency, analysis of their impact on the performance of the operators, if the results are not satisfying return to previous point.
8. Proposition of efficient and secure basic $GF(2^m)$ arithmetic operators and their final evaluation.
9. Analysis, documentation and publication of obtained results.

Having identified the existing problems we may formulate now the main thesis we want to prove with our researches. The thesis is as follows:

It is possible to create efficient and secure against some side-channel power analysis attacks $GF(2^m)$ arithmetic operators dedicated to reconfigurable hardware.

We find that it is possible to create very efficient $GF(2^m)$ arithmetic modules dedicated for elliptic curve cryptosystems, working on operands of sizes up to 600 bits, and that it is possible to secure them against information leakage without significant overhead. Moreover we claim that it is possible to develop such countermeasures against power analysis attacks which would not decrease significantly the performance of our elaborated modules but will significantly increase their security against power analysis attacks.

The following chapters describe the researches conducted to prove our thesis.

3. Arithmetic operators on $GF(2^m)$

There are two main operations defined in $GF(2^m)$: addition and multiplication. All other operations (inversion, division, squaring, etc) can be performed using multiplication and addition. The complexity of some operations depends on the representation of the elements of the field.

The most popular bases (for definition see Chapter 2, Section 2.2) used for representing elements in cryptographic applications are [36]: polynomial (canonical basis), normal basis and its variations (optimal normal bases, gaussian normal bases), dual basis. Choosing right basis is not a simple task. We have performed vast research on bases of finite fields elements taking into account our target devices, results described in known literature obtained for each basis and the application of developed finite fields arithmetic operators. The basic theory about each basis is presented in Chapter 2, Section 2.2, here we will present the most popular applications of each basis, its advantages, disadvantages and reasons why we have chosen the certain basis.

For our solutions we have chosen polynomial (standard, canonical basis). At first we have eliminated from our choices the dual basis. Mainly due to the fact that it was mentioned in few articles, such as [72], that it is not suitable for large m (field size). What is more all described dual basis solutions were designed for very small m (up to 16). We have observed that dual basis is used usually in error correcting codes applications, which does not require use of large finite fields. It is usually used for smaller fields because it requires basis transformation operation, which heavily depends on field generator (irreducible polynomial) form.

The hardest was to eliminate either polynomial or normal basis. Both

standard and normal basis seem to have properties suitable for dedicated application of our basic arithmetic operators. Below we present summarised characteristics of those bases, and some comments found in literature.
Polynomial basis:

- characterised by regularity and simplicity (during implementation yields regular and simple structures of the solution),
- clear and easy to understand from the mathematical point of view,
- the highest clock rate is achieved for polynomial basis solutions,
- multiplication in polynomial basis offers scalability,
- according to [45]: *"The time and space complexities of bit-parallel canonical basis multipliers are much better than that of multipliers based on the normal basis."*

Normal bases:

- squaring operation is very simple, it is just a rotation of vector elements,
- yields irregular structures,
- requires large area,
- it is claimed that for Optimal Normal Bases (ONB) [73] very fast solutions can be achieved,
- using ONBs, requires basis conversion from normal to optimal normal basis, which may be costly.

The features of both bases presented here were collected during study of literature on the subject. Some information contradict the other, thus it is hard to choose the right basis. Finally we have decided to choose polynomial basis mainly due to the fact that it yields regular and simple structures. As to this feature all known to us sources agree. We presume that for hardware implementation it is much better when the structure is regular and simple. Otherwise we may experience synchronisation, routing or hazard problems [85]. The more irregular and complex architecture inserted into FPGA circuit the harder to control it.

Moreover according to [36], a very important position in literature on

ECC : *"Experience has shown that optimal normal bases do not have any significant advantages over polynomial bases for hardware implementation. Moreover, field multiplication in software for normal basis representations is very slow in comparison to multiplication with a polynomial basis;"*. However we find that normal bases group maybe very promising and it would be worth analysing it in the future.
The other important operators' parameters are the field size m (operands size) and field generator $f(x)$. Designed arithmetic operators should serve ECC applications thus we will use the values recommended in ECC standards [32]. The most recent values are presented in Table 3.1.

Table 3.1.: NIST recommended parameters [32]

field size m	irreducible polynomial $f(x)$
163	$f(x) = x^{163} + x^7 + x^6 + x^3 + 1$
233	$f(x) = x^{233} + x^{74} + 1$
283	$f(x) = x^{283} + x^{12} + x^7 + x^5 + 1$
409	$f(x) = x^{409} + x^{87} + 1$
571	$f(x) = x^{571} + x^{10} + x^5 + x^2 + 1$

We target all our solutions to Field Programmable Gate Arrays (FPGA), for simulation and testing purposes we use Xilinx circuits. We will implement our operators in small Spartan-3E XC3S1200E device [116] and one of the biggest FPGAs: Virtex-6 LX240T [20]. For testing robustness against side-channel we also implement our circuits in Virtex-II Pro XC2VP7 FPGA [115] mounted on SASEBO-G board [94]. The elaborated solutions architectures are described in VHDL, and synthesised, placed and routed, and implemented using Xilinx ISE 12.2 (for Virtex-6 and Spartan-3E) and ISE 9.2 (Spartan-3E, Virtex-II Pro) environments and their tools. All the behavioral and post-route simulations are performed using built-in ISim simulator or ModelSim simulator from Menthor Graphics. All the implementation results given in this chapter are values predicted/calculated by Xilinx ISE environment.

3.1. Finite Field Addition

Addition is a very simple operation. In case of polynomial basis, it is a polynomial addition, which can be viewed as a XOR operation of two m-bit binary vectors (m is a field size). Let $a(x)$ and $b(x)$, two polynomials of size m, be the elements of $GF(2^m)$ and let $c(x)$ be its sum and also an element of the field. Addition of two polynomials is carried out under modulo 2 arithmetic, i.e. to obtain c we have to perform the bitwise exclusive OR operation. This operation is regarded as a very simple one due to the fact that we do not need to bother about carry propagation and length of carry chains. On the other hand, if we perform very simple operation but on large numbers, we may experience some problems. If we XOR large numbers the operation, although simple, may take a lot of hardware resources.

Summarising, addition is rather fast operation however for large numbers it may take large amount of area and should be well synchronised. In order to see if there exist real problems with this operation we have designed few very simple operators.

It is very easy to paralellise this operation and it is possible to perform on-the-fly addition while receiving consecutive parts of both operands with no need for storing them as well as the result. In terms of efficiency the design of addition unit does not cause many problems. However in terms of security the addition may be very problematic what will be shown in next chapter.

The ECC processor, in which the designed units will be built-in, assumes sending data over buses in words (16, 32-bit words). According to this assumption we propose some addition operator solutions.

We have studied the following cases:

1: Addition on-the-fly of a, b words, putting partial results in shifted register c (solution 1);

2: Addition on-the-fly of a, b words, putting partial results in memory block (solution 2);

3: Waiting for the whole a, b vectors, adding them at the end (solution 3);

The results we have obtained shown us that this operation is rather simple and should not generate a lot of problems. We do not present here values obtained for the solution 3. This solution is really very simple, we may say that it is a translation of input signal to the output, thus it is difficult to synthetise it to obtain some credible values.

Looking at the results presented in Table 3.2 it is easy to observe that addition in $GF(2^m)$ is really simple. Our solutions are very small, around 20-30 LUTs and fast. With growth of the size of input operands the area grows slightly, the same applies to the decrease of maximum speed of the solution. However for solution 1 and the biggest field the frequency in comparison for the smaller solutions drastically drops. In practice, in complete ECC system, addition is usually performed parallely to other operations or interleaved with them due to its simplicity.

Table 3.2.: Addition solutions (Virtex-6)

field size	solution 1		solution 2	
m	[LUT]	[MHz]	[LUT]	[MHz]
163	21	771	26	562
233	21	771	26	562
283	22	767	28	560
409	22	767	28	560
571	24	578	31	558

3.2. Finite Field Multiplication

The second basic operation defined in finite field is multiplication. Multiplication in contrary to addition is regarded as a complex operation.

Designing an efficient hardware algorithm for multiplication in binary finite field extensions is one of the aims of this thesis, thus the thorough analysis of existing methods and solutions is necessary. Our solutions will not be totally novel due to the fact that we base on old and known mathematical theories. Our goal is to modify or merge the existing algorithms in order to fulfill demanding cryptographic requirements. We also have to find the best way of translating the algorithms to hardware and not losing much of their features. Our operators are aimed for ECC applications thus they need to perform multiplication on large numbers (150–600 bits) and simultaneously be very fast and occupy reasonable amount of area of the target device. Moreover their structures should be easily modifiable (flexible) to add countermeasures against physical attacks.

Finite field multiplication can be regarded as modular operation because it consists of two steps: multiplication and reduction. In order to obtain the finite field multiplication result we have to multiply operands and then reduce the product with use of field generator, so the resulting element will be the element of the same field. Let $a, b \in GF(2^m)$, be the $(m-1)$ degree polynomials where $a = a(x) = \Big(\sum_{i=0}^{m-1} a_i x^i \Big) = a_0 + a_1 x + a_2 x + \ldots + a_{m-2} x^{m-2} + a_{m-1} x^{m-1}$ and $b = b(x) = \Big(\sum_{j=0}^{m-1} b_j x^j \Big) = b_0 + b_1 x + b_2 x + \ldots + b_{m-2} x^{m-2} + b_{m-1} x^{m-1}$, and let $f = f(x) = 1 + f_1 x + f_2 x + \ldots + f_{m-2} x^{m-2} + f_{m-1} x^{m-1} + f_m x^m$ be m degree irreducible polynomial generating the field (for examples of irreducible polynomials used in cryptography see 3.1).
We want to perform:

$$c(x) = a(x)b(x) \bmod f(x) \tag{3.1}$$

First we have to perform multiplication:

$$\begin{aligned} d(x) &= a(x)b(x) \\ &= a(x)\Big(\sum_{i=0}^{m-1} b_i x^i\Big) = \Big(\sum_{i=0}^{m-1} b_i a(x) x^i\Big) \\ &= \Big(b_0 a(x) + b_1 a(x)x + b_2 a(x)x^2 + ... + b_{(m-1)} a(x) x^{(m-1)}\Big) \end{aligned} \tag{3.2}$$

and then reduction [25] (the degree $2m-2$ polynomial $d(x)$ is reduced iteratively by irreducible polynomial $f(x)$ of degree m):

$$\begin{aligned} d^{(2m-2)} &= d(x) \\ d^{(k-1)} &= d^{(k)}(x) + f(x) d_k^{(k)} x^{k-m}, \quad m \le k \le 2m-2, \end{aligned} \tag{3.3}$$

where $d^{(k)}$ is a partial remainder and $d^{(2m-2)}(x) = c(x)$.

Generally $GF(2^m)$ multiplication algorithms can be divided in two groups: **two-step algorithms**, in which we perform separately multiplication and reduction (in two consecutive steps), and **interleaved algorithms**, in which we interleave/combine multiplication with reduction. The most popular algorithms of both groups are presented and discussed below.

We have analysed many algorithms and their different versions to be able to take and combine those features of the algorithms, which will allow us to create the most efficient algorithm fulfilling, assumed requirements. Here we present only the most interesting results of our analysis.

3.2.1. Two-step algorithms

One of the most popular group of algorithms comprises two-step algorithms. Two-step because to perform finite field multiplication we need to perform two separate steps: multiplication $d = ab$ and reduction $c = d \bmod f$. There exist many versions of two-step multipliers. They combine different meth-

ods for multiplication and reduction in order to achieve the best, the most efficient solutions.

In the following paragraphs, different methods for performing multiplication and reduction are presented. Their features are thoroughly analysed in terms of hardware design and efficiency. Finally the chosen combinations of multiplication and reduction are described and compared in order to expose their advantages and disadvantages.

Multiplication step Here we will present the analysis of some widely known polynomial multiplication methods.

Schoolbook multiplication method (shift-and-add method). The simplest and the most known polynomial multiplication method is so called schoolbook method (shift-and-add method). Having two polynomials $a(x)$ and $b(x)$ of degree $m-1$ we obtain product $d(x)$ of degree $2m-2$ in the way presented in Equation 3.2. We multiply polynomial a by each coefficient of polynomial b, i.e. we successively shift the polynomial a by each coefficient of polynomial b, and add (`XOR` in our case) the partial results. One may regard the a and b polynomials as two binary vectors. Vector a, which is being shifted left (multiplied by 2^n) by 2, 4, 8, etc., and vector b, which bits decide if we should accumulate the particular shift of vector a or not. The illustrative example for $m = 4$ is shown below:

$$
\begin{array}{lrl}
 & a_3a_2a_1a_0 & \\
\times & b_3b_2b_1b_0 & \\
\hline
\oplus & a_3a_2a_1a_0 & \wedge\ b_0 \\
\oplus & a_3a_2a_1a_0 & \wedge\ b_1 \\
\oplus & a_3a_2a_1a_0 & \wedge\ b_2 \\
\oplus & a_3a_2a_1a_0 & \wedge\ b_3 \\
\hline
= & d_6d_5d_4d_3d_2d_1d_0 &
\end{array}
$$

The method is very simple and allows creating regular combinatorial de-

signs. However if we want to use it to multiply large numbers it can be inefficient, especially regarding chip space utilisation.

All basic polynomial multiplication algorithms are usually some kind of variations of this method. The difference between variations usually lies in the way the operands are represented. There exists a simple vector version, matrix-vector version and divide-and-conquer version (operands are partitioned). Different representations of operands strongly influence the hardware structures elaborated to perform the multiplication. Some cause acceleration of the solution but on the other hand they increase amount of resources used, some decrease the area cost but also degrade speed of the solution. If we want to find the efficient solution, we always have to look for some trade-off between resources occupied and time needed to execute the algorithm. Different representations of operands let us also perceive new ideas of optimisation of the basic algorithm.

In order to efficiently design hardware unit for a polynomial multiplier it is important to notice the advantages of different mathematical algorithms and to find the tradeoffs between area and speed of the elaborated solution. Unfortunately even the best theoretical optimisations and simplifications of the algorithms do not give the expected efficiency increase when translated to hardware. We have analysed many of the theoretical improvements by translating them to hardware in order to see and maybe use their advantageous properties. However we were not very satisfied with obtained results. In practice many of those improvements either yielded hardware structures similar to original proposition or decreased efficiency of hardware solution. In many cases it was not clear to what field sizes the algorithm improvements target. As we would present, there exist improvements suitable only for certain field sizes, while for the other they does not work as expected. However basing on the theoretical approaches presented in the literature it is possible to extract the best parts, promising for hardware solution design, of all the proposals and merge them to create an efficient hardware algorithm.

Matrix-vector approach. One of the most popular variations of the shift-and-add method is its matrix-vector version [25]. There, polynomial $a(x)$ is represented by a specific matrix A of size $(2m-1)\times m$, in which each column represents consecutive shifts left of $a(x)$. Element $b(x)$ is represented by m size vector and product $d(x)$ is also a vector but of size $(2m-1)$. Instead of $d(x) = a(x)b(x)$ we perform:

$$d = Ab = \begin{bmatrix} d_0 \\ d_1 \\ \vdots \\ d_{m-1} \\ d_m \\ d_{m+1} \\ \vdots \\ d_{2m-3} \\ d_{2m-2} \end{bmatrix} = \begin{bmatrix} a_0 & 0 & 0 & \dots & 0 & 0 \\ a_1 & a_0 & 0 & \dots & 0 & 0 \\ \vdots & \vdots & \ddots & \dots & \ddots & \vdots \\ a_{m-2} & a_{m-3} & a_{m-4} & \dots & a_0 & 0 \\ a_{m-1} & a_{m-2} & a_{m-3} & \dots & a_1 & a_0 \\ 0 & a_{m-1} & a_{m-2} & \ddots & a_2 & a_1 \\ 0 & 0 & a_{m-1} & \dots & a_3 & a_2 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \dots & a_{m-1} & a_{m-2} \\ 0 & 0 & 0 & \dots & 0 & a_{m-1} \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ \vdots \\ b_{m-3} \\ b_{m-2} \\ b_{m-1} \end{bmatrix} \tag{3.4}$$

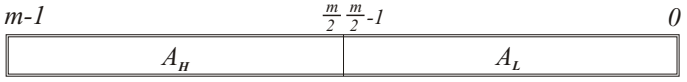
As mentioned above and what can be observed comparing Equation 3.2 and Equation 3.4 each column of matrix A represents shifted vector a, index of column indicates how many times vector a was left shifted. Further, as in case of shift-and-add method, bits of vector b denote which column of matrix A (which shift of vector a) should be accumulated to obtain the product vector d. We have implemented this algorithm in hardware in order to compare it with the schoolbook version and concluded that they give basically the same implementation results. However the matrix representation of $a(x)$ reveals to us more ideas of optimisation of hardware realisation. Combining elements of standard algorithm with the operands representation used in matrix-vector approach allowed us creating an efficient algorithm

for polynomial multiplication in $GF(2^m)$. The implementation results and more details concerning implementation are presented below.

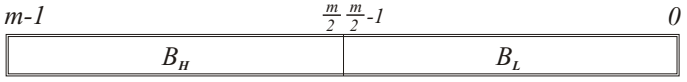
Divide-and-conquer algorithm and Karatsuba Ofman trick. One of the most popular improvements of the classical multiplication method is Karatsuba-Ofman trick [44]. The trick improves divide-and-conquer version of multiplication algorithm. Theoretically it decreases number of the most complex operations we have to perform to get product of a and b.

The aim of divide-and-conquer algorithms is to reduce large (hard) problem into a set of smaller (easier) problems. In our case we partition input polynomials (multiplicands) so instead of performing one multiplication of very large polynomials we perform many multiplications of much smaller polynomials and finally combine the partial results. It is even possible to partition our inputs into single coefficients, however this may not be very efficient.

The method assumes the following partitioning of the input parameters. For polynomials of size $m = 2t$ (m is even) we have:



$$a(x) = x^{\frac{m}{2}} A_H + A_L = x^{\frac{m}{2}}(x^{\frac{m}{2}-1} a_{m-1} + \cdots + a_{\frac{m}{2}}) + (x^{\frac{m}{2}-1} a_{\frac{m}{2}-1} + \cdots + a_0)$$



$$b(x) = x^{\frac{m}{2}} B_H + B_L = x^{\frac{m}{2}}(x^{\frac{m}{2}-1} b_{m-1} + \cdots + b_{\frac{m}{2}}) + (x^{\frac{m}{2}-1} b_{\frac{m}{2}-1} + \cdots + b_0)$$

Hence one can denote multiplication as follows:

$$\begin{aligned} d(x) &= a(x)b(x) \\ &= (x^{\frac{m}{2}} A_H + A_L)(x^{\frac{m}{2}} B_H + B_L) \\ &= x^m A_H B_H + x^{\frac{m}{2}}(A_H B_L + A_L B_H) + A_L B_L. \end{aligned} \tag{3.5}$$

According to Equation 3.5 in order to multiply a and b one has to perform: four sub-multiplications and three additions.

Karatsuba and Ofman [44] had modified the Equation 3.5 in such a way that the number of needed sub-multiplications decreased. However the number of necessary additions, `XOR` operations, increased. The modified equation employing Karatsuba-Ofman trick is as follows:

$$\begin{aligned} d(x) &= a(x)b(x) \\ &= (x^{\frac{m}{2}}A_H + A_L)(x^{\frac{m}{2}}B_H + B_L) \\ &= x^m A_H B_H + x^{\frac{m}{2}}((A_H + A_L)(B_H + B_L) - A_H B_H - A_L B_L) + A_L B_L. \end{aligned} \tag{3.6}$$

The number of necessary multiplications is now three and number of `XOR` operations needed is six. It is assumed that multiplication operation is more costly than addition. If we operate on single coefficients and we assume that multiplication equals to `AND` operation and addition is equivalent to `XOR` operation than this assumption does not hold, especially in case of FPGA circuit.

In theory when we build FPGA circuits on LUTs there is no difference if we use `XOR` or `AND` gate because each of those gates can take one LUT if they yield the values of different outputs. However if we assume that LUT size is finite (the function size, which can be implemented is limited) then the `XOR` gates take more space. We have conducted some tests on this matter and found that this is caused by the fact that however one describes the `XOR` gate in HDL (hardware description language) it is always substituted by the combination of `AND` and `OR` gates. This can be observed while inspecting technology schematics generated by the Xilinx ISE environment. According to this the `XOR` function can be regarded as three gates plus inverters. Thus it is obvious that it is more costly than a single `AND` gate. All that proves the opposite to what is assumed; that is that `XOR` function is suggested to be the trivial operation, on the contrary it appears to be a source of delay

and space problems in the design.

Taking into consideration our observations we have decided to test the performances of basic divide-and-conquer algorithm (Equation 3.5) alongside with testing the algorithm involving Karatsuba-Ofman trick (Equation 3.6).

The first obtained results confirmed our observations concerning addition and multiplication complexity. The straightforward implementation of Karatsuba-Ofman optimisation for short binary vectors did not yield solutions more efficient than the algorithm with four multiplications, on the contrary, first attempts for small input vectors (4-bit) had shown that the optimisation is not efficient at all and yields bigger and slower designs. The advantage of Karatsuba-Ofman trick could be observed starting from 16-bit input vectors as will be discussed later. The implementation analysis presented further will describe the practical effectiveness of Karatsuba-Ofman optimsation.

The main problem, which occurs during the implementation of divide-and-conquer algorithms, is how to wisely partition the polynomials (operands) to make the designed multiplier really efficient. In order to find the most efficient partitioning, we had performed vast analysis of various possibilities.

We have started our tests and analysis from implementation of small number multipliers. We presumed that starting from implementations of large number multipliers might cause that some algorithms' properties will be invisible due to being masked by other problems existing in extensive designs.

Multiplication step - hardware realisation. The presented multiplication algorithms can be translated to hardware in two manners: straightforward as a combinatorial circuits or can be partitioned to perform the multiplication steps sequentially. Combinatorial circuits are usually very fast but also very extensive whereas sequential ones are slower, but better synchronised and more compact. The best solution will be to combine combinatorial and

sequential features. In hardware it is important to find trade-offs between those two types of realisations in order not to end up with extensive and very slow solutions.

To ease the comparison of elaborated solutions a new measure, the AT efficiency factor is introduced.

Definition 3.2.1. *The efficiency factor AT is the measure of efficiency in terms of execution time of the operation and area taken by the circuit.*

$$AT = (area \times execution\ time)$$

It is a normalised product of area taken by the solution and the time needed to perform the operation.

To calculate AT for combinatorial designs we have multiplied maximum delay value given by Xilinx ISE environment by predicted area. In case of sequential designs we have multiplied number of clocks needed to obtain result by minimal period given by ISE environment and by predicted area. All the AT values were normalised to ease comparison.

During the primary analysis we have assumed that big number multipliers will be designed as a combination of smaller number multipliers. We have presumed that such approach will yield best results.

We begin our study of each algorithm for a very small input numbers (4-bit) and then observe what are the best combinations and best sizes of primary multipliers when it comes to designing bigger multipliers. Such approach seems advisable if one wants to create large number multipliers of various sizes it should make the multiplier architectures scalable and flexible.

The first analysed multiplication method was the shift-and-add method. We have started from the straightforward implementation and created pure combinational structures multiplying two vectors. The simplest version of the circuit was a translation of Equation 3.2. To each bit of resulting polynomial d a combination of `XOR`ed values were assigned as presented for $m = 4$

in Figure 3.1.

We have started the analysis with implementation of 4-bit multiplier. The

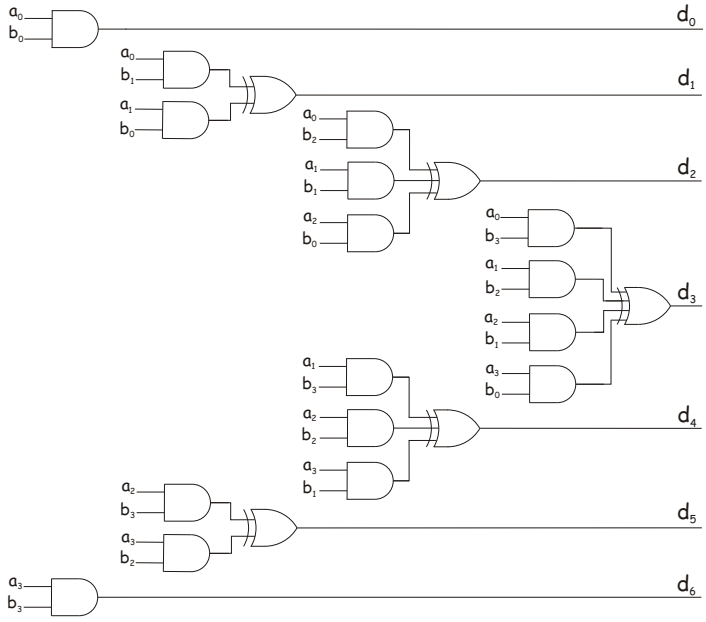


Figure 3.1.: Idea of circuit performing shift-and-add method for $m = 4$

synthesis of the 4-bit multiplier circuit yielded satisfying and promising results. Created design has quite regular structure and works rather fast. It is compact (11 LUTs) and a total combinatorial path delay is fairly low - around 7.7 ns (for Spartan-3E). With the doubling of the size of the input polynomial the number of LUTs taken increases approximately 4 times. For 4-bit inputs the multiplier circuit takes 11 LUTs, for 8-bit it is already over 40, then for 16 its nearly 200, thus for 32-bit it is approximately 800 LUTs. According to this 256-bit multiplier will take over 52000 LUTs. Spartan-3E has 17344 LUTs, which means that purely combinatorial design of 256-bit multiplier far exceeds number of available resources. One may say that there are bigger devices available. That is why as the second target FPGA we have chosen Virtex-6, which has 150720 LUTs, so such multiplier will fit

in but will occupy 1/3 of the chip. For bigger multiplier, 512-bit, the design grows four times so probably it will have approx. 200000 LUTs, which exceeds also the area of Virtex-6. Of course there exist still bigger chips, in Virtex-6 family of FPGAs the biggest one has 566784 6-input LUTs, so that one will fit 512-bit combinational multiplier. However building a design with the assumption that there is always newer and bigger FPGA, which will fit it, is nonsense unless we always have huge amount of spare money to spend. Nowadays in a world where one of the most important features of the device is the cost of its production, the excessive, usually containing much redundancy, designs are useless. Additionally the routing costs and delay problems are smaller if we connect the units inside the chip instead of connecting the chips with some external buses. As a result of our study one may conclude that it is not wise to create purely combinatorial circuit for a large number multiplier. Although they are fast (the combinatorial path delay increases slightly with the increase of the size of the input parameter, around 0.5 ns or even less), for large inputs, they take a huge amount of hardware resources, which makes them not suitable for integration with other units. The overall hardware cost will be too big. Extensive designs cause also routing and synchronisation problems.

Thus we may conclude that big straightforward combinatorial multipliers are very area inefficient. However such regular structures seem to be very appropriate to be implemented in FPGAs. Our next idea was to use small combinational multipliers to build bigger solutions.

The biggest implemented by us fully combinatorial multiplier was 32-bit multiplier. All obtained results are presented in Table 3.3.

Because purely combinatorial circuit yields huge solutions for large sizes of input operands we have decided to test what are the results and advantages of sequential solutions. The question arising in this case was how to "sequence" (partition) the design to obtain a compact and fast multiplier.

The first idea was to observe the purely sequential solution, which means to perform each stage of Equation 3.2, each `XOR` operation, in a separate

clock cycle. The primary size of input operands was 4 bits as in previous cases. The resulting hardware structure in Spartan-3E is very regular and can work with high frequencies, around 300 MHz. However such a solution requires, for 4-bit inputs, more LUTs (35 LUTs) than the combinatorial solution. Such a result is not surprising and could be predicted. What is surprising is the fact that for larger input vectors its area increases almost similarly as in case of combinatorial circuits. Hence it requires more space than the combinatorial circuit, simultaneously requiring more time to perform the task. Probably it caused by the synthesizer settings which duplicate gates structures, instead of reusing just one structure in each clock cycle (state of finite state machine).

Our next idea was to create the sequential circuit utilising the classical combinatorial multiplier units. We have analysed the behaviour of few 32-bit multipliers built as a sequential combination of 16, 8 and 4-bit multipliers. To combine obtained partial results we have used divide-and-conquer algorithm. 32-bit multiplier is still small in comparison to ones we ought to create (150–600 bits), however it may show us what is the best way of partitioning input operands in order to achieve the best efficiency. In the following equations A_i, B_i denote i-bit A, B elements (vectors). Our first proposition was to create a circuit using sixteen times one 8-bit sub-multiplier according to the equation 3.7.

$$\begin{aligned} A_{32}B_{32} &= A_{16}^{H}B_{16}^{H} + A_{16}^{H}B_{16}^{L} + A_{16}^{L}B_{16}^{H} + A_{16}^{L}B_{16}^{L} \\ &= A_{8}^{Hh}B_{8}^{Hh} + A_{8}^{Hh}B_{8}^{Hl} + A_{8}^{Hl}B_{8}^{Hh} + A_{8}^{Hl}B_{8}^{Hl} \\ &+ A_{8}^{Hh}B_{8}^{Lh} + A_{8}^{Hh}B_{8}^{Ll} + A_{8}^{Hl}B_{8}^{Lh} + A_{8}^{Hl}B_{8}^{Ll} \\ &+ A_{8}^{Lh}B_{8}^{Hh} + A_{8}^{Lh}B_{8}^{Hl} + A_{8}^{Ll}B_{8}^{Hh} + A_{8}^{Ll}B_{8}^{Hl} \\ &+ A_{8}^{Lh}B_{8}^{Lh} + A_{8}^{Lh}B_{8}^{Ll} + A_{8}^{Ll}B_{8}^{Lh} + A_{8}^{Ll}B_{8}^{Ll}, \end{aligned} \tag{3.7}$$

Each expression of the form ${A^{Ii}}_8{B^{Ii}}_8$ symbolises one operation performed by the 8-bit sub-multiplier. Second proposed solution uses four times one

16-bit sub-multiplier.

$$A_{32}B_{32} = A_{16}^H B_{16}^H + A_{16}^H B_{16}^L + A_{16}^L B_{16}^H + A_{16}^L B_{16}^L. \tag{3.8}$$

Here each expression of the form $A^I{}_16B^I{}_16$ symbolises one operation performed by the 16-bit sub-multiplier. The last combination uses four 8-bit sub-multipliers four times. In the equation 3.7 each row of the form $A_8^{Ii}B_8^{Ii} + A_8^{Ii}B_8^{Ii} + A_8^{Ii}B_8^{Ii} + A_8^{Ii}B_8^{Ii}$ symbolises operations performed by one 8-bit sub-multiplier.

Those tests were made to see what is the best combination of sub-multipliers. The best result was obtained for combination 3.8: the solution takes the smallest number of LUTs and can work at high frequencies; what is more it needs small number of clock cycles to perform a, b multiplication. We have compared that solution with the combinational one using four 16-bit units. In terms of area taken the second solution seems to be much worse. Comparing AT efficiency factors (see Definition 3.2.1), for sequential so-

Table 3.3.: Schoolbook multiplication: implementation results (Spartan-3E)

Multiplier	Combinatorial delay [ns] / Max. freq. [MHz] (# of clock cycles)	area [LUT]	AT
4-bit	7.68 ns	11	84
8-bit	8.9 ns	43	383
16-bit	9.4 ns	195	1833
32-bit, combinational, four 16-bit multipliers	12 ns	818	9816
32-bit, sequential, one 16-bit multiplier: eq.3.8	451 MHz (6 clks)	277	3685
32-bit, sequential, one 8-bit multiplier: eq.3.7	451 MHz (14 clks)	318	9871
32-bit, sequential, four 8-bit multipliers: eq.3.7	451 MHz (6 clks)	279	3712

lution we have $AT_{seq} = 2.213\text{ns} \times 6 \times 277 \approx 3678$ and for combinational

$AT_{comb} = 12.076\text{ns} \times 789 \approx 9527$, we can see that sequential solution is overall three times better than the combinational one. The solution using four 8-bit sub-multipliers yields similar results to the one using one 16-bit unit, however we have to deal with more sub-blocks. The worse solution happens to be the one using only one 8-bit unit. Mainly due to the fact that it needs the greatest number of clock cycles to compute the result. We have analysed many more variations of 32-bit sequential multiplier reusing combinatorial multipliers, but those presented seemed to be the most interesting from our point of view, for further analysis of bigger multipliers. The results of implementations for Spartan-3E FPGA are presented in Table 3.3.

Before starting the analysis of matrix-vector approach we have attempted to build bigger multipliers employing just the schoolbook approach and divide-and-conquer methods. We have built 64-bit multipliers and 128-bit multipliers but we were not very satisfied with results so we have switched to analysis of matrix-vector approach.

On the other hand the analysis of 64-bit multipliers allowed us to see advantages of Karatsuba-Ofman optimisation. In Table 3.4, we present results obtained for two 64-bit multipliers, built from the same elements (built of 32-bit sub-multipliers, built of 16-bit sub-multipliers, which are built of 8-bit combinational sub-multipliers). One multiplier combines all partial results using standard divide-and-conquer approach (see Figure 3.2.1) and the other using Karatsuba-Ofman trick (see Figure 3.2.1).

According to the results presented in Table 3.4 it is clear that the design using Karatsuba-Ofman trick is much faster and smaller than the one using standard version of divide-and-conquer method.

The last multiplier built with use of pure combinational sub-multipliers was 128-bit multiplier. The most efficient classic 128-bit multiplier designed, however we did not explored all possibilities, was built of eight 16-bit sub-multipliers. The idea for this multiplier was to divide input polynomials into 16-bit words. Thus for 128-bit inputs we had to use eight times eight

Table 3.4.: Classic divide-and-conquer technique and Karatsuba-Ofman trick comparison

AT		64-bit multiplier	
		Divide-and-conquer	Karatsuba-Ofman
Area [LUT]	Spartan-3E	2897	1754
	Virtex-6	1782	1424
Max.delay [ns]	Spartan-3E	14.25	13.37
	Virtex-6	3.79	5.13
AT	Spartan-3E	41270	23442
	Virtex-6	6746	7301

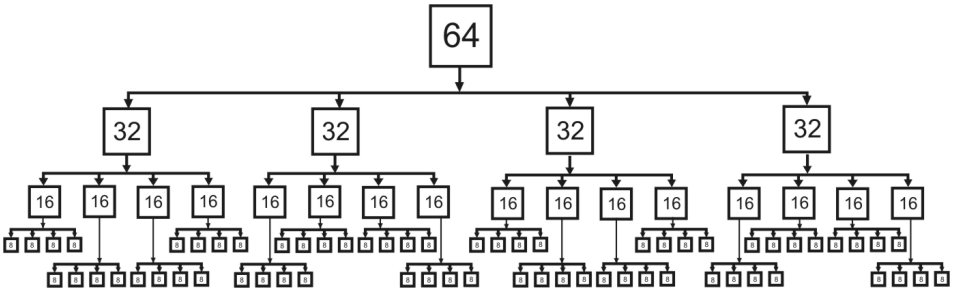


Figure 3.2.: Classic divide-and-conquer approach

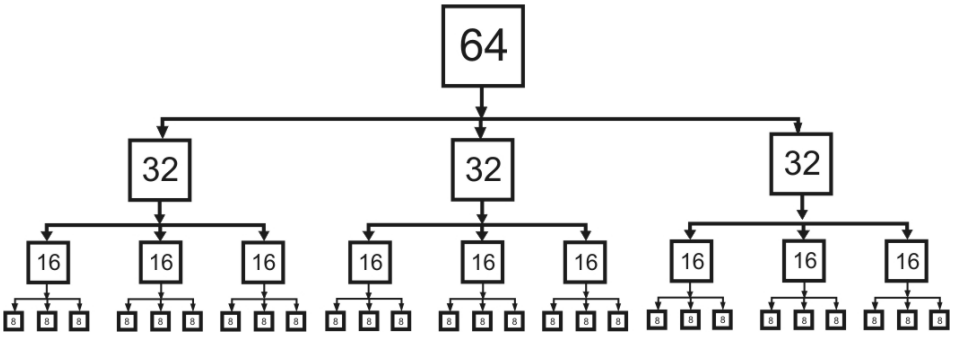


Figure 3.3.: Karatsuba-Ofman approach

16-bit sub-multipliers. That is in each clock cycle we multiply each 16-bit word of vector a by successive 16-bit words of vector b. Finally we combine all partial results. The constructed multiplier has the parameters presented in Table 3.5.

Table 3.5.: 128-bit multiplier

Area (LUTs)		Max.frequency (MHz)		AT	
Spartan-3E	Virtex-6	Spartan-3E	Virtex-6	Spartan-3E	Virtex-6
2919	1561	288	775	101354	20142

The most advantageous feature of the circuit is that it needs only 10 clock cycles to perform multiplication, thus it seems rather efficient in terms of speed.

Further analysis aimed at investigating how the area of combinatorial Karatsuba-Ofman designs depend on the size of input polynomials. By analysis of 32, 64 and 128-bit multipliers we saw that the design area grows almost 4 times for doubling of the size of the input arguments. However in contrary to pure combinatorial schoolbook method solutions areas, which were each time, multiplied by more than 4 (for large numbers the multiplication factor grows to 5), here areas of multipliers grow by less than four. For example pure combinatorial 64-bit multiplier has taken over 2000 LUTs but Karatsuba-Ofman multiplier took about 1750 LUTs. Nevertheless in case of this approach the combinatorial path delay increases faster and for 64-bit multipliers it is over 16 ns while in case of previous solution it is about 12 ns. The results for different sizes of divide-and-conquer multipliers are compared in Table 3.6. Observing parameters of obtained designs we can clearly see that Karatsuba-Ofman trick is efficient for large operands. Up to 16-bit input vectors it is visibly less efficient than classic divide-and-conquer method.

Next we have analysed what advantages yield matrix-vector approach.

Initially we have implemented the method straightforward for small multipliers, but for example for 4 bits we have obtained exactly the same struc-

Table 3.6.: Divide-and-conquer methods implementations

Multiplier (basic unit)	Type	Delay [ns]	Area [LUT]	*AT*
8-bit				
4-bit combinatorial multiplier	*dq*	*9*	*45*	*405*
	KO	9.3	46	427
16-bit				
4-bit combinatorial multiplier	*dq*	*12.6*	*158*	1991
8-bit combinatorial multiplier	*dq*	*10.3*	*172*	*1772*
	KO	11.7	159	1860
8-bit Karatsuba-Ofman multiplier	KO	12.7	170	2159
32-bit				
(16-bit divide-and-conquer multiplier built of 8-bit combinational multipliers)	*dq*	*11.9*	*518*	*6167*
(16-bit Karatsuba-Ofman multiplier built of 8-bit combinational multipliers)	KO	13.3	429	5708
64-bit				
(32-bit Karatsuba-Ofman multiplier) built of 16-bit Karatsuba-Ofman multiplier) built of 8-bit combinational multiplier)	KO	16	1753	28048

*(dq: classic divide-and-conquer, KO: Karatsuba-Ofman optimisation)

ture as in case of shift-and-add method. Thus we have reanalysed the Equation 3.4 used in matrix-vector approach and came out with new idea for structure of a multiplier. It seemed obvious that storing whole matrix A during the computations and `XOR`ing its rows in the end would yield enormous solutions, especially in the case of large operands. For example in case of 4-bit inputs, matrix A would be of size 6×4 which could be regarded as six 4-bit vectors. So instead of having one 6-bit output vector and two 4-bit inputs we have one 6-bit input and seven 4-bit inputs. Thus the redundancy here is big. For example if $m = 100$ we would need $2m - 2 = 198$ additional vectors. Our idea is to store only two columns of the matrix A at a time, which in fact means working on two registers representing matrix A

columns. In the first we exchange values of consecutive columns of A and in the second we accumulate partial results. We may actually regard one register as a column of matrix A and the other as our product d. The solution is sequential and due to the fact that we have to process each column of matrix A and we have only one register for storing its values (we may use more registers to process more columns at a time) it requires for m-bit input $2m$ clock cycles to perform multiplication. The analysed implementation can multiply at most 600-bit operands. Exactly the same structure can be used for multiplying for example 2, 4 or 8-bit operands. Depending on how many bits of our "column" vectors we use Xilinx synthetiser optimises the area taken by the solution. Table 3.7 presents parameters of matrix-vector designs for few values of m.

Table 3.7.: Matrix-vector approach implementation results (Virtex-6)

m	Area [LUT]	Maximal frequency [MHz]	AT
32	230	520	28.3
128	838		412.2
163	1050		658.5
233	1490		1335.3
256	1632		1606.9
283	1820		1981
409	2617		4116.7
571	3644		8002.8
600	3707		8554.6

The solution can be optimised, for example number of clock cycles needed to obtain the result could be minimised by increasing the area of the solution, i.e. increasing the number of matrix A's columns stored. We can also try to reuse small multipliers and combine partial results by means of divide-and-conquer techniques. There exist many possibilities. So far the algorithm was the easiest to implement efficiently. The resulting circuit was sequential. The structure as in previous cases was regular and fairly simple.

Next we have analysed if it is possible to increase the efficiency of matrix-vector solutions by utilisation of divide-and-conquer techniques, increasing number of "columns" used in computation process or storing whole matrices. Obtained results with comments are presented in Table 3.8.

It can be observed that inserted optimisations did not really increased the efficiency of the designs. Some of them caused significant decrease of number of clock cycles needed to compute the product, from $2m$ to $m/2$, however in the same moment they have increased area. Interesting seem the designs utilising three sub-multipliers and employing Karatsuba-Ofman trick. Those ones need small number of clock cycles and do not need much more area than in case of simple solution. In fact their efficiency factor AT is better then the one obtained for simple matrix-vector solution.

To start the process of design of required NIST size, i.e. large size multipliers we have compared our most efficient hardware solutions for classic and matrix-vector approach in order to decide which design approaches are most worth utilising. We have considered designs of 32-bit, 64-bit and 128-bit multipliers, see Table 3.9.

For us the most promising solutions are matrix-vector solutions utilising Karatsuba-Ofman optimisation, working on halved input operands. They occupy a reasonable amount of area and they are quite fast. However according to Table 3.9 their AT coefficients are not always the best ones, they are usually average. Examining closer the results we may also see that those solutions which occupy more area can perform multiplication faster, and otherwise. Furthermore we find that the solutions with average AT factors usually contain some trade-offs between space and speed. In fact the designer always have to decide what parameter is the most important and then decide which type of solution to choose.

Initially we have assumed that the best hardware solutions for large num-

Table 3.8.: Modified matrix-vector multipliers: implementation results (Spartan-3E)

m	Area [LUT]	Max. freq./delay [MHz]/[ns]	Description	AT
15	132	205 MHz/30 clks	Simple solution, 2 columns used	19.3
15	324	218 MHz/8 clks	Optimised solution, 8 columns used	11.9
15	171	9.4 ns	Combinational solution, whole matrices "stored"	1.6
64	511	228 MHz/128 clks	Simple solution, 2 columns used	286.88
64	984	223 MHz/64 clks	Uses three 32-bit matrix-vector multipliers	282.4
64	1626	233 MHz/32 clks	Uses four 32-bit matrix-vector multipliers	223.3
64	1299	225 MHz/32 clks	Optimised solution, 8 columns used	184.7
256	1955	231 MHz/512 clks	Simple solution, 2 columns used	4333.2
256	5070	217 MHz/128 clks	Optimised solution, 8 columns used	2990.6
256	2118	220 MHz/228 clks	Uses three 128-bit matrix-vector multipliers	2195
512	3881	234 MHz/1024 clks	Simple solution, 2 columns used	16983.5
512	10983	198 MHz/256 clks	Optimised solution, 8 columns used	14200.2
512	10445	220 MHz/256 clks	Uses three matrix-vector 256-bit multipliers built of 3 128-bit matrix-vector units	12154.2
512	6596	220 MHz/512 clks	Uses three 256-bit matrix-vector multipliers	15350.7

ber multipliers are the ones working on vectors of regular sizes. By regular

Table 3.9.: Comparison of classic and matrix-vector approach implementations results (Virtex-6)

Multiplier	Area [LUT]	Max. f(#clks)/delay [MHz]/[ns]	AT
32-bit			
Three 16-bit multipliers built of three 8-bit combinational multipliers	540	13.3 ns	7.2
matrix-vector multiplier	230	520 MHz (64 clks)	28.3
64-bit			
Built of 32-bit Karatsuba-Ofman units built of 16-bit Karatsuba-Ofman built of 8-bit combinational multipliers	1424	5.13 ns	7.3
matrix-vector multiplier	433	520 MHz (128 clks)	106.6
Karatsuba-Ofman multiplier built of 32-bit matrix-vector units	810	535 MHz (64 clks)	96.9
Divide-and-conquer multiplier built of 32-bit mv multipliers	1476	533 MHz (32 clks)	88.6
Matrix-vector multiplier (matrix divided into 4 parts)	1112	520 MHz (32 clks)	68.4
128-bit			
Multiplier built of eight 16-bit combinational multipliers	1561	775 MHz (10 clks)	20.1
Matrix-vector multiplier	838	520 MHz (256 clks)	412.6
256-bit			
Matrix-vector multiplier	1632	520 MHz (512 clks)	1606.9
Matrix-vector multiplier (matrix divided into 4 parts)	4580	520 MHz (128 clks)	1127.4
Karatsuba-Ofman multiplier, (128-bit matrix-vector units)	2009	535 MHz (228 clks)	856.2
512-bit			
Matrix-vector multiplier	3268	463 MHz (1024 clks)	7227.7
Matrix-vector multiplier (matrix divided into 4 parts)	8769	520 MHz (256 clks)	4317
Karatsuba-Ofman multiplier, built of 256-bit mv multipliers) built of 3 128-bit mv multipliers	10357	528 MHz (256 clks)	5021.6
Karatsuba-Ofman multiplier (256-bit multipliers)	6026	535 MHz (512 clks)	5766.9

we mean 32, 64, 128, 256, 512, i.e. power of 2 sizes. Thus we have created such multipliers to have them as reference designs, i.e. to compare them with multipliers of NIST (irregular, prime) sizes and to see if it is really better (more efficient) to have redundant but regular multiplier or the one optimised for certain prime size. We have assume that it is possible to perform multiplication of:

- $m = 163, 233$-bit vectors using 256-bit multiplier,
- $m = 283, 409$-bit vectors using 512-bit multiplier,
- $m = 571$-bit vectors using 1024-bit multiplier.

It can be observed that for $m = 283$ and 571 the redundancy is huge, thus in those cases we are rather sure that optimised to recommended sizes solutions will give better results.

Table 3.10.: Implementations of classic 163-bit multiplier (Virtex-6)

Multiplier	Area [LUT]	Max. freq.(#clks) [MHz]/[ns]	AT
163-bit matrix-vector multiplier	1050	520 MHz (326 clks)	658.5
256-bit matrix-vector multiplier	1625	520 MHz (512 clks)	1600
163-bit multiplier built of three 82-bit matrix-vector units	1977	535 MHz (164 clks)	606
256-bit multiplier built of three 128-bit matrix-vector units	2009	535 MHz (228 clks)	856.2
163-bit multiplier built of two 128-bit matrix-vector multipliers and one 64-bit matrix-vector unit	2098	535 MHz(256clks)	1003.9
256-bit matrix-vector multiplier (matrix divided into 4 parts)	4580	520 MHz (128 clks)	1127.4

We have started from 163-bit multipliers and obtained solutions with parameters presented in Table 3.10. The best seems to be our multiplier based on Karatsuba-Ofman trick, utilising three 82-bit matrix-vector multipliers. Quite nice results gives also simple matrix-vector 163-bit multiplier. It is slower than modified version but much smaller, what is more its AT factor is

not much higher. Another conclusion, which can be drawn from the results is that "regular" size solution does not give better results either in terms of area or in terms of speed. However difference between 163 and 256 is rather big thus we could predict such comparison results. Unfortunately another "regular" value close to 163 is 128 which is too small. We have also tried to combine smaller "regular" size multipliers to create 163-bit multiplier (see 5^{th} solution in Table 3.10) however, as could be observed, the results were not very satisfying.

After evaluation of big number multipliers (see tables in this section for results) and multipliers, which can perform 163-bit multiplication, we have concluded that the best solutions which can be achieved with classical methods, are multipliers based on three matrix-vector multipliers. In those solutions the sub-multipliers size is half the size of original input polynomial. To combine partial results obtained from sub-multipliers we base on Karatsuba-Ofman trick. Regarding those conclusions we have created multipliers of size 233, 283, 409, 571. The results are presented in Table 3.11. Next we have switched to analysis of reduction methods to be able to create complete finite field multipliers.

Table 3.11.: 233, 283, 409 and 571-bit multipliers (Virtex-6)

Multiplier	Area [LUT]	Max.freq.(#clks) [MHz]	*AT*
233-bit multiplier built of three 117-bit matrix-vector units	2625	520 MHz (234 clks)	1181.3
283-bit multiplier built of three 142-bit matrix-vector units	3381	535 MHz (284 clks)	1794.8
409-bit multiplier built of three 206-bit matrix-vector units	4834	535 MHz (412 clks)	3722.6
571-bit multiplier built of three 286-bit matrix-vector units	7095	522 MHz (572 clks)	7774.6

Reduction step Generally there exist two reduction methods. One method employs a classic scheme, see Equation 3.3 on page 53, the other employs a specific reduction matrix R (see [25]). However many variations of a classic method optimised for a specific irreducible polynomials (trinomials, pentanomials, equally spaced polynomials) have been proposed (see [36]).

Classic reduction. The classical method in case of polynomials may be interpreted as follows: we look for bits equal to 1 in the upper part of $d(x)$, that is on positions: $(2m-2)$ down to m, and step by step reduce vector $d(x)$, `XOR` vector $d(x)$ by appropriate shift of irreducible polynomial $f(x)$ (see Algorithm 6).

Algorithm 6 Classic reduction (our interpretation)

Input: $d(x)$, $f(x)$
Output: $c(x) = d(x) \bmod f(x)$

```
1: for i = 2m − 2 to m do
2:     if d_i = 1 then
3:         e = shift f by (i − m)          // producing shift of vector e //
4:         d = d XOR e                     // reducing d //
5:     end if
6: end for
7: return d
```

The drawback of this reduction method is that it is very time consuming (we need at least m clock cycles to perform reduction). Utilising properties of special irreducible polynomials one may optimise classical algorithm. The aim of optimisations is usually to decrease number of sequential `XOR` operations needed to reduce polynomial $d(x)$, thus to reduce time needed to obtain result.

Reduction matrix method. [25] As in case of multiplication the matrix approach to reduction was derived. To reduce the product we need spe-

cial reduction matrix R. The reduction matrix method allows significantly speeding up reduction process.
Having polynomial $d(x) = a(x)b(x) = d_0 + d_1x + \cdots + d_{2m-2}x^{2m-2}$ in order to reduce it we partition it in two parts. The lower part containing the least significant bits of d : $d^{(L)}(x) = d_0 + d_1x + \cdots + d_{m-1}$ and the upper part containing the most significant bits of d: $d^{(H)}(x) = d_m + d_{m-1} + \cdots + d_{2m-2}$. Then representing both parts of d as vectors and using an $(m \times m-1)$ matrix R, reduction is performed as follows:

$$c = d^{(L)} + Rd^{(H)}. \tag{3.9}$$

Reduction matrix R is defined in terms of irreducible polynomial $f(x)$, generating the field. Let f_i and $r_{i,j}$ denote coefficients of $f(x)$ and entries of reduction matrix R respectively. Let $r_j = [r_{0,j}, r_{1,j}, ..., r_{m-1,j}]^T$ denote the j-th column of matrix R; $f = [1, f_1, ..., f_{m-1}]^T$ denotes vector representing irreducible polynomial generating the field and let $\downarrow$ denote shift right (shift down) of any vector. Then,

$$r_j = \begin{cases} f \quad for \quad j = 0 \\ r_{j-1}[\downarrow 1] + r_{m-1,j-1}f \quad for \quad j = 1, ..., (m-2) \end{cases} \tag{3.10}$$

Thus we have :

$$c = d^{(L)} + Rd^{(H)} =$$

$$\begin{bmatrix} d_0^{(L)} \\ d_1^{(L)} \\ \vdots \\ d_{m-2}^{(L)} \\ d_{m-1}^{(L)} \end{bmatrix} + \begin{bmatrix} r_{0,0} & r_{0,1} & r_{0,2} & \cdots & r_{0,m-3} & r_{0,m-2} \\ r_{1,0} & r_{1,1} & r_{1,2} & \cdots & r_{1,m-3} & r_{1,m-2} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ r_{m-2,0} & r_{m-2,1} & r_{m-2,2} & \cdots & r_{m-2,m-3} & r_{m-2,m-2} \\ r_{m-1,0} & r_{m-1,1} & r_{m-1,2} & \cdots & r_{m-1,m-3} & r_{m-1,m-2} \end{bmatrix} \begin{bmatrix} d_m^{(H)} \\ d_{m+1}^{(H)} \\ \vdots \\ d_{2m-3}^{(H)} \\ d_{2m-2}^{(H)} \end{bmatrix} \tag{3.11}$$

Matrix approach to reduction yields very good results in terms of hard-

ware design. Many researchers find it even advisable to perform multiplication in any way, using any method, and then to reduce the result using reduction matrix R.

Reduction step - hardware realisation. We have started with analysis of classic not optimised to irreducible polynomial reduction method. However knowing that it is not very efficient we have created just one version of such reduction unit, just to be assured that its efficiency makes it unworthy considering. Created solution (see first solution in Table 3.12 at the end of the paragraph) is rather big and time inefficient. We find that it can be improved, especially in terms of area. According to us also the number of clocks could be reduced but we presume that the smallest number of clocks, which is possible to achieve is m.
Next we have started to optimise classic algorithm reducing product $d(x)$ regarding irreducible polynomials recommended for NIST's fields. Usually for those fields there are defined generators with special properties, e.g. trinomials, pentanomials. Trinomials have only three and pentanomials five coefficients equal to 1, which is very advantageous.

Our version of reduction algorithm for elements of field of size $m = 233$, optimised for trinomial $f(x) = x^{233} + x^{74} + 1$ (recommended in [32]), is as follows:

Similar reduction algorithms optimised for trinomials can be found in [36].

It may seem that Algorithm 7 requires additional multiplications (line 2 and 5), however if one of the operands is known in advance and number of its bits equal to 1 is very low, multiplication is very simple. It can be substituted with few `XOR` operations. Moreover we do not need to reduce those products. So instead of performing costly modular multiplication we perform simple set of `XOR` operations.

Last implemented reduction unit was the one employing reduction matrix R. We propose two types of such solution, sequential (synchronised) and

Algorithm 7 Reduction algorithm (optimised version for $m = 233$))

Input: $d(x)$, $f(x)$
Output: $c(x) = d(x) \bmod f(x)$

1: $e = d[2m-2, ..., m]$ // assign part of vector d to e //
2: $e1 = e \times f$
3: $d1 = d$ `XOR` $e1$ // first reduction step //
4: $e = d1[74 + (m-1), ..., m]$ // assign part of new vector d to e //
5: $e2 = e \times f$
6: $c = d1$ `XOR` $e2$ // second reduction step //
7: **return** c

combinational one. Results obtained for all discussed types of reduction units, for field of size $m = 233$, are presented in Table 3.12.

As we can observe the non-optimised classical unit is the biggest and needs the greatest number of clock cycles to perform reduction operation. Quite promising seems the solution of optimised version of classical unit, especially in terms of speed. As mentioned, number of operations necessary to be performed depends strongly on the form of irreducible polynomial $f(x)$. For $m = 409$ number of operations performed by this type of reduction unit remains the same, for $m = 233$ and 409 it is possible to have trinomials. However for $m = 163,\ 283,\ 571$ number of operations performed increases to 14 (which is still small) due to the fact that for those fields NIST defines pentanomials (there are no trinomials for those fields). Concluding, one may say that optimisations regarding irreducible polynomials are highly recommended because they improve much the overall efficiency of reduction unit.

The most efficient seem to be solutions employing reduction matrix R. In fact that reduction method is very simple. The reduction matrices for trinomials and pentanomials contain a lot of zeroes (see exemplary matrix on Figure 3.6 for trinomial defined for $m = 233$). Thus the whole complicated reduction operations are in fact a set of simple `XOR` operations. The more bits equal to 1 the irreducible polynomial has the more `XOR` operations we

Table 3.12.: Comparison of different types of reduction units for 233-bit multiplier (Virtex-6)

Reduction unit	Area [LUT]	Max.freq.(#clks) [MHz]	AT
Classical not optimised	3528	209 MHz (600 clks)	10128.2
Classical optimised	1165	571 MHz (8 clks)	16.3
Reduction matrix (sequential)	466	1264 MHz* (2 clks)	0.74
Reduction matrix (combinational)	233	1.13 ns	0.26

*the results are the one given by Xilinx ISE, in this case we presume that combining such unit with other will not have impact on the speed

have to perform. The proposed reduction circuits integrate additional mechanisms serving data exchange and communication with multiplier units.

When we use finite field operators for cryptographic purposes we are provided with secure field sizes and irreducible polynomials (NIST) thus it is easier to optimise the reduction process. What is more the recommended field generators are usually trinomials or pentanomials (they contain either only three or five bits equal to 1) whose properties may really simplify reduction process of large numbers.

Two-step finite field multipliers - proposed solutions, summary and comparison The analysis of multiplication and reduction allowed us to create the hardware solutions for complete two-step $GF(^m)$ multipliers. Basing on the obtained results to create a **complete** $GF(2^m)$ **multipliers** we have decided to use:

- as a **multiplier unit**: matrix-vector multiplier utilising three sub-multipliers of half the original input size ($m/2$), employing Karatsuba-Ofman trick,

- as a **reduction unit**: classical optimised to irreducible polynomial reduction unit or the one utilising reduction matrix R.

We present as exemplary the implementation results for our $GF(2^m)$ multipliers where $m = 233$. The results are presented in Table 3.13.

Table 3.13.: Complete classic $GF(2^{233})$ multipliers (Virtex-6)

Multiplier m=233	Area [LUT]	Max.freq. [MHz]	# of clock cycles	AT normalised
multiplier block with classic reduction	3638	302 MHz	264	3.18×10^3
multiplier block with matrix reduction	2862	302 MHz	238	2.25×10^3

3.2.2. Interleaved algorithms

Another group of finite field multiplication algorithms comprises interleaved algorithms. In this type of algorithms instead of performing separately multiplication and reduction we interleave or combine the two operations. Classic interleaved method is based on the following idea:

$$\begin{aligned} c(x) = a(x)b(x) \bmod f(x) &= a(x)\Big(\sum_{i=0}^{m-1} b_i x^i\Big) \bmod f(x) \\ &= \Big(\sum_{i=0}^{m-1} b_i a(x) x^i\Big) \bmod f(x) = b_0 a(x) \bmod f(x) + b_1 a(x)x \bmod f(x) \\ &\quad + \cdots + b_{(m-1)} a(x) x^{(m-1)} \bmod f(x) \end{aligned} \tag{3.12}$$

As in previous two step methods we multiply $a(x)$ by each coefficient of $b(x)$, i.e. we shift $a(x)$, but here before accumulating the partial result we perform its reduction. We shift, reduce and then accumulate each partial result. Thus finally we obtain a result, which is already reduced.

After theoretical analysis of the algorithm it was concluded that its complexity (number of the operations we have to perform) is comparable with a complexity of shift-and-add method combined with standard reduction. The only difference is that here we interleave shifting with reduction. Additionally in previous case maximal number of reductions (divisions/`XOR`) we have to perform is $m/2 - 1$ here even regarding the fact that only half of the partial results have to be reduced it may occur that the number of reduction operations will increase.

Apart this standard formulation of the interleaved multiplication there exist few popular algorithms, which combine multiplication and reduction and by exploiting certain arithmetic properties they speed up a bit process of multiplication in $GF(2^m)$ field. The most worth analysing methods according to us are multiplication with use of Mastrovito matrix and Montgomery multiplication algorithm.

Mastrovito matrix approach. Mastrovito matrix method [62] is an extension of basic matrix-vector approach (see previous section) where $c(x)$ and $b(x)$ are represented in form of m size vectors and $a(x)$ is transformed into $(2m-2) \times m$ matrix. In standard matrix-vector approach we perform two steps:

1. Multiplication $d = Ab =$

$$\begin{bmatrix} d_0 \\ d_1 \\ \vdots \\ d_{m-1} \\ d_m \\ d_{m+1} \\ \vdots \\ d_{2m-3} \\ d_{2m-2} \end{bmatrix} = \begin{bmatrix} a_0 & 0 & 0 & \dots & 0 & 0 \\ a_1 & a_0 & 0 & \dots & 0 & 0 \\ \vdots & \vdots & \ddots & \dots & \ddots & \vdots \\ a_{m-2} & a_{m-3} & a_{m-4} & \dots & a_0 & 0 \\ a_{m-1} & a_{m-2} & a_{m-3} & \dots & a_1 & a_0 \\ 0 & a_{m-1} & a_{m-2} & \dots & a_2 & a_1 \\ 0 & 0 & a_{m-1} & \dots & a_3 & a_2 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \dots & a_{m-1} & a_{m-2} \\ 0 & 0 & 0 & \dots & 0 & a_{m-1} \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ \vdots \\ b_{m-3} \\ b_{m-2} \\ b_{m-1} \end{bmatrix}$$

2. Reduction

$$\left[\begin{array}{c} d_0 \\ d_1 \\ \vdots \\ d_{m-1} \\ \hline d_m \\ d_{m+1} \\ \vdots \\ d_{2m-3} \\ d_{2m-2} \end{array}\right] \Rightarrow \left[\begin{array}{c} \\ d^{(L)} \\ \\ \\ d^{(H)} \\ \\ \end{array}\right] \Rightarrow$$

$$\left[\begin{array}{cccccc} a_0 & 0 & 0 & \dots & 0 & 0 \\ a_1 & a_0 & 0 & \dots & 0 & 0 \\ \vdots & \vdots & \ddots & \dots & \ddots & \vdots \\ a_{m-2} & a_{m-3} & a_{m-4} & \dots & a_0 & 0 \\ a_{m-1} & a_{m-2} & a_{m-3} & \dots & a_1 & a_0 \\ \hline 0 & a_{m-1} & a_{m-2} & \dots & a_2 & a_1 \\ 0 & 0 & a_{m-1} & \dots & a_3 & a_2 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \dots & a_{m-1} & a_{m-2} \\ 0 & 0 & 0 & \dots & 0 & a_{m-1} \end{array}\right] \left[\begin{array}{c} b_0 \\ b_1 \\ b_2 \\ \vdots \\ b_{m-3} \\ b_{m-2} \\ b_{m-1} \end{array}\right] \Rightarrow \left[\begin{array}{c} A^L \\ \hline A^H \end{array}\right] \left[\begin{array}{c} \\ b \\ \\ \end{array}\right]$$

$$\Rightarrow c = d^{(L)} + d^{(H)}R = A^L b + A^H R b \tag{3.13}$$

In Mastrovito matrix method we perform only one step:

$$c = Mb, \tag{3.14}$$

where M is so called Mastrovito matrix. Comparing Equation 3.13 and Equation 3.14 we can see that Mastrovito matrix M is a combination of

A^L, A^H and R matrices [25], that is:

$$c = Mb = (A^L + A^H R)b \tag{3.15}$$

Matrix M construction and storage is very problematic. As it will be presented it strongly affects area and speed of elaborated hardware solution.

There exist many approaches to handling matrix M. For example one may store whole matrix M, however as we have pointed in previous section, such approach requires a lot of resources. The smallest matrix used in ECC is matrix of size 163×163 bits, which is 26569 bits, which is rather big amount of data to store. Moreover matrix M entries depend on variable a value. Thus we would have to create matrix M before each multiplication, which will add time overhead.

Another idea of handling M matrix is the one used to create the two-step $GF(2^m)$ multiplier multiplication unit. That is to work with vectors/registers of size $(2m-2)$ representing columns of required matrix. In our case we have used two registers, one filled with contents of consecutive columns of the matrix A (we have processed columns from left to right) and the other accumulated partial results of multiplication. Varying number of vectors (columns) used, the design behaviour and parameters can be easily changed.

Our new idea for handling matrix M is to partition it, i.e. matrices of which it constitutes, into sub-matrices, to save area and ease optimisation and synchronisation of circuits. The entries of sub-matrices of matrix M are supposed to be calculated on-the-fly during multiplication, from parts of matrices A^L, A^H and R, i.e. from vectors a and f. Our approach may be regarded as a variation of divide-and-conquer technique because we partition large problem of operating on large matrices into a set of problems operating on much smaller matrices. We presume that such manner of division may increase overall efficiency of elaborated circuits and may make the designs flexible, easily adaptable to new tasks.

The chosen size of sub-matrices is 16×16 bits. It was chosen regarding the

analysis performed for two-step multipliers (see previous section). For sizes over 500 bits however, it may be wiser to work with 32×32 sub-matrices blocks.

As mentioned one of the advantages of block structure is its flexibility. In our interpretation of Mastrovito proposition to perform multiplication we use 16-bit sub-multiplier units and we control their work with use of finite state machine (FSM). The FSM controls reseting, starting and switching-off the units. It also controls the order of sub-multiplications. Results of sub-multiplications are independent of each other and can be calculated in arbitrary order. We can group sub-matrices multipliers in different manners and that way easily change computing time (number of clocks needed to perform the multiplication) or somehow the area occupied by the design. We can group sub-matrices in rows spanning several rows of matrix M, we may try to utilise sub-block multipliers as efficiently as possible, we may change the order of sub-multiplications to adapt the circuit to our needs.

Partitioning into sub-blocks allows to simplify the multiplication operation. Observing the structure and placement of vital entries (entries equal to 1) in matrices A^L, A^H and R we may decrease number of operations, which have to be performed to obtain final solution. We may omit in multiplication process operations on those sub-blocks whose multiplication result by part of vector b will be always equal to zero. In fact we omit processing of sub-matrices, which entries are all equal to zero. If we look closely at partitioned A^L, A^H and R matrices (Figures: 3.4, 3.5, 3.6) we can see that there are a lot of zero blocks, blocks in which all entries are equal to zero. Thus we will consider in processing only those sub-matrices, which may influence the final result (those which contain at least one non-zero elements). In fact A^L and A^H matrices are triangular, they are almost half-filled with zeroes. Figures 3.4 and 3.5 show illustrations of matrices partitioning for field of sizes $m = 233$, for other NIST sizes, matrices A^H and A^L look similarly (they differ in number of sub-blocks). The R matrix is different for each

field due to the fact that its contents are derived from irreducible polynomial generating the field. However matrices R for NIST fields are similar in number of non-zero blocks and their placement. On Figures 3.4, 3.5, 3.6 all

A^L	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
0	a_0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	a_1	a_0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	a_2	a_1	a_0	0	0	0	0	0	0	0	0	0	0	0	0
3	a_3	a_2	a_1	a_0	0	0	0	0	0	0	0	0	0	0	0
4	a_4	a_3	a_2	a_1	a_0	0	0	0	0	0	0	0	0	0	0
5	a_5	a_4	a_3	a_2	a_1	a_0	0	0	0	0	0	0	0	0	0
6	a_6	a_5	a_4	a_3	a_2	a_1	a_0	0	0	0	0	0	0	0	0
7	a_7	a_6	a_5	a_4	a_3	a_2	a_1	a_0	0	0	0	0	0	0	0
8	a_8	a_7	a_6	a_5	a_4	a_3	a_2	a_1	a_0	0	0	0	0	0	0
9	a_9	a_8	a_7	a_6	a_5	a_4	a_3	a_2	a_1	a_0	0	0	0	0	0
10	a_{10}	a_9	a_8	a_7	a_6	a_5	a_4	a_3	a_2	a_1	a_0	0	0	0	0
11	a_{11}	a_{10}	a_9	a_8	a_7	a_6	a_5	a_4	a_3	a_2	a_1	a_0	0	0	0
12	a_{12}	a_{11}	a_{10}	a_9	a_8	a_7	a_6	a_5	a_4	a_3	a_2	a_1	a_0	0	0
13	a_{13}	a_{12}	a_{11}	a_{10}	a_9	a_8	a_7	a_6	a_5	a_4	a_3	a_2	a_1	a_0	0
14	a_{14}	a_{13}	a_{12}	a_{11}	a_{10}	a_9	a_8	a_7	a_6	a_5	a_4	a_3	a_2	a_1	a_0

Figure 3.4.: Illustration of A^L matrix partitioning for $m = 233$

zero blocks are marked in gray. Getting rid of those blocks and operations performed on them allows us to save some clock cycles and area.

Proceeding in analysis we also observe that our proposed partitioning into 16×16 sub-matrices makes not only easier to spot zero blocks but also repeating blocks (marked with the same indices on the figures) and thus to optimise the solution. By repeating we mean the ones with same positioning of vital (depending on values of a, f) entries. Such observation minimises number of different sub-multipliers we have to propose for calculation of each sub-product.

Unfortunately for most m's (also for sizes recommended by NIST) the ma-

A^H	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
0	a_{14}	a_{13}	a_{12}	a_{11}	a_{10}	a_9	a_8	a_7	a_6	a_5	a_4	a_3	a_2	a_1	a_0
1	0	a_{14}	a_{13}	a_{12}	a_{11}	a_{10}	a_9	a_8	a_7	a_6	a_5	a_4	a_3	a_2	a_1
2	0	0	a_{14}	a_{13}	a_{12}	a_{11}	a_{10}	a_9	a_8	a_7	a_6	a_5	a_4	a_3	a_2
3	0	0	0	a_{14}	a_{13}	a_{12}	a_{11}	a_{10}	a_9	a_8	a_7	a_6	a_5	a_4	a_3
4	0	0	0	0	a_{14}	a_{13}	a_{12}	a_{11}	a_{10}	a_9	a_8	a_7	a_6	a_5	a_4
5	0	0	0	0	0	a_{14}	a_{13}	a_{12}	a_{11}	a_{10}	a_9	a_8	a_7	a_6	a_5
6	0	0	0	0	0	0	a_{14}	a_{13}	a_{12}	a_{11}	a_{10}	a_9	a_8	a_7	a_6
7	0	0	0	0	0	0	0	a_{14}	a_{13}	a_{12}	a_{11}	a_{10}	a_9	a_8	a_7
8	0	0	0	0	0	0	0	0	a_{14}	a_{13}	a_{12}	a_{11}	a_{10}	a_9	a_8
9	0	0	0	0	0	0	0	0	0	a_{14}	a_{13}	a_{12}	a_{11}	a_{10}	a_9
10	0	0	0	0	0	0	0	0	0	0	a_{14}	a_{13}	a_{12}	a_{11}	a_{10}
11	0	0	0	0	0	0	0	0	0	0	0	a_{14}	a_{13}	a_{12}	a_{11}
12	0	0	0	0	0	0	0	0	0	0	0	0	a_{14}	a_{13}	a_{12}
13	0	0	0	0	0	0	0	0	0	0	0	0	0	a_{14}	a_{13}
14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	a_{14}

Figure 3.5.: Illustration of A^H matrix partitioning for $m = 233$

trices will always contain an "irregular" row and column. Irregular means that it is impossible to partition it into 16×16-bit blocks. That fact increases slightly area and decreases speed of design. But as recommended field sizes are usually primes it is not possible to find such a partitioning, in which all sub-matrices will have equal size.

Our basic algorithm for computing the result of $a(x)b(x) \bmod f(x)$ works as follows: the 16×16-bit sub-matrices are grouped into 16-bit wide rows spanning sixteen rows of matrix M and for each such row 16-bit part of ab product is calculated. In order to save some space, sub-matrices are not stored in the unit, they are calculated on-the-fly during computations basing on parts of incoming a and b operands.

R	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
0	q_0	0	0	0	0	0	0	0	0	q_3	q_4	0	0	0	0
1	0	q_0	0	0	0	0	0	0	0	0	q_3	q_4	0	0	0
2	0	0	q_0	0	0	0	0	0	0	0	0	q_3	q_4	0	0
3	0	0	0	q_0	0	0	0	0	0	0	0	0	q_3	q_4	0
4	q_1	0	0	0	q_0	0	0	0	0	q_5	q_6	0	0	q_3	q_{x4}
5	q_2	q_1	0	0	0	q_0	0	0	0	0	q_7	q_6	0	0	0
6	0	q_2	q_1	0	0	0	q_0	0	0	0	0	q_7	q_6	0	0
7	0	0	q_2	q_1	0	0	0	q_0	0	0	0	0	q_7	q_6	0
8	0	0	0	q_2	q_1	0	0	0	q_0	0	0	0	0	q_7	q_{x6}
9	0	0	0	0	q_2	q_1	0	0	0	q_0	0	0	0	0	q_{x7}
10	0	0	0	0	0	q_2	q_1	0	0	0	q_0	0	0	0	0
11	0	0	0	0	0	0	q_2	q_1	0	0	0	q_0	0	0	0
12	0	0	0	0	0	0	0	q_2	q_1	0	0	0	q_0	0	0
13	0	0	0	0	0	0	0	0	q_2	q_1	0	0	0	q_0	0
14	0	0	0	0	0	0	0	0	0	q_2	q_1	0	0	0	q_{x0}

Figure 3.6.: Illustration of R partitioning matrix for $m = 233$

In order to provide the best solution we have analysed few variations of the unit. Initially we have implemented equation $c = A^L b + A^H Rb$. That is, the solution calculated separately results for $A^L b$ and $A^H Rb$ and eventually the two partial solutions were `XOR`ed to obtain final result c. Resulting hardware unit is very efficient however it seems to be rather big. Thus to improve overall efficiency of the solution we have implemented equation $c = Mb$. Instead of calculating on-the-fly contents of sub-matrices A^L and $A^H R$ we are providing on-the-fly contents of M's sub-matrices to sub-multipliers. Such approach has visibly improved area of the solution. Results for both solutions are presented in Table 3.14. It is possible in both cases to modify number of clock cycles needed to perform the operation. Number of clock cycles can be modified by varying the order and way, in which sub-multiplier

units are utilised. Initially we have grouped them into rows spanning sixteen rows of M (A^L and $A^H R$) consisting up to fifteen sub-blocks, but the sub-matrices may be grouped into longer or shorter chains.

Table 3.14.: Mastrovito matrix approach solution

Virtex-6 XC6VLX240T	$(A^L b + A^H R)b$	Mb
area(A)[LUT]	5014	3760
max.frequency	297 MHz	276 MHz
time of execution (T)[clock cycles]	65	75
efficiency AT	1097	1021

Computations of multiplication results of each sub-matrix are controlled by means of finite state machine. At first the FSM had 15 states (for $m = 233$, there are 15 blocks in a matrix M row) and we have used separate sub-units for calculation of each 16-bit of final result. Each 16-bit part of final result is calculated in separate clock cycle, according to the following equations:

$$\begin{aligned}
c_0 &= (A^H R_{(0,0)} + A^L_{(0,0)})b_0 + (A^H R_{(0,1)} + A^L_{(0,1)})b_1 + \\
&\quad \cdots \qquad + (A^H R_{(0,m/16)} + A^L_{(0,m/16)})b_{m/16} \\
c_1 &= (A^H R_{(1,0)} + A^L_{(1,0)})b_0 + (A^H R_{(1,1)} + A^L_{(1,1)})b_1 + \\
&\quad \cdots \qquad + (A^H R_{(1,m/16)} + A^L_{(1,m/16)})b_{m/16} \\
&\vdots \\
c_{m/16} &= (A^H R_{(m/16,0)} + A^L_{(m/16,0)})b_0 + (A^H R_{(m/16,1)} + A^L_{(m/16,1)})b_1 + \\
&\quad \cdots \qquad + (A^H R_{(m/16,m/16)} + A^L_{(m/16,m/16)})b_{m/16},
\end{aligned} \tag{3.16}$$

where c_i denotes 16-bit chunk of final result $c(x)$. Additionally we have to compute partial results for "irregular row and column" and combine it with

previously obtained values. Last row equation:

$$c_{m/16+1} = (A^H R_{(m/16+1,0)} + A^L_{(m/16+1,0)})b_0 + (A^H R_{(m/16+1,1)} + A^L_{(m/16+1,1)})b_1 + \cdots + (A^H R_{(m/16+1,m/16+1)} + A^L_{(m/16+1,m/16+1)})b_{m/16+1}. \tag{3.17}$$

Last column parts:

$$\begin{aligned} {c^{m/16+1}}_0 &= (A^H R_{(0,m/16+1)})b_{m/16+1} \\ {c^{m/16+1}}_1 &= (A^H R_{(1,m/16+1)})b_{m/16+1} \\ &\vdots \\ {c^{m/16+1}}_{m/16} &= (A^H R_{(m/16,m/16+1)})b_{m/16+1}, \end{aligned} \tag{3.18}$$

The second version of our Mastrovito multiplier was optimised according to the fact that some 16×16-bit sub-blocks structures are similar. Thus we have used only one of each type of sub-multiplier. We have decreased that way the number of instantiations of sub-units but increased the number of states of the controlling FSM. Figure 3.7 shows matrix M partitioning for $m = 233$. There are marked blocks requiring same sub-multiplier unit. In total in our new solution, for $m = 233$, we need seven different simple sub-multipliers, before we have utilised eleven sub-multipliers. On the Figure 3.7 we may observe, which blocks may be multiplied with use of the same sub-multiplier unit (one with corresponding names).

It is easy to observe that there exist many variations of the order of sub-multiplications, which may be very useful for physical security purposes, see Chapter 4. We can create longer multiplication controllers (finite state machines), increase or decrease number of instantiation of sub-multipliers or change the order of multiplications.

We have performed small analysis of variations of our solution and in terms of efficiency we found that solution containing longer finite state machines but single instances of each sub-multiplier gives the best results, especially in terms of area. However we have not tried even half of possible

M	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
0	M0	M0	M0	M0	M0	M0	M0	M0	M0	M0	M1	M1	M1	M1	Mc
1	M0	M0	M0	M0	M0	M0	M0	M0	M0	M0	M0	M1	M1	M1	Mc
2	M0	M0	M0	M0	M0	M0	M0	M0	M0	M0	M0	M0	M1	M1	Mc
3	M0	M0	M0	M0	M0	M0	M0	M0	M0	M0	M0	M0	M0	M1	Mc
4	M2	M2	M2	M2	M4	M4	M4	M4	M4	M4	M4	M4	M4	M4	Mc
5	M2	M2	M2	M2	M2	M4	M4	M4	M4	M4	M4	M4	M4	M4	Mc
6	M0	M2	M2	M2	M2	M2	M4	M4	M4	M4	M4	M4	M4	M4	Mc
7	M0	M0	M2	M2	M2	M2	M2	M4	M4	M4	M4	M4	M4	M4	Mc
8	M0	M0	M0	M2	M2	M2	M2	M2	M4	M4	M4	M4	M4	M4	Mc
9	M0	M0	M0	M0	M2	M2	M2	M2	M2	M3	M3	M3	M3	M3	Mc
10	M0	M0	M0	M0	M0	M2	M2	M2	M2	M2	M3	M3	M3	M3	Mc
11	M0	M0	M0	M0	M0	M0	M2	M2	M2	M2	M2	M3	M3	M3	Mc
12	M0	M0	M0	M0	M0	M0	M0	M2	M2	M2	M2	M2	M3	M3	Mc
13	M0	M0	M0	M0	M0	M0	M0	M0	M2	M2	M2	M2	M2	M3	Mc
14	MR	MR	MR	MR	MR	MR	MR	MR	MR	MR	MR	MR	MR	MR	MR

Figure 3.7.: Illustration of Mastrovito matrix partitioning for $m = 233$

combinations of sub-units thus there may exist more efficient versions of our design. The most efficient solution, obtained after few experiments, takes 3760 LUTs and can work with frequencies up to 276 MHz, number of clocks needed to perform the operation is 75.

Montgomery multiplication algorithm. The second most popular interleaved multiplication method is Montgomery method [71]. The algorithm is constructed in a specific way in order to avoid most costly operations. Instead of performing $c(x) = a(x)b(x) \bmod f(x)$ it performs

$$c(x) = a(x)b(x)r^{-1}(x) \bmod f(x). \tag{3.19}$$

The Montgomery method assumes operating on Montgomery versions of the operands throughout the chain of operations and recovering original (proper) value at the end of computation. For fair comparison with other multiplication solution we present the complete multiplication process needed to obtain proper product.

To obtain the complete result of modular multiplication $a(x)b(x) \bmod f(x)$ we must run the Montgomery algorithm (see Algorithm 8), twice, at first for $a(x)b(x)$ and then for the obtained result $d(x)$ and fixed value $r^2(x) \bmod f(x)$. Thus operation $c(x) = a(x)b(x) \bmod f(x)$ comprises in fact two steps:

1. $d = MontMult(a, b)$
2. $c = MontMult(d, r^2 \bmod f)$

Utilising Montgomery method for chain of computations to perform multiplication we run $MontMult(a, b)$ once (we perform just first step). At the end of computations we perform second step to recover from Montgomery representation. The algorithm for Montgomery multiplication ($MontMult(a, b)$) is presented below.

Algorithm 8 Montgomery multiplication algorithm ($MontMult(a, b)$) [50]

Input: $a(x), b(x), r(x), f(x), f'(x)$
Output: $c(x) = a(x)b(x)r^{-1}(x) \bmod f(x)$
1: $t(x) = a(x)b(x)$
2: $u(x) = t(x)f'(x) \bmod r(x)$
3: $c(x) = [t(x)$ `XOR` $u(x)f(x)]/r(x)$
4: **return** c

To utilise the algorithm we need three additional values, which depend on the value of the irreducible polynomial. We need polynomials $r(x)$, $r^2(x) \bmod f(x)$, $f'(x)$. Element $r(x)$ is a fixed element. Requirements for element $r(x)$, given by Montgomery, are as follows:

- it should be an element of the field,
- it should be relatively prime and $gcd(r(x), f(x)) = 1$.

According to [50] for best results for field $GF(2^m)$ it is chosen to be a simple polynomial x^m. To find polynomials $r^{-1}(x)$ and $f'(x)$ we use Bezout's iden-

tity: since $r(x)$ and $f(x)$ are relatively prime there exist also polynomials $r^{-1}(x)$ and $f'(x)$ such that $r(x)r^{-1}(x) + f(x)f'(x) = 1$. For cryptographic application irreducible polynomials are defined in standards. Thus for given m we calculate $r(x), r^2(x) \bmod f(x), f'(x)$ and store them, we do not need to precalculate those vectors before each multiplication.

The most complicated in the Algorithm 8 is first operation where we need to perform multiplication of two large binary vectors. In the second line we also need to perform multiplication but this time we may substitute it with few `XOR` operations because we know $f'(x)$ in advance. What is more it has, for irreducible polynomials recommended by NIST, low number of coefficients equal to 1, thus the operation gets really simple. In the modulo operation in this line we just cut out all elements of order higher or equal to m due to the fact that $r(x) = x^m$. In third line we can again substitute multiplication with a chain of simple additions. The division operation in this line is a simple shift right by m positions.

Summing up we may say that the algorithm combines multiplication and reduction steps. However it is not easy to distinguish standard reduction and standard multiplication processes and it is impossible to separate them. Although looking at the complete algorithm for finite field multiplication (see Algorithm 9) one can observe that in its first line we perform multiplication and all the remaining operations are responsible for reduction.

It is easy to observe that if the irreducible polynomial is unknown/variable we need to add to our solution units, which will precalculate additional values needed. In fact that operations maybe much more complicated than multiplier itself. Knowing irreducible polynomial, which is the case for typical cryptographic applications, we may perform optimisations and forget about additional precomputations. That way we gain a lot on efficiency. The Montgomery method based algorithm calculating the modular product $c(x)$ is given in Algorithm 9.

To perform multiplication in second line of Algorithm 9 we have used multiplier based on shift-and-add method but different from the one proposed

Algorithm 9 Modular multiplication algorithm based on Montgomery method [50]

Input: $a(x), b(x), r(x), f(x), f'(x), r^2(x) \bmod f(x)$
Output: $c(x) = a(x)b(x) \bmod f(x)$

1: // $MontMult(a, b)$ //
2: $t(x) = a(x)b(x)$
3: $u(x) = t(x)f'(x) \bmod r(x)$
4: $d(x) = [t(x)$ `XOR` $u(x)f(x)]/r(x)$
5: // $MontMult(d, r^2 \bmod f)$ //
6: $t(x) = d(x)(r^2(x) \bmod f(x))$
7: $u(x) = t(x)f'(x) \bmod r(x)$
8: $c(x) = [t(x)$ `XOR` $u(x)f(x)]/r(x)$
9: **return** c

previous section. For $m = 233$ we partition vector b into 16-bit words (we add bits equal to 0 on MSB positions if necessary), multiply sequentially a by all parts of vector b (we need to perform 15 multiplications) and sequentially cumulate partial results. In Table 3.15 we compare the performance of Montgomery solution utilising such multiplier and utilising multiplier based on matrix-vector approach (see Section 3.2.1). To construct full finite field multiplier based on Montgomery method we may use different types of multipliers but we have to remember that they strongly influence the solution. In fact large binary vector multipliers perform half of the calculations (or even more) required by the complete Montgomery finite-field multiplier.

All other multiplications needed to perform the algorithm, multiplication by $r(x), r^2(x) \bmod f(x), f'(x), f(x)$ are simpler due to the fact that we know the values of those operands. Thus we may substitute those multiplications with short chains of `XOR` operations as already mentioned. Summing up, knowing the irreducible polynomial generating the field we may say that the algorithm comprises a set of very simple operation. The only difficulty here is that we operate on large numbers, thus we have to manage large binary vectors.

Table 3.15.: Montgomery finite field multipliers ($m = 233$) (Virtex-6)

	area [LUT]	max.freq [MHz]	# clock cycles	multiplier used	AT
1	3197	338	270	Shift-and-add multiplier built of 233x16-bit multipliers area: 2308 LUT max.f: 323MHz	2554
2	3730	302	244	Matrix-vector multiplier built of three 117-bit multipliers area: 2625 LUT max.f: 302MHz	3014

3.2.3. Summary, conclusions and comparison

Comparing our solution with already existing ones is not easy. Available literature does not always contain all necessary data. The designs are usually not fully described. Many references does not contain practical results but theoretical description, i.e. evaluation of predicted number of gates used and probable delay. As the solution depends not only on the algorithm used but also on the way it is described in HDL it is not easy to say, which solutions are the best. Even if we would try to implement other described just theoretically algorithms we may get different results than predicted and probably achieved by the inventors. In Table 3.16 we present our exemplary solutions. In Table 3.17 we present the solutions found in accessible literature. Looking on both tables we may conclude that our units are rather fast and small.

We present in Table 3.16 implementation results for Virtex-6 device of first versions of our finite field multipliers. We present all the solutions for exemplary field size $m = 233$. To ease the comparison of our solutions we have calculated the AT factors. It seems that the best in terms of efficiency is Mastrovito multiplier. The Mastrovito multiplier outperforms the rest

Table 3.16.: Our exemplary solutions

Algorithms	area [LUT]	freq. [MHz]	clock cycles	AT
Classical 1	3638	302	264	3.18
Classical 2	2862	302	238	2.25
Mastrovito	3760	297	75	0.95
Montgomery (full)	3197	338	270	2.55

mainly due to the fact that it needs only 75 clock cycles to perform the multiplication.

In Table 3.17 there are presented results for other existing solutions we have found described in literature. Unfortunately it is difficult to calculate the AT factor for them. Mainly due to the fact that we have insufficient data. The other reason is that area is interpreted (not in all cases) differently than in our work thus comparison of such AT factors maybe inadequate.

The second goal of our research is to secure the elaborated efficient arithmetic operators. That is why their versions presented here are not fully optimised. We have left some space (some "gaps") for the countermeasures. What is more we have structured our designs in such a way that would be possible to secure our operators. Some of those additional mechanisms may occur to be useless however at this stage we will not suppress them. The final fully optimised and secured operators will be described in consecutive chapters. Results presented in this chapter have been published at conferences and in journal [82, 81, 83].

Table 3.17.: Existing solutions

Solution	m	Device	Area	Max.freq. /delay	T
[21]	256	Virtex II 2000-6	5267 LUT	44.91 MHz	5.75 us
	1033				23.07 us
[110]	1024	XCV2000E-6	4355 CLB	100.4 MHz	-
		XC40150XV-7	8339 CLB	44.4 MHz	-
		XC4VFX100-10	2793 CLB	150.5 MHz	-
[27]	233	XC2V-6000-4	415 slices	-	2.42 us
[75]	233	Stratix EP1S40F780C5	3728 LE	4.04 ns	12 cycles
	283		3396 LE	3.66 ns	20 cycles
[114] by [75]	233	Stratix EP1S40F780C5	3353 LE	6.91 ns	16 cycles
	283		3118 LE	6.95 ns	20 cycles
[34]	233	XC2V-6000 FF1517-4	37296 LUT	77 MHz	-
			11746 LUT	90.33 MHz	-
			36857 LUT	62.85 MHz	-
			45435 LUT	93.20 MHz	-
[95]	191	XCV2600E	8721 CLB	-	82.4 us
[18]	88	Altera EP2S60	6644 ALUT	-	
[30]	163	Virtex XCVL330	201,989 LUT 214,703 DFF	241 MHz	-
			1471 LUT 982 DFF	241 MHz	- -
[57]	283	Virtex 4 XC4VFX140	1781 CLB 2156 FF 3367 LUT	246.670 MHz	- - -
	1132		25,955 CLB 32,578 FF 48,591 LUT	248.447 MHz	- - -

4. Physical security of ECC cryptosystems

Modern cryptographic devices suffer from more threats than their predecessors. The mathematical cryptographic systems are now very secure. As claimed in [68], there exist a mathematical/theoretical impassive barrier. Organisations such as NIST or SECG develop and issue standards for cryptographic systems. Those organisations verify the security level of cryptographic systems very often in order to provide up-to-date parameters of cryptosystems ensuring certain level of security and maintaining the mathematical barrier.

In todays world mathematical cryptographic system needs to be implemented on some device in order to be useful. The most popular devices used for cryptographic purposes are microprocessors, VLSI circuits (FPGA, ASIC) or smart cards. Until very recently (beginning of 90's), if such device contained secure, according to standards, cryptographic system, it was considered as unbreakable (i.e. safe). Nowadays the security of whole system relies not only on the security of algorithms and protocols, but also on the security of their implementation [80]. It was found that cryptographic devices leak information during their activity, i.e. they need certain time to perform the operation, they consume specific amount of power and they emit electromagnetic radiations. The leaking information was always considered as useless noise. Unfortunately cryptanalysts observing work of implemented cryptographic systems noticed that the leaking information maybe useful for discovering secret data/keys on which the cryptographic device operates. It was presented that the information depends on the ma-

nipulated operands values. The observation process is called side-channel analysis and is now a serious threat for modern cryptosystems. Its main advantage is that it is rather cheap and in many cases does not require direct access to the device [61, 76].

Side-channel attacks - an introduction Eavesdropping of devices is not a recent idea. For years people have been eavesdropping mechanical devices such as safe locks. It is for example possible to open mechanical lock through analysis of sound of the lock's wheels dialed in a certain manner. To do that the burglar must possess deep knowledge about lock being manipulated, nevertheless then it is possible to open the lock without leaving a trace. Fortunately the digital devices can counteract such attacks and if the developer of the device is aware of those threats he will surely try to secure the device developed.
The notion of side-channel and the idea to eavesdrop information leaking from electric material was proposed in 1918 by H.Yardley and his team [89, 108]. Later in mid-thirties IBM typewriter was studied and the study indicated that the information leakage resulting from the device activity cannot be neglected and constitutes a serious threat [89]. Afterwards in 70's in USA, a TEMPEST program [77] was initiated to counteract threats resulting from leaking information, it concerned information leaking through the electromagnetic radiations. The early research was concerned on the electromagnetic radiations mainly due to the fact that intercepting electromagnetic radiations did not require direct access to the spied device.

The idea of analysis of side-channel information leakage started to be attractive to cryptanalysts of modern cryptosystems in the 90's. Around this time most cryptographic systems were standardised and it was even defined when there will be available enough computing power to break the system working on certain size of keys. Around that time the mathematical security barrier was also set thus the cryptanalysts started to look for new ways of retrieving secrets. Their interest turn to observation of implementations

of cryptographic algorithms and the side-channel information leakages.

According to [80] a **side-channel** can be explained as follows: *A device can have different types of outputs. If such outputs unintentionally deliver information (about the secret key), then the outputs deliver side-channel information and are called side-channel outputs (or side-channels).*

The analysis of side-channels information occurred to be a very efficient and in the same way cheap manner of stealing the secrets thus it gained a lot of interest.

The most popular types of developed side-channel attacks are:

- timing attack - analyses the device running time [51],
- electromagnetic attack - analyses electromagnetic field emitted by the device during its work [89],
- power analysis attacks - analyses power consumption of the device during its work [61, 52],

In our researches we focus on securing our arithmetic units against some power analysis attacks, which seem to gain a lot of attention nowadays [61]. We find that our ideas for protections against power analysis may be extended to protections against electromagnetic field analysis.

Power Analysis Attacks All the details presented here on power analysis attacks are based on [61, 80, 52] and some recent summaries of those type of attacks, such as [26, 86].
The power analysis attacks and their effectiveness were introduced in 1998 by Kocher *et al.*, in [52]. Below we present the idea of such attacks.

Power Analysis Attacks [61]
The attacks rely on the fact that instantaneous power consumption of a cryptographic device is correlated with data being processed and operation being performed

Types of power analysis attacks (according to [53, 26]):

- Simple Power Analysis (SPA) [52], goal: reveal the secret using few power traces; attacks exploit key-dependent differences (patterns) within a trace; *"In SPA attacks, a device's power consumption is analysed mainly along the time axis. The attacker tries to find patterns in a single trace."* [39]
- Differential Power Analysis (DPA) [52], requires many power traces, it is usually necessary to physically possess a device to be able to obtain large set of traces; *"In DPA attacks, the shape of the traces along the time axis is not as important. DPA attacks analyse how the power consumption at fixed moments of time depends on the processed data."* [39]
- Correlation attack [11], improved DPA attack; the measurements allow to predict at once more than one bit (e.g. 4 bits); more optimised DPA,
- Template attack [16], requires to physically possess cryptographic device; attacker constructs a model of the wanted signal source including the characterisation of noise then he compares it with measurement values; improved DPA,
- Refined Power Analysis (RPA) [33], improved DPA
- Carry-based attack [29]
- others

Main principles of the power analysis attacks [86]:
In order to perform power analysis attacks, the attacker analyses the attacked device's power traces. In SPA the adversary tries to deduce secret information by observation of variations and repetitive patterns in the obtained power traces (usually a single power trace). In DPA and more complex attacks the adversary requires a model of cryptographic device to be attacked. The better the model, e.g. low-level descriptions (netlists), the more advantageous for the attacker. The device model is used to predict certain intermediate values, depending on inputs, outputs and secret key, which are assumed to appear during computation. The dependency of those

intermediate values on the secret key implies that it is possible to guess at least a part of the key. Those intermediate values allow for creating a hypothetical power consumption model of the attacked device. To reveal the part of the secret key depending on chosen intermediate values the attacker compares hypothetical model with the measured one, for illustration of the DPA attack idea see Figure 4.1. To perform DPA the adversary needs many power traces.

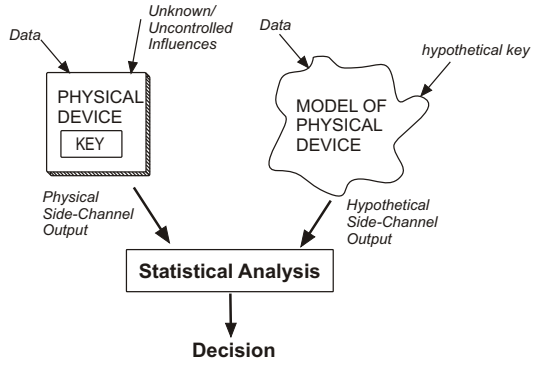


Figure 4.1.: Differential power analysis principle [80]

Summing up, according to [80], the simple side-channel analysis exploits the relationship between the executed instructions and the side-channel output. Differential side-channel analysis exploits the relationship between the processed data and the side-channel output.

Popular ECC curve operation level countermeasures The ECC systems and their main protocols are described in Chapter 2, there we have also mentioned which is the most vulnerable to SCAs operation in ECC systems. The operation is a scalar point multiplication $[k]P$, multiplication of the point on the curve P by a large scalar k. Scalar k is usually the private key or an ephemeral (secret) key. It has been shown that the scalar multiplication of a secret value with a known elliptic curve point is vulnerable to simple and differential side-channel analysis. A successful attack on this operation

results in revealing scalar k (secret key) thus in breaking the cryptographic system.

In order to perform $[k]P$ operation one needs to perform a set of point addition ($P+Q$) and point doubling ($2P$) operations. The simplest algorithm for performing scalar multiplication $[k]P$ is *double-and-add* algorithm, see Algorithm 10.

Algorithm 10 Double-and-add algorithm (right-to-left binary algorithm) [36]

Input: $k = (k_{t-1}, ..., k_1, k_0)_2$, $P \in E(\mathbb{F}_q)$
Output: $[k]P$

```
1: Q ← ∞
2: for i = 0 to t − 1 do
3:     if k_i = 1 then
4:         Q ← Q + P
5:     end if
6:     P ← 2P
7: end for
8: return Q
```

Looking at the algorithm it appears obvious that straightforward implementation of *double-and-add* is a very vulnerable algorithm. The type of operation performed in the algorithm during each step depends on the value of k. Depending on the value of bit of secret k the algorithm performs either point addition and point doubling or only point doubling. If those operations have different power traces, which is usually the case, then the adversary analysing power consumption of the device implementing such algorithm is able to deduce the secret key easily. Seeing the sequence of performed doublings and additions the adversary is able to derive the secret key.

For us the countermeasures for ECC systems against side-channel analysis have generally two goals. One is to mask/hide sequences of doubling and addition operations, to make impossible to deduce from power traces the operations performed. The second is to remove as much as possible

the dependency of manipulated operands values on the power consumption. There exist and are still developed many countermeasures. There are hardware countermeasures and algorithmic countermeasures. To protect against SPA there exist the following types of algorithmic countermeasures [39]:

- unification of the addition formula [13, 12] or alternative parameterizations [40, 104, 9];
- insertion of dummy instructions or operations [19, 17];
- utilisation of "regularly" behaving algorithms (so called "atomicity") [104, 79, 70, 13, 38, 28].

To protect further the device against DPA it is suggested to [39, 19, 41, 15]:

- randomise base-point P,
- randomise/recode secret scalar k

All those countermeasures aim at goals described above. For example unification of addition formula aims to unify addition and doubling in terms of number, order and type of finite-field operations needed to perform $2P$ or $P+Q$. Other countermeasures manipulate the order of sequence of $2P$ and $P+Q$ operations needed to perform $[k]P$. For example, algorithm *double-and-add*, see Algorithm 10, is modified to perform additional, dummy point addition each time it is executed, see Algorithm 11: *double-and-add always*.

Algorithm 11 Double-and-add always algorithm [19]

Input: $k = (k_{t-1}, ..., k_1, k_0)_2$, $P \in E(\mathbb{F}_q)$
Output: $[k]P$
1: $Q_0 \leftarrow P$; $Q_1 \leftarrow \infty$
2: **for** $i = t-1$ to 0 **do**
3: $\quad Q_0 \leftarrow 2Q_0$
4: $\quad Q_1 \leftarrow Q_0 + P$
5: $\quad Q_0 \leftarrow Q_{k_i}$
6: **end for**
7: **return** Q_0

The DPA countermeasures "mask" the chain of $2P$ and $P+Q$ operations

by randomising the base-point P or scalar k. There are either specific values added to P, k, which maybe easily subtracted at the end, or there exist various methods for k recoding (NAF, DBNS). By recoding the key the sequence of doublings and additions is randomised. With DBNS for example it is possible to have lot of distinct chains of additions and doublings using the same key [15].

According to [39] preventing side-channel power analysis is a two step process. At first the device needs to be secured against SPA and then against DPA. That way it should be impossible to mount a successful power analysis attack on the device. For ECC many SCAs and various countermeasures or protections against them (see [39]) have been proposed. For instance, addition chains allow performing only one type of curve-level operation (point addition) during scalar multiplications [14]. In [15] randomized and very redundant representations of the scalar k are used. All yet proposed protections are at the curve-level not the finite-field one.
Efficient and secure computation units for finite-field arithmetic are important elements of ECC processors. It was already mentioned few years ago in [39] that *"each elliptic curve operation itself consists of a sequence of field operations. In most implementations, the standard sequence of field operations in point addition differs from that in point doubling. Every field operation has its unique side-channel trace. Hence, the sequence of field operations of point addition has a different side-channel pattern than that of point doubling"*. However nobody yet, according to known references, tried to secure finite-field arithmetic operations against information leakage. All the efforts were put to randomise or unify the sequences of those operations and curve-level operations.
We find that securing finite-field arithmetic operations should increase the security of the ECC system. It may even occur that some countermeasures of higher layers of cryptosystem (see Chapter 2) will yield better results when mounted on secured arithmetic operators.

Our goal is to secure the operators in such a way that finally the sequence of finite field operations needed to compute $2P$ or $P + Q$ will yield either unified or random power traces. We want that observing the sequence of e.g. finite-field multiplications it is impossible to distinguish the beginning and end of a single multiplication thus to identify point addition or point doubling. We find that it is possible to flatten/unify power consumption of finite-field arithmetic operations and that it is also possible, in case of some algorithms, to randomise the current signature of finite-field operation, i.e. each time a single operation is executed it will have different current trace.

4.1. Physical security of hardware $GF(2^m)$ arithmetic operators

In this chapter, we investigate protections against some power analysis attacks at the field level in $GF(2^m)$ multiplication algorithms and their architectures dedicated for ECC systems. All other $GF(2^m)$ arithmetic operations needed to multiply points of elliptic curves, such as squaring or inversion, can be performed with use of multiplication and addition in the field [96]. According to this and the claim that finite-field arithmetic operators are crucial for ECC system, see Chapter 2, it follows that finite-field multiplication operators are significant units of ECC system. The finite-field addition operation is very simple and when implemented on reconfigurable circuits it can be performed parallely to all other finite-field elements operations. Due to the fact that the multiplication operation is very costly in terms of time and area and have to be performed many time during scalar multiplication operation, we presume that it has a huge impact on the operations performed in ECC system not only in terms of efficiency but also in terms of security.
In previous chapters we have proposed very efficient multipliers for $GF(2^m)$, in this chapter we analyse the security of elaborated operators and pro-

pose protections and countermeasures against some power analysis attacks. Those protections can be easily extended to protections against electromagnetic attacks. The proposed security modifications are not autonomous countermeasures for ECC systems but an additional protection element, which should enhance higher-level (curve-level) countermeasures. Results presented in this chapter have been published at WAIFI 2012 conference [84].

The cryptanalysts concern two types of power consumption leakage [80]:

- *transition count leakage* - related to the number of bits that change their state at a time.
- *the Hamming weight (HW) leakage* - related to the number of 1-bits, being processed at a time.

Here we concern *transition count leakage* (*Hamming distance* $= HW(t+1) - HW(t)$) due to the fact that in VLSI circuits instantaneous power is linked with the number of useful transitions in the operator. Useful transitions are the theoretical changes of bit state during the operation (from one clock cycle to the next one). This is also called useful circuit activity. To estimate information leakage in typical $GF(2^m)$ multipliers, we have accurately measured their useful activity [84].

Power analysis based SCAs use possible correlations between internal secret values (e.g. keys) and information leakage related to instantaneous power of the executed operations (see [61] for details).

Definition 4.1.1. ***Instantaneous power*** *at time t is $P_{DD}(t) = i_{DD}(t) \times V_{DD}$, where $i_{DD}(t)$ is the instantaneous current and V_{DD} is the power supply. Power consumption components are:* ***static power*** *and* ***dynamic power****. See [112, Sec.4.4] for circuit-level details in CMOS circuits.*

Static power does not depend on circuit activity and is not used in this work. **Dynamic power** appears due to circuit activity: charging and discharging load/parasitic capacitances and short-circuit currents. It strongly depends on the executed operations and data values. Dynamic power vari-

ations are used as a source of information leakage for power attacks.

Dynamic power components are: *useful* activity and *parasitic* activity as illustrated on Figure 4.2. Useful (or theoretical) activity is due to complete and stable transitions required by computations from one clock cycle to the next one (i.e. $0 \rightarrow 1$ and $1 \rightarrow 0$ for each bit). Parasitic (or glitching) activity is due to non-useful transitions. For instance, in case of non-equal arrival times for a gate inputs, the output may have multiple transitions before reaching a steady state.

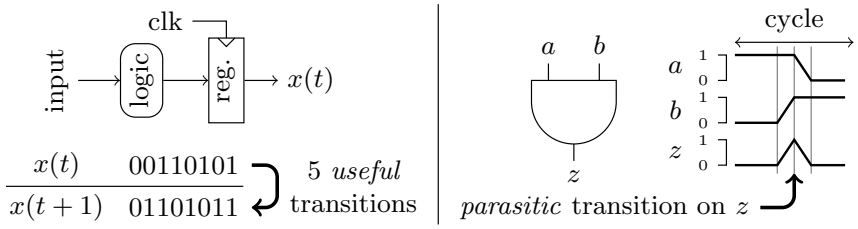


Figure 4.2.: Useful (left) and parasitic (right) transitions.

Parasitic activity in $GF(2^m)$ multipliers is small. This is not the case for all arithmetic operators (e.g. operators in high-performance CPUs [106]). In $GF(2^m)$ arithmetic units, the logical depth is small. Power consumption of memory elements (e.g. flip-flops) used in $GF(2^m)$ multipliers is important compared to power consumption in logic gates. In this work, we only focus on useful activity as a large contribution to $i_{\mathrm{DD}}(t)$.

Several methods can be used to evaluate useful activity: cycle-accurate and bit-accurate (CABA) simulation [31] of a low-level architecture description, electrical simulation or FPGA emulation. Fast high-level behavioral simulation is not sufficient to catch cycle-accurate and bit-level coding aspects. As the target operators have large operands (e.g. 160 to 600 bits for ECC) and long computations, CABA simulation would be too slow. This is even more critical with electrical simulation. Thus we use FPGA emulation for evaluating useful activity.

An activity counter was attached to each monitored signal [107] which

counts the number of useful transitions as illustrated on Figure 4.3. The `D` flip-flop and the `XOR` gate produce `1` for each useful transition between $s(t+1)$ and $s(t)$. The k-bit counter accumulates transitions counts (k depends on test vector length).

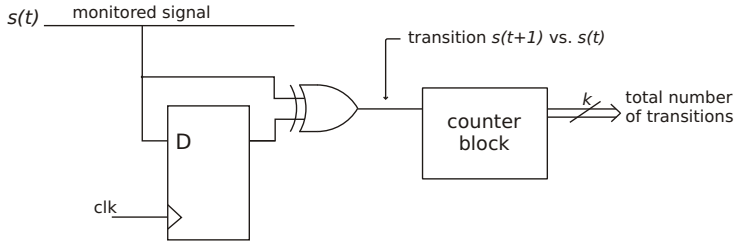


Figure 4.3.: Activity counter architecture for a 1-bit signal $s(t)$ (control not represented).

Comparisons with electrical simulations in [107] show that this is reasonable assumption for small parasitic activity. At the end of this chapter we compare FPGA emulations result with current measurements to show that the assumption holds. We insert activity counters at the output of each internal register and for each signal of the multiplier. Outputs of all `XOR` gates (radix-1 representation of transitions number) are compressed into a binary value as the total transitions count for cycle t. This value is monitored using ChipScope Pro tool from Xilinx. ChipScope Pro enables observation of internal signals of FPGA device during its work. It is possible to record the monitored value changes and analyse it later with other tools.

One of our goals was to design efficient finite-field multipliers in such a way that their architectures can be easily modified to add protection against SCAs. We have analysed many algorithms, with different variants, to be able to take and combine those parts, which will allow us to create the most efficient algorithm fulfilling assumed requirements. The analysis and final solutions are presented in Chapter 3. As a result of our study, we have prepared three efficient $GF(2^m)$ multipliers based on three different algorithms:

classical two-step, Montgomery and Mastrovito algorithm. In this chapter we analyse the security and possibility of inserting various countermeasures for each of those solutions. Using FPGA emulation, it is possible to quickly and accurately evaluate useful activity in $GF(2^m)$ finite-field multipliers for large and relevant test vectors (this cannot be done using "slow" software simulations). Activity counters do not change the multiplier mathematical behavior. Moreover insertion of activity counters allowed us to optimise some multipliers. For example in multiplier based on Montgomery algorithm we were able to reduce the total number of registers used. Initially we had put in this architecture some auxiliary registers and assumed that the synthesis tool will optimise the solution. However it was not the case. After optimisation done by hand, the Montgomery multiplier unit is much smaller than the one presented in Section 3.2.2.

Corresponding implementation results without and with activity counters are reported in Table 4.1. The table reports huge area overhead and about a $\div 3$ frequency decrease due to activity counters inserted. These overheads are very important, but they only appear during evaluation not in final circuit. FPGA emulation leads to activity evaluation running at more than 100 MHz (see Table 4.1) which would not be possible using software simulations.

Table 4.1.: FPGA implementation results of $GF(2^m)$ multipliers without (original operators) and with (monitored operators) activity counters.

Algorithms	*without* activity counters			*with* activity counters		
	area LUT	freq. MHz	clock cycles	area LUT	freq. MHz	clock cycles
Classical	3638	302	264	11383	133	264
Montgomery	2178	323	270	6100	121	270
Mastrovito	3760	297	75	5956	110	75

4.1.1. Security level verification, problem identification

For all experiments, random operands have been used with uniform and equiprobable distribution for all bits. We have performed numerous experiments (corresponding to hundreds of thousands clock cycles for each tested solution). The traces presented correspond to typical traces. We find that even though it occurred that overall traces shape dependency on operand random values is relatively small, using average trace is not possible since this may flatten the activity variations and mask information leakage. Thus we have been evaluating our modifications by running modified multipliers for several various sets of experimental data. Here we present representative traces.

All proposed hardware solutions are analysed for one of field sizes $m = 233$ recommended in ECC standards (similar results are obtained for other field extension sizes).

The first analysed unit is the one based on classical algorithm. In Chapter 3 we have proposed two classical multipliers, one with standard, optimised to irreducible polynomial reduction and the other utilising matrix reduction. After collecting and analysing activity traces of those two types of classic multipliers we have concluded that the reduction does not impact activity traces very much. In fact the activity traces for both solutions are very similar, almost alike. Thus we have decided to focus on only one classical multiplier, the one using standard, optimised to irreducible polynomial reduction. We presume that this unit architecture can be more advantageous for inserting countermeasures mainly due to the fact that the standard reduction requires few steps. Figure 4.4 (left) presents useful activity measurement results for a typical sequence of $GF(2^m)$ multiplications of random operands using classical algorithm.
One may observe that there is a high peak at the beginning of each multiplication. The peak occurs due to the initialisation phase and loading of new operands values. Figure 4.4 (right) presents an extract for a single representative multiplication (all random operands lead to the similar overall

shape). We have noticed that dependency of the shape of activity variation curves on input data is rather low.

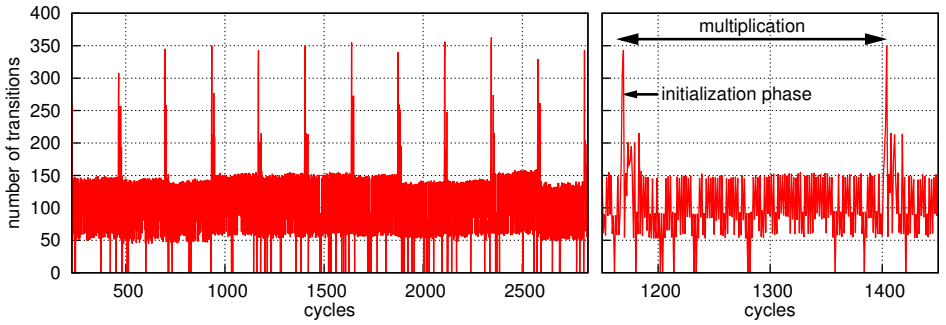


Figure 4.4.: Useful activity measurement results for random $GF(2^m)$ multiplications with classical algorithm (left). Extract for a single representative multiplication (right).

Measurement results for a sequence of random $GF(2^m)$ multiplications using Montgomery algorithm form are presented in Figure 4.5 (left) with an extract of a single representative multiplication (right). The reported measurements are shown for complete multiplications with final reduction (for conversion from Montgomery "representation"). We have provided comments on that point in Chapter 3. One can observe that there is a large activity drop at the end of each multiplication. We presume that it occurs due to the reduction step (recovery from Montgomery representation) and multiplier control.

Figure 4.6 (left) presents useful activity measurement results for a typical sequence of random $GF(2^m)$ multiplications using Mastrovito algorithm form with an extract for a single representative multiplication (right). The variations of the useful activity during a multiplication have a very specific decreasing "step-wave" shape.

Measurements for all three multiplication algorithms show very specific shapes for useful activity variations, which may lead to some information leakage. Those specific shapes provide the attacker with strong temporal

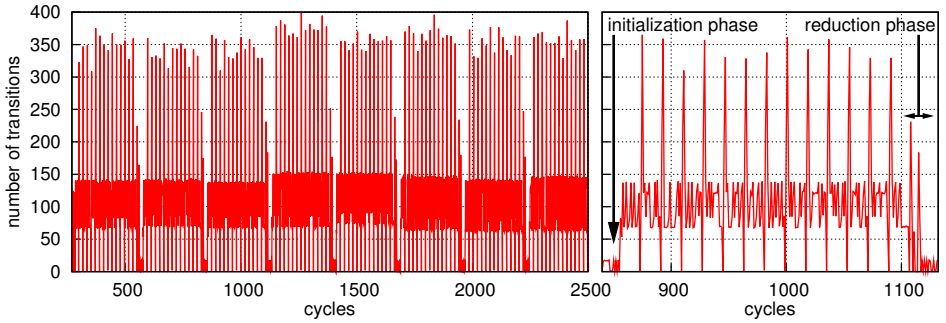


Figure 4.5.: Useful activity measurement results for random $GF(2^m)$ multiplications with Montgomery algorithm (left). Extract for a single representative multiplication (right).

references of the operations time location. Based on these references about field-level operations, higher-level operations (e.g. point addition and doubling) can be guessed.

The analysis of the obtained activity variation traces and the architectures of our solutions allowed us to came up with the following conclusions.

Peaks due to the initialisation phase at the beginning of operations in Figure 4.4 are not related to the selected algorithm but to the implemented architecture and especially its control. Resetting all internal registers generates a lot of activity and can give information about the time borders of the operation. Then this specific different shape for the initialisation phase may occur for other algorithms and architectures.

Activity drops at the end of operations in Figure 4.5 are due to low-complexity reduction step for the considered irreducible polynomial compared to multiplication iterations complexity. We reported measurements for complete multiplication (with final reduction) for fair comparison with other algorithms. In practice, those drops should not appear since reduction is only used at the end of a sequence of operations (with operands in Montgomery domain). However uniformising reduction step activity variations leads to uniformisation of activity variations of all finite field operations,

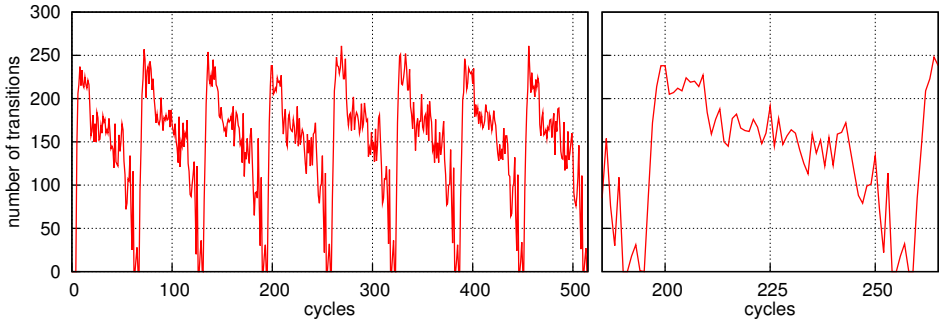


Figure 4.6.: Useful activity measurement results for random $GF(2^m)$ multiplications with Mastrovito algorithm (left). Extract for a single representative multiplication (right).

which is our goal.

The most problematic shape is the one for Mastrovito algorithm in Figure 4.6. The decreasing "step-wave" shape is due to variation of the computations quantity in the algorithm. In next section, we will present modifications of this multiplier at algorithmic and arithmetic levels to reduce information leakage.

4.1.2. Proposed countermeasures, circuit modifications

Analysing the obtained activity variations curves, we can define modification objectives. First, we have to suppress the peaks at the initialization phase. This is an architecture issue (i.e. modification of the operator control). All multiplication algorithms may benefit from this type of modification. Second, we have to take care of the activity drops during the reduction phase of Montgomery algorithm. But as stated in previous section, this phase is only used at the end of long sequence of operations in real ECC applications. Last, we have to make the "step-wave" shape of useful activity variations of Mastrovito algorithm less distinguishable. All this modifications aim at masking the multiplication operations. Aim at making impossible to localise the operation in time. Below, we describe our modifications for

each algorithm.

Classical two-step multiplication: The analysis shows that peaks at the beginning of each multiplication occur due to circuit initialisation. To suppress them, we have modified multiplier control and initialisation method. Initially to ensure the correct work of our circuit we have been resetting all registers before the start of computation. We have been resetting all registers also those not used at the beginning of multiplication process. Now we do not reset all registers in the first cycle but we have spread the reset activity over several cycles. We reintialise/reset register before it is used, if it is possible. What is more we have observed that not all registers need to be reinitialised. Thus we have skipped their initialisation/resetting.

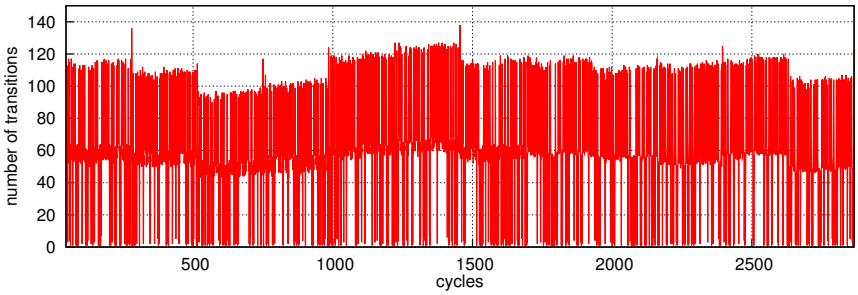


Figure 4.7.: Useful activity measurement results for random $GF(2^m)$ multiplications with modified classical algorithm.

Figure 4.7 shows useful activity measurements for a sequence of random multiplications using the modified multiplier. To reduce activity variations, we have also optimised the reduction step by reducing number of registers involved in reduction and merging all the steps of algorithm presented in Algorithm 7 into a chain of `XOR` operations. In the modified multiplier the average activity varies between 100 and 120 transitions (see Figure 4.7) while it was about 150 transitions in the original one (see Figure 4.4). Our modifications reduce the number of active registers in the operator thus they reduce also a little the power consumption of the operator. Comparing the

original operator's useful activity variations (Figure 4.4) with variations of modified multiplier (Figure 4.7), we can notice the absence of high initialisation peaks. For instance, between cycles 1500 and 2400 it is difficult to detect the executed operations boundaries.

Montgomery multiplication: If we do not consider the reduction step, we may say that the activity variations of Montgomery multiplier are more or less uniform (see Figure 4.5). The only thing, which may still give some information to the attacker is the initialisation phase. Activity drops at this phase occur due to a specific way, in which the input data are fetched. Like for classical algorithm, a modification of the initialisation control removes these drops. Figure 4.8 shows Montgomery activity variations with improved control (bottom curve) and without reduction (top curve).

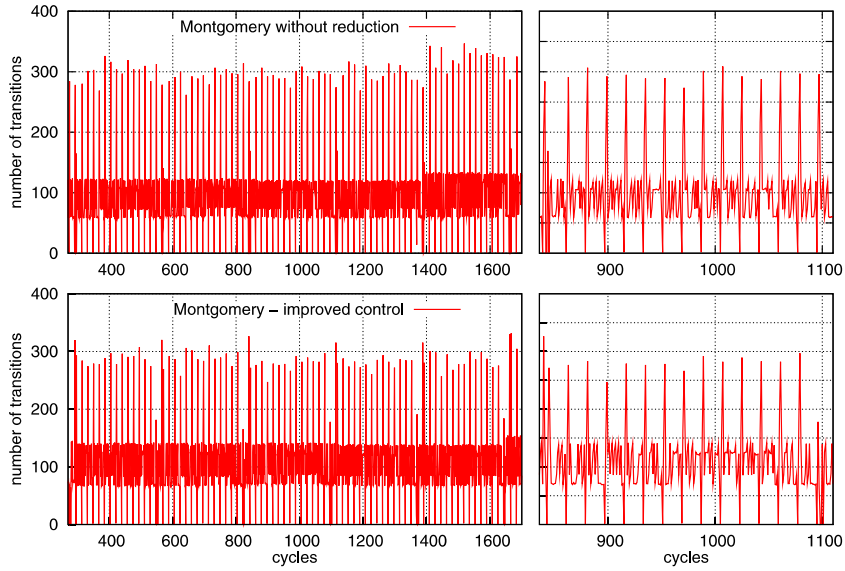


Figure 4.8.: Useful activity measurement results for random $GF(2^m)$ multiplications with Montgomery algorithm.

Mastrovito multiplication: The "step-wave" shape of useful activity variations of Mastrovito multiplier in Figure 4.6 is specific and can provide the attacker with a lot of information. Our objective is to modify the algorithm and the architecture in such a way that single multiplications cannot be too easily distinguished.

It is clear that the "step-wave" shape occurs mainly due to unequal number of registers switching in one clock cycle. Thus we have investigated two types of modifications for Mastrovito multiplier: "uniformisation" of the number of sub-multipliers' registers used in each clock cycle and "randomisation" of the starting times of the operator sub-multipliers. "Uniformisation" aims at making approximately the same number of bits switch in one clock cycle and "randomisation" at randomising number of bits switching in one clock cycle. We have derived many versions of those two types of modifications.

In order to explain modifications we have performed to the initial solution we recall some details of Mastrovito multiplier solution. Figure 4.9 presents the way we have divided matrix M into sub-matrices (see Section 3.2.2). The boxes with same indices Mi denote blocks, which can be multiplied by parts of vector b, using the same sub-multiplier unit.

It can be observed that some sub-multipliers are used more than the others. In initial solution we utilise one instance of each sub-multiplier (compare for example occurence of $M0$ and $M3$ on Figure 4.9), thus if we start them all at the same time, the activity is higher at the beginning of the operation (where all sub-multipliers are used) and lower at the end (almost all sub-multipliers are already switched off).

Our first proposition for "uniformisation" is to make the utilisation, in one clock cycle, of the number of sub-multipliers more uniform. We have tried not to change the total computation time of original multiplier, i.e. the number of states of FSM controlling the sub-multipliers work. The best obtained for this type of modification activity variation curve is shown on Figure 4.10, see curve V1.

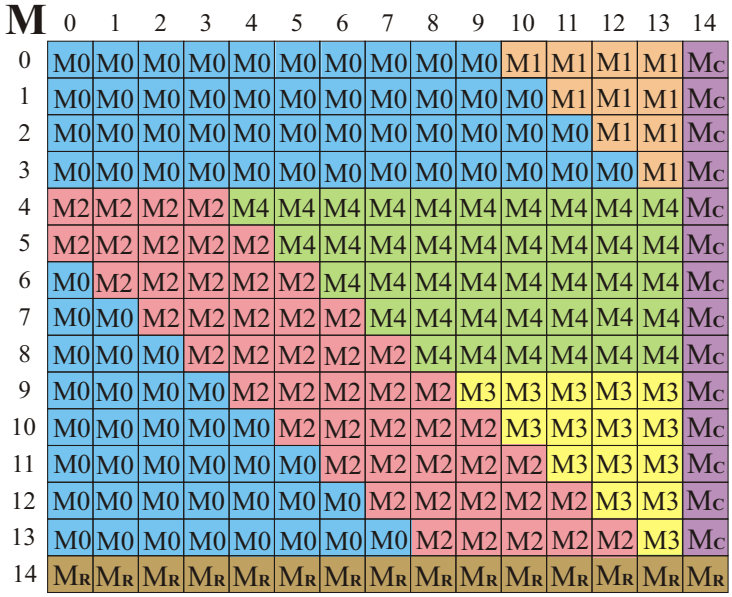


Figure 4.9.: Illustration of Mastrovito matrix partitioning for $m = 233$

Further, due to the fact that sub-multipliers use different number of different size registers, we have tried to uniform number of registers used in one clock cycle. Finally we have tried to consider not only number of registers used but also their sizes, thus the number of bits which possibly switch in each clock cycle. We have performed various attempts and tests. Then due to many dependencies between data we have decided to change number of states of control FSM and increase number of sub-multipliers used. The results obtained were very promising however we have found another problem. The other problem was the control of the circuit. We find that it causes drops to 0 in certain points. We have tried to modify the control algorithm and the idle, transient state (moment between the end of computation and the start of the new one) of the multiplier in order to avoid those sudden drops to zero, which may give information about operation time location. After unification of number of bits switching in one clock cycle and modification of circuit control we have obtained activity variation

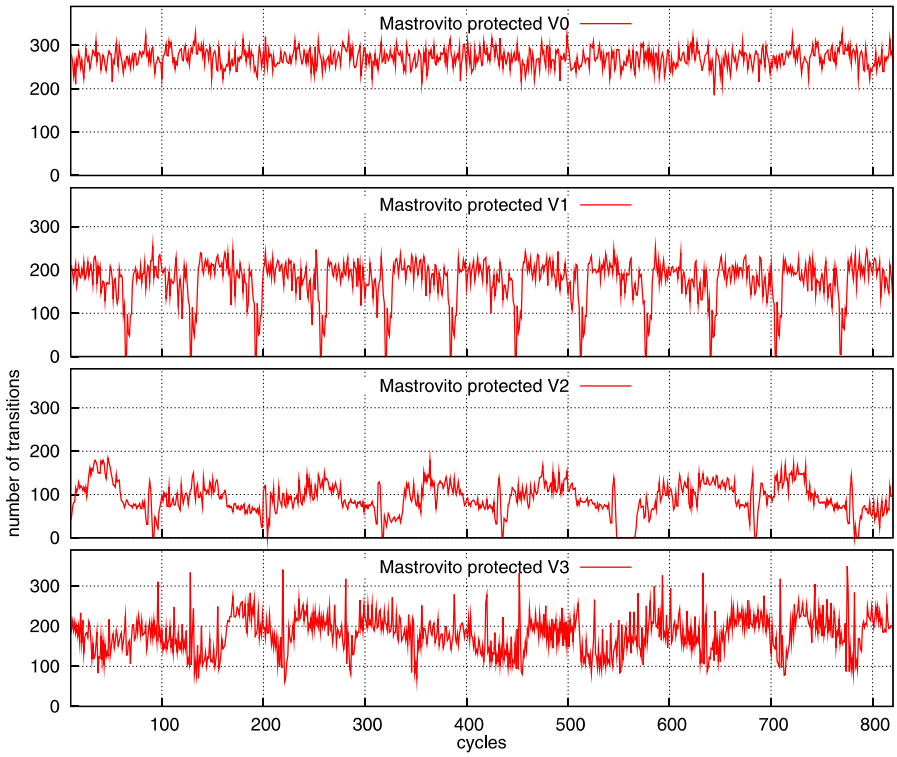


Figure 4.10.: Useful activity measurement results for random $GF(2^m)$ multiplications with 4 versions of modified Mastrovito algorithm.

curve V0, see Figure 4.10.

Our next objective was to randomise the starting moment of each sub-multiplier. This should "spread more" the activity over the whole computation. In order to randomise the beginning of sub-multiplications, we have used 8-bit LFSR (Mastrovito V2) and pseudo-random start sequence generator based on 4-bit LFSR (Mastrovito V3), which initialisation values depend on some bits of a and b operands. In order to avoid blocking the multiplier we exchange the initialisation values (seed) many times at random moments throughout multiplication operation. We have also tried

other methods but the best, according to us, results so far were achieved with use of our random start sequence generator based on 4-bit LFSRs, see Figure 4.11.

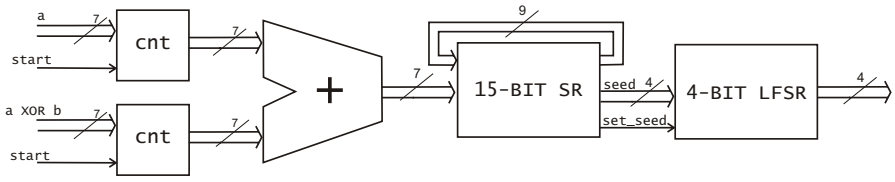


Figure 4.11.: Random start sequence generator based on 4-bit LFSR.

Due to the randomisation, the time needed to perform the complete multiplication, depending on which sub-multiplier is started first, will either decrease or increase randomly. The average number of clock cycles for Mastrovito V2 is 116 (minimal value: 98, maximal value: 126), whereas for Mastrovito V3 average number of clock cycles needed is 80 (minimal value: 64, maximal value: 108). Useful activity measurements for V2 (middle curve) and V3 (bottom curve) modifications are presented in Figure 4.10. As one can observe on Figure 4.10, the shapes of useful activity variations are more irregular and not easily predictable compared to the curve for the initial version in Figure 4.6.

The presented analysis and modifications aimed at masking the characteristic shapes of finite field multipliers activity variations curve. Additionally we have investigated the dependency on values of operands on the activity variations shape. In Figure 4.12 we present how the change of 1-bit and 16-bit in both operands affects the activity variation curves. As it can be observed it is difficult to predict where the curve changes and the variations of shapes are very low.

Implementation results for the modified multipliers: All modified multiplication algorithms have been implemented in FPGA. The corresponding results are reported in Table 4.2. Three optimisation targets were used

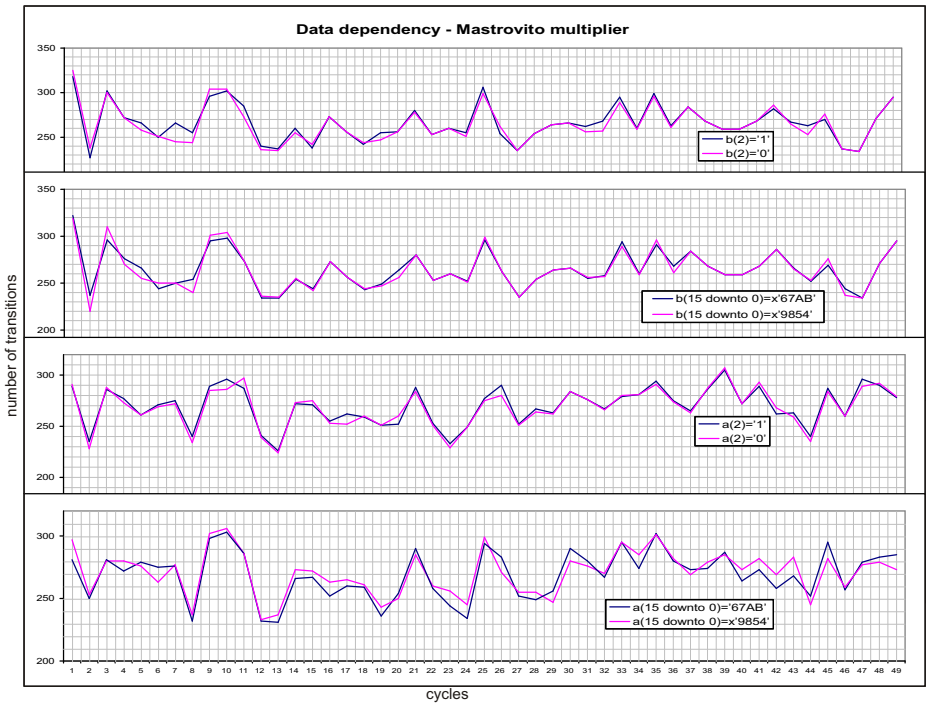


Figure 4.12.: Data dependency on activity variations curves for Mastrovito multiplier

for the synthesis tool: balanced area/speed, area and speed optimisations. In order to compare the modified multipliers to the original ones (see Table 4.1), we report a comparison factor α such as $\texttt{modified} = \alpha \times \texttt{original}$ both for area and frequency.

Evaluation of activity variation reduction To evaluate our modifications we have used signal processing tools. To do this the measured activity traces were transformed from time domain to frequency domain using Fast Fourier Transform (FFT), see [87]. Figure 4.13 presents those results for unprotected and protected versions of some of our multipliers. It represents the mathematical power for each frequency bin. The same logarithmic scale is used for all versions. One can observe an important reduction in the

Table 4.2.: Implementation results of $GF(2^m)$ multipliers with reduced activity variations.

Algorithms	balanced		area		speed		#
	area LUT	freq. MHz	area LUT	freq. MHz	area LUT	freq. MHz	clock cycles
Classical	2868	270	2778	228	3444	420	260
$\times\alpha$ factor	$\times$0.79	$\times$0.89	$\times$0.76	$\times$0.75	$\times$0.95	$\times$1.39	$\times$0.98
Montgomery	2099	323	2093	338	2099	423	264
$\times\alpha$ factor	$\times$0.96	$\times$1.00	$\times$0.96	$\times$1.05	$\times$0.96	$\times$1.31	$\times$0.98
Mastrovito v0	3889	225	3894	197	3933	308	48
$\times\alpha$ factor	$\times$1.04	$\times$0.75	$\times$0.97	$\times$0.66	$\times$1.05	$\times$1.04	$\times$0.64
Mastrovito v1	3463	414	3439	343	3489	384	75
$\times\alpha$ factor	$\times$1.09	$\times$1.39	$\times$1.09	$\times$1.15	$\times$0.93	$\times$1.29	$\times$1.00
Mastrovito v2	3700	306	3667	253	3717	388	avg.116
$\times\alpha$ factor	$\times$1.02	$\times$1.03	$\times$0.98	$\times$0.85	$\times$0.99	$\times$1.3	$\times$1.55
Mastrovito v3	3903	319	3837	250	4335	375	avg.80
$\times\alpha$ factor	$\times$1.03	$\times$1.07	$\times$1.02	$\times$0.84	$\times$1.15	$\times$1.26	$\times$1.07

potential information leakage for all frequencies.

In order to numerically compare solutions, we have computed the spectral flatness measure (SFM) [87]:

$$\text{SFM} = \frac{\sqrt[n]{\prod_{i=1}^{n} p(i)}}{\frac{1}{n}\sum_{i=1}^{n} p(i)} \qquad \in [0,1]$$

SFM is the ratio of the geometric mean to the arithmetic mean for a collection of n frequency bins $p(i)$ (power for frequency bin i). A SFM close to 1 indicates a spectrum with power well distributed in all frequency bins (flat curve) while a SFM close to 0 indicates that power is concentrated into a few bins (curve with peaks). SFM values are reported on Figure 4.13. Improvement is limited for classic algorithm, but for Mastrovito, our modifications lead to significant improvement (from 0.31 for unprotected version to 0.58 for the best protected version). The obtained results are rather satisfying.

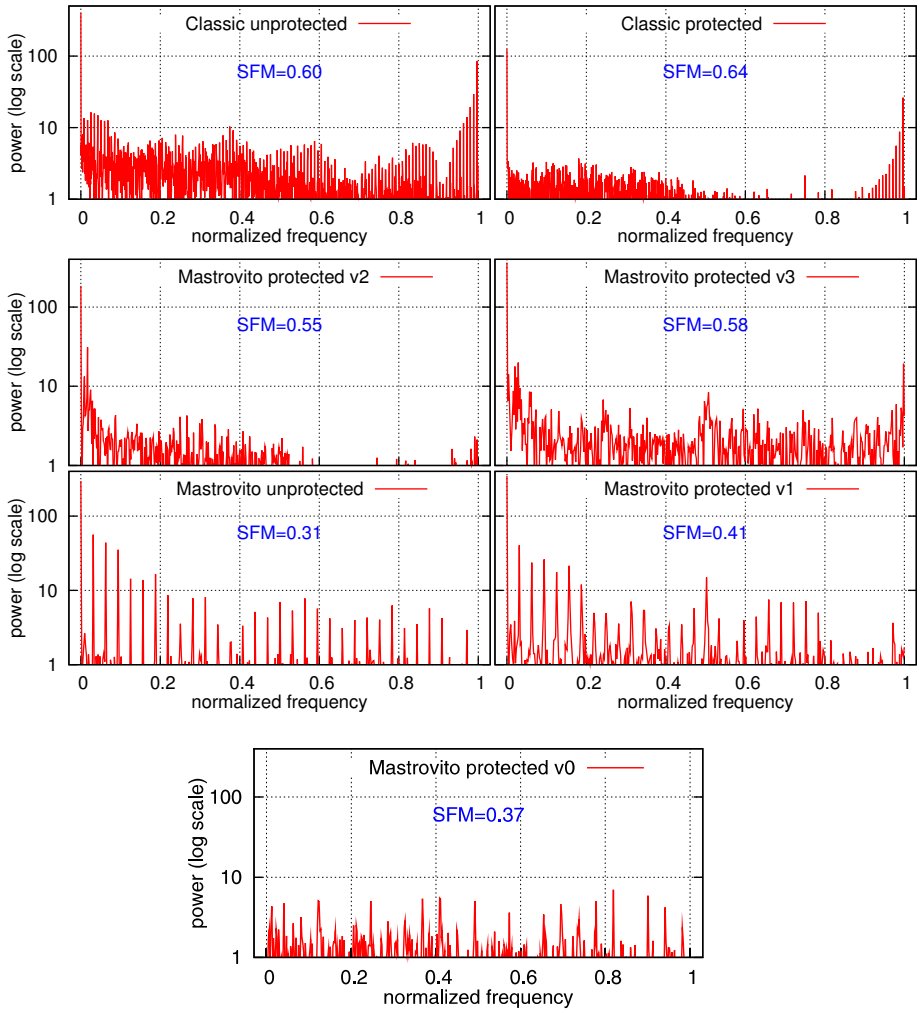


Figure 4.13.: FFT analysis results for unprotected and protected versions of multipliers (top: classic algorithm, middle and bottom: Mastrovito algorithm for various versions).

We can see that there is a way to reduce information leakage.

Lastly we have tried to implement point doubling operation, using Lopez-Dahab projective coordinates for elliptic curve point representation, (for al-

gorithm see [36] section 3.2.3) with our basic finite-field operators to see if it is easy to distinguish the sequence of operations performed. Figure 4.14 presents activity traces for calculation of double of elliptic curve point ($2P$) performed by our protected and unprotected Mastrovito multiplier. Looking at the activity traces of unprotected multiplier (upper trace) it is easy to notice each multiplication performed. This information may allow distinguishing between point addition ($P + Q$) operation and doubling ($2P$) operation, thus recovering the sequence of those operations and eventually retrieving the secret key.

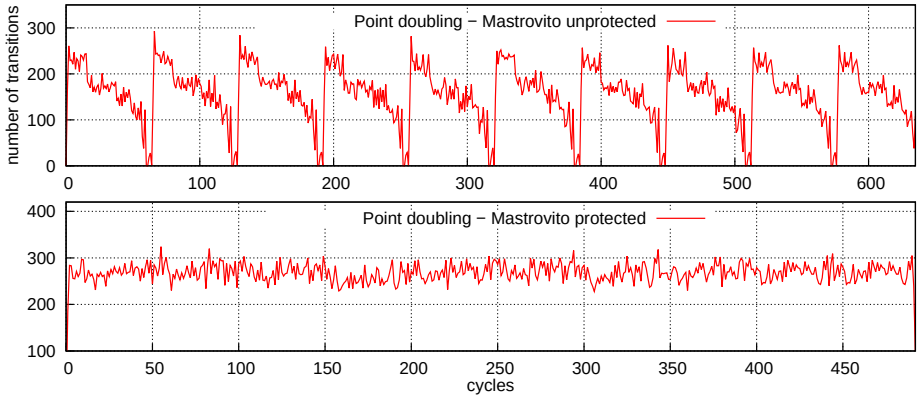


Figure 4.14.: Useful activity measurement results for $2P$ operation for unprotected (top figure) and protected (bottom figure) $GF(2^m)$ operators.

Comparison of obtained activity traces with current measurements In order to finally evaluate obtained protection results the instantaneous current consumed by the device performing finite-field operations was measured. We have measured current supplied to the Virtex-II Pro device mounted on SASEBO-G side-channel attack standard evaluation board which allows for taking such specific measurements with use of Lecroy Waverunner 104Xi-a oscilloscope and Tektronix CT1 current probe. The measurement station is controlled by HP Z800 computing server. To ease the measure-

ments the FPGA board is supplied from low noise HP E3610A power supply. Figure 4.15 shows comparison of multiplication activity traces obtained using activity counters and the ones obtained by current probe measurement.

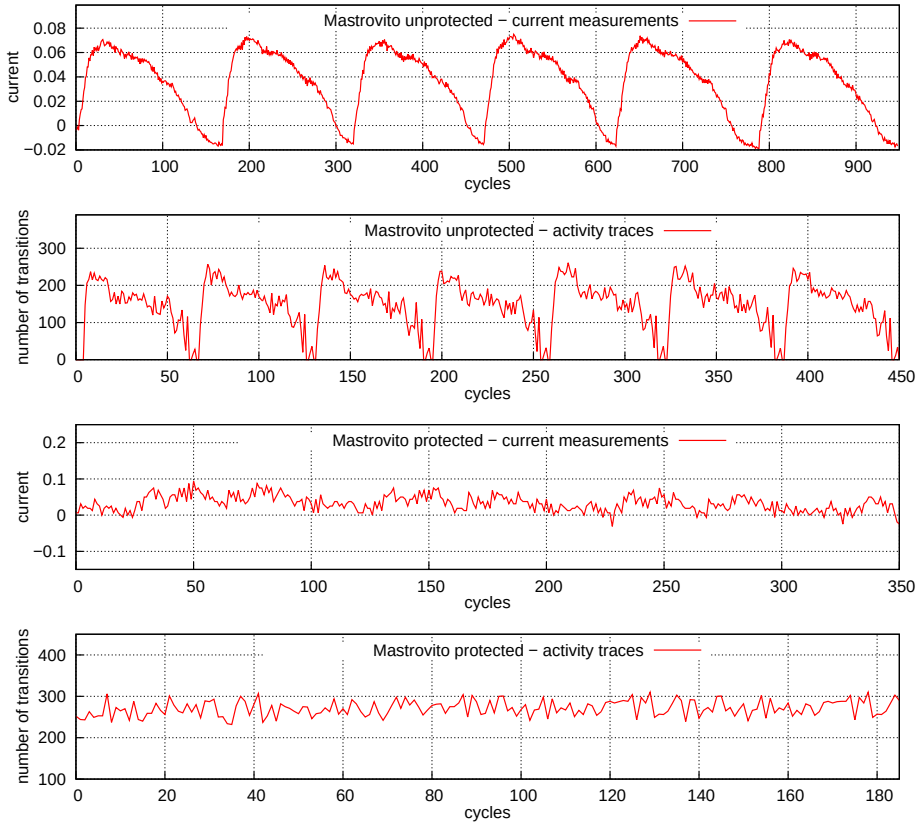


Figure 4.15.: Comparison of activity traces and current measurements for: Mastrovito multiplier unprotected version – 5 multiplications in a row and protected version (uniformised) – 3 multiplications in a row

Analysing measurements results we can see that the activity traces obtained by observation of internal switching are rather accurate. The evaluation of activity of multipliers done with activity monitors and ChipScope seems to reflect the real activity of multipliers. Figure 4.15 presents ac-

tivity traces for protected unprotected multiplier (top curves) and (bottom figures). It is easy to observe that our modifications yielded desired results. Observing traces obtained for protected multiplier it is hard to distinguish single multiplications.

4.1.3. Conclusions

Activity measurements analysis revealed that the implemented multiplication algorithms (classical, Montgomery and Mastrovito) lead to specific shapes for the curve of activity variations which may be used as a small source of information leakage for some side channel attacks.

We have proposed modifications of selected $GF(2^m)$ multipliers to reduce this information leakage source at two levels: architecture level by removing activity peaks due to control (e.g. reset at initialisation) and algorithmic level by modifying the shape of the activity variations curve. Due to optimisatons performed at a very low-level of a circut there is no significant area and delay overhead.

Analysing activity traces obtained for protected multipliers we may conclude that proposed modifications lead to masking the trace of multiplication operation.

5. Summary and Conclusions

In this work $GF(2^m)$ arithmetic operators dedicated to elliptic curve cryptography applications have been studied. Conducted researches aimed at providing efficient and secure against some side-channel power analysis attacks $\mathbf{GF(2^m)}$ ***hardware arithmetic operators*** which can be integrated in ECC processor.

The first goal was to provide efficient hardware arithmetic operators units. To do this we have performed vast research on existing algorithms and their improvements/optimisations/variations. The first requirement for the operators was that they should be dedicated to reconfigurable hardware. Thus during the analysis we have implemented many of described finite field arithmetic operations algorithms in order to notice their features which may be efficiently utilised in hardware (such as: decomposition schemes, computation order, internal coding, operands represenation, etc.). The second requirement was that they should serve ECC applications thus they should operate on 150–600 bit numbers (i.e. large numbers) and should be efficient in terms of speed, area and energy. The analysis allowed us to find and combine such features thus leading to design of our own $GF(2^m)$ hardware arithmetic operators based on known algorithms and dedicated to FPGA circuits (see Chapter 3).

Second goal was to evaluate the efficiency and overall cost of designed operators. The evaluation aimed at providing final speed and area efficient $GF(2^m)$ arithmetic operators solutions. The $GF(2^m)$ operators are vital part of cryptographic systems and their effectiveness strongly impacts the effectiveness of the whole system. The designed operators should work as a

part of ECC system thus in order to do not degrade its performance they should be small and fast. Comparing our hardware arithmetic operators to other solutions found in literature we may conclude that we have succeeded in providing efficient and low cost operators which will not degrade performance of the cryptographic system they will be part of (see Chapter 3).

The final goal of was to secure elaborated $GF(2^m)$ arithmetic units against some of popular passive attacks: side-channel attacks. According to known sources no one yet attempted to secure the lowest level operations of elliptic curve cryptographic systems that is the finite-field operations. However there were developed many countermeasures against side-channel attacks for protecting curve- and protocol-level operations neglecting information leakage existing at the lowest level of ECC operations. We have chosen to secure the operators against some of power analysis attacks which by observation of power consumed by the device performing cryptographic operations aim at revealing the secret. However we presume that our countermeasures may be extended to countermeasures against electromagnetic attacks. Before providing protections we had to evaluate the security of previously elaborated efficient operators in order to identify information leakage sources. The operators security was evaluated in two manners: using FPGA emulation with activity counters on standard Xilinx FPGA board and monitoring of internal signals using Chipscope; and by measurement with use of very fast LeCroy WaveRunner 104Xi-A oscilloscope and high frequency Tektronix CT1 current probe of instantaneous current supplied to the FPGA performing finite field operations mounted on SASEBO-G side-channel attack standard evaluation board. The evaluation of security of the unprotected operators showed specific activity trace shapes which may be used as a small source of information leakage. The shapes may enable the attacker to localise operations time boundaries or identify the point doubling ($2P$) or point addition ($P + Q$) operation and thus to reveal the secret. To avoid information leakage in $GF(2^m)$ operators we have

performed their algorithmic and architectural modifications. In Chapter 4 we have presented and illustrated that inserted countermeasures diminish significantly observed sources of information leakage in the operators. Both methods of evaluation confirmed that after inserting the countermeasures it is difficult to distinguish specific shapes of operations and what is more to localise their time boundaries.

Summarising, as a result of conducted researches the following original results were obtained:

- efficient in terms of speed and area $GF(2^m)$ **hardware arithmetic operators** dedicated to ECC applications were proposed;
- **successful protections** against some power analysis side channel attacks for $GF(2^m)$ hardware arithmetic operators were developed;
- the **tradeoff** between efficiency and security of $GF(2^m)$ hardware arithmetic operators was found.

Concluding it can be claimed that we have succeeded in providing speed, area and energy efficient $GF(2^m)$ hardware arithmetic operators dedicated to FPGA circuits and ECC applications. We have detected sources of information leakage in the operators and have modified the operators to reduce the information leakage. Thus we claim that **it is possible to create not only efficient but also secure against some side channel power analysis attacks elaborated** $GF(2^m)$ **arithmetic operators**.

References

[1] ISO/IEC 15946-2: Information technology – Security techniques – Cryptographic techniques based on elliptic curves – Part 1: Digital Signatures, 2002.

[2] ANSI X9.62:2005 Public Key Cryptography for the Financial Services Industry, The Elliptic Curve Digital Signature Algorithm (ECDSA), 2005.

[3] Draft Standard for Specifications for Password based Public Key Cryptographic Techniques (Revision of IEEE 1363-2000), 2007.

[4] R. J. Anderson and M. G. Kuhn. Tamper resistance: a cautionary note. In *Proc. 2nd USENIX Workshop on Electronic Commerce - Volume 2*, WOEC'96, page 1, Berkeley, CA, USA, 1996. USENIX Association.

[5] R. J. Anderson and M. G. Kuhn. Low Cost Attacks on Tamper Resistant Devices. In *Proc. 5th International Workshop on Security Protocols*, pages 125–136, London, UK, 1998. Springer.

[6] S. Arora and B. Barak. *Computational Complexity - A Modern Approach*. Cambridge University Press, 2009.

[7] A. Barenghi, L. Breveglieri, I. Koren, and D. Naccache. Fault Injection Attacks on Cryptographic Devices: Theory, Practice, and Countermeasures. *Proceedings of the IEEE*, pages 1–21, 2012.

[8] L. Batina, S. B. Örs, B. Preneel, and J. Vandewalle. Hardware architectures for public key cryptography. *Integration, the VLSI Journal*, 34(1–2):1–64, May 2003.

[9] O. Billet and M. Joye. The Jacobi Model of an Elliptic Curve and

Side-Channel Analysis. In *AAECC*, pages 34–42, 2003.

[10] I. F. Blake, G. Seroussi, and N. P. Smart. *Elliptic curves in cryptography*. Cambridge University Press, New York, USA, 1999.

[11] E. Brier, Ch. Clavier, and F. Olivier. Correlation Power Analysis with a Leakage Model. *CHES 2004, LNCS 3156. Springer*, pages 135–152.

[12] E. Brier, I. Déchéne, and M. Joye. *Embedded Cryptographic Hardware: Methodologies & Architectures*, chapter Unified addition formulæ for elliptic curve cryptosystems. Nova Science Publishers, Inc., Commack, NY, USA, 2004.

[13] E. Brier and M. Joye. Weierstraß Elliptic Curves and Side-Channel Attacks. In *Public Key Cryptography*, pages 335–345, 2002.

[14] A. Byrne, N. Meloni, A. Tisserand, E. M. Popovici, and W. P. Marnane. Comparison of Simple Power Analysis Attack Resistant Algorithms for an Elliptic Curve Cryptosystem. *Journal of Computers*, 2(10):52–62, 2007.

[15] T. Chabrier, D. Pamula, and A. Tisserand. Hardware implementation of DBNS recoding for ECC processor. In *Proc. 44th Asilomar Conference on Signals, Systems and Computers*, pages 1129–1133, Pacific Grove, California, U.S.A., November 2010. IEEE.

[16] S. Chari, J. Rao, and P. Rohatgi. Template Attacks. *CHES 2002, LNCS 2523. Springer*, pages 51–62.

[17] B. Chevallier-Mames, M. Ciet, and M. Joye. Low-Cost Solutions for Preventing Simple Side-Channel Analysis: Side-Channel Atomicity. *IEEE Transactions on Computers*, 53(6):760–768, June 2004.

[18] Ch. W. Chiou, J.-M. Lin, Ch.-Y. Lee, and Ch.-T. Ma. Novel Mastrovito Multiplier over $GF(2^m)$ Using Trinomial. In *Proc. 5th International Conference on Genetic and Evolutionary Computing*, ICGEC '11, pages 237–242, Washington, DC, USA, 2011. IEEE Computer Society.

[19] J.-S. Coron. Resistance against Differential Power Analysis for Elliptic Curve Cryptosystems. *CHES 1999, LNCS 1717. Springer*, pages 292–

302.

[20] Xilinx Corporation. Virtex-6 family overview (product specification), 2012.

[21] F. Crowe, A. Daly, and W. Marnane. A scalable dual mode arithmetic unit for public key cryptosystems. In *Information Technology: Coding and Computing (ITCC)*, volume 1, pages 568–573, April 2005.

[22] W. Diffie and M.E. Hellman. New Directions in Cryptography. *IEEE Transactions on Information Theory*, IT-22:644–654, 1976.

[23] T. Elgamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Transactions on Information Theory*, 31(4):469–472, July 1985.

[24] A. Enge. *Elliptic curves and their applications to cryptography: an introduction.* Kluwer Academic Publishers, Norwell, MA, USA, 1999.

[25] S. S. Erdem, T. Yanik, and C. K. Koc. Polynomial Basis Multiplication over GF(2^m). *Acta Applicandae Mathematicae*, 93(1-3):33–55, September 2006.

[26] J. Fan, X. Guo, E. De Mulder, P. Schaumont, B. Preneel, and I. Verbauwhede. State-of-the-art of Secure ECC Implementations: A Survey on Known Side-channel Attacks and Countermeasures. In *HOST*, pages 76–87, 2010.

[27] E. Ferrer, D. Bollman, and O. Moreno. A fast finite field multiplier. In *Proc. 3rd International Conference on Reconfigurable Computing: Architectures, Tools and Applications*, ARC'07, pages 238–246. Springer, 2007.

[28] W. Fischer, Ch. Giraud, E.W. Knudsen, and J.-P. Seifert. Parallel scalar multiplication on general elliptic curves over $\mathrm{F_p}$ hedged against Non-Differential Side-Channel Attacks. *IACR Cryptology ePrint Archive*, 2002:7, 2002.

[29] P.-A. Fouque, D. Réal, F. Valette, and M. Drissi. The Carry Leakage on the Randomized Exponent Countermeasure. *CHES 2008, LNCS 5154. Springer*, pages 198–213.

[30] A. P. Fournaris and O. Koufopavlou. Applying systolic multiplication-inversion architectures based on modified extended Euclidean algorithm for $GF(2^k)$ in elliptic curve cryptography. *Computers & Electrical Engineering, Elsevier*, 33(5-6):333–348, September 2007.

[31] A. Fraboulet, T. Risset, and A. Scherrer. Cycle Accurate Simulation Model Generation for SoC Prototyping. In *SAMOS*, pages 453–462, 2004.

[32] P. Gallagher. FIPS PUB 186-3 Federal Information Processing Standards Publication Digital Signature Standard (DSS), 2009.

[33] L. Goubin. A Refined Power-Analysis Attack on Elliptic Curve Cryptosystems. *PKC 2003, LNCS 2567. Springer*, pages 199–211.

[34] C. Grabbe, M. Bednara, J. Teich, J. von zur Gathen, and J. Shokrollahi. FPGA designs of parallel high performance $GF(2^{233})$ multipliers [cryptographic applications]. In *Proc. International Symposium on Circuits and Systems (ISCAS)*, volume 2, pages 268–271, May 2003.

[35] X. Guo, J. Fan, P. Schaumont, and I. Verbauwhede. Programmable and Parallel ECC Coprocessor Architecture: Tradeoffs between Area, Speed and Security. *CHES 2009, LNCS 5747. Springer*, pages 289–303.

[36] D. Hankerson, A. Menezes, and S. Vanstone. *Guide to Elliptic Curve Cryptography*. Springer, 2004.

[37] S. Hauck and A. DeHon. *Reconfigurable Computing: The Theory and Practice of FPGA-Based Computation*. Morgan Kaufmann, November 2007.

[38] T. Izu and T. Takagi. A Fast Parallel Elliptic Curve Multiplication Resistant against Side Channel Attacks. In *Public Key Cryptography*, pages 280–296, 2002.

[39] M. Joye. *Advances in Elliptic Curve Cryptography*, volume 317 of *London Mathematical Society Lecture Note Series*, chapter Defenses Against Side-Channel Analysis, pages 87–100. Cambridge University Press, April 2005.

[40] M. Joye and J.-J. Quisquater. Hessian Elliptic Curves and Side-Channel Attacks. *CHES 2001, LNCS 2162. Springer*, pages 402–410.

[41] M. Joye and Ch. Tymen. Protections against Differential Analysis for Elliptic Curve Cryptography. *CHES 2001, LNCS 2162. Springer*, pages 377–390.

[42] J. T. Santini JR., M. J. Cima, and R. S. Langer. Microchip drug delivery devices. Patent, 07 2006. US 7070590.

[43] D. Kahn. *The Codebreakers: The Comprehensive History of Secret Communication from Ancient Times to the Internet.* Scribner, December 1996.

[44] A. Karatsuba and Y. Ofman. Multiplication of Multi-Digit Numbers on Automata (in Russian). *Doklady Akad. Nauk SSSR*, 145(2):293–294, 1962. Translation in Soviet Physics-Doklady, 44(7), 1963, p. 595-596.

[45] R. Katti and J. Brennan. Low complexity multiplication in a finite field using ring representation. *IEEE Transactions on Computers*, 52(4):418–427, April 2003.

[46] S. Kilts. *Advanced FPGA Design: Architecture, Implementation, and Optimization.* Wiley-IEEE Press, 2007.

[47] Ch.H. Kim, S. Kwon, J.J. Kim, and Ch.P. Hong. A New Arithmetic Unit in $GF(2^m)$ for Reconfigurable Hardware Implementation. In *FPL*, pages 670–680, 2003.

[48] N. Koblitz. *A course in number theory and cryptography.* Springer, New York, USA, 1987.

[49] N. Koblitz. Elliptic Curve Cryptosystems. *Mathematics of Computation*, 48(177):203–209, 1987.

[50] C. K. Koc and T. Acar. Montgomery Multiplication in GF(2^k). *Designs, Codes and Cryptography*, 14(1):57–69, April 1998.

[51] P. C. Kocher. Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. In *CRYPTO*, pages 104–113, 1996.

[52] P. C. Kocher, J. Jaffe, and B. Jun. Differential Power Analysis. In

CRYPTO, pages 388–397, 1999.

[53] F. Koeune and F.-X. Standaert. A Tutorial on Physical Security and Side-Channel Attacks. In *FOSAD*, pages 78–108, 2004.

[54] RSA Laboratories. PKCS #1 v2.1: RSA Cryptography Standard, 2002.

[55] S. Lang. *Elliptic Curves : Diophantine Analysis*. Springer, 1978.

[56] K. Lauter. The advantages of elliptic curve cryptography for wireless security . *Wireless Communications, IEEE*, 11(1):62–67, 2004.

[57] H. Li, J. Huang, P. Sweany, and D. Huang. FPGA implementations of elliptic curve cryptography and Tate pairing over a binary field. *Journal of Systems Architecture, Elsevier*, 54(12):1077–1088, December 2008.

[58] R. Lidl and H. Niederreiter. *Finite fields*. Cambridge University Press, 1983.

[59] R. Lidl and H. Niederreiter. *Introduction to Finite Fields and Their Applications*. Cambridge University Press, 2nd edition, 1994.

[60] S. Mangard. Hardware Countermeasures against DPA - A Statistical Analysis of Their Effectiveness. In *Topics in Cryptology – CT-RSA*, pages 222–235, 2004.

[61] S. Mangard, E. Oswald, and T. Popp. *Power Analysis Attacks: Revealing the Secrets of Smart Cards*. Springer, 2007.

[62] E. Mastrovito. *VLSI Architectures for Computation in Galois Fields*. PhD thesis, Department of Electrical Engineering, Linkoping University, Sweden, 1991.

[63] E. McCallister, T. Grance, and K. Scarfone. Guide to Protecting the Confidentiality of Personally Identifiable Information (PII). NIST Special publication 800-122, 2010.

[64] R.J. McEliece. *Finite field for scientists and engineers*. Kluwer Academic Publishers, 1987.

[65] A. Menezes. *Elliptic Curve Public Key Cryptosystems*, volume 234 of *The Springer International Series in Engineering and Computer*

Science. Springer, 1993.

[66] A. Menezes, S. Vanstone, and P.C. Van Oorschot. *Handbook of Applied Cryptography*. CRC Press, Inc., Boca Raton, FL, USA, 1st edition, 1996.

[67] R. C. Merkle. Secure communications over insecure channels. *Commun. ACM*, 21(4):294–299, April 1978.

[68] S. Micali and L. Reyzin. Physically observable cryptography. *Theory of Cryptography, First Theory of Cryptography Conference (TCC), LNCS 2951. Springer*, pages 278–296, 2004.

[69] V. S. Miller. Use of elliptic curves in cryptography. *CRYPTO 1985, LNCS 218. Springer*, pages 417–426.

[70] B. Möller. Securing Elliptic Curve Point Multiplication against Side-Channel Attacks. In *Proc. 4th International Conference on Information Security (ISC)*, pages 324–334. Springer, 2001.

[71] P. L. Montgomery. Modular Multiplication Without Trial Division. *Mathematics of Computation*, 44(170):519–521, April 1985.

[72] S. Moon, J. Park, and Y. Lee. Fast VLSI arithmetic algorithms for high-security elliptic curve cryptographic applications. *IEEE Transactions on Consumer Electronics*, 47(3):700–708, August 2001.

[73] R. C. Mullin, I. M. Onyszchuk, S. Vanstone, and R. M. Wilson. Optimal normal bases in $GF(p^n)$. *Discrete Appl. Math.*, 22(2):149–161, February 1989.

[74] S. Murugiah and K. Scarfone. Guidelines for Securing Wireless Local Area Networks (WLANs). NIST Special publication 800-153, 2012.

[75] A. H. Namin, W. Huapeng, and M. Ahmadi. Comb Architectures for Finite Field Multiplication in $F(2^m)$. *IEEE Transactions on Computers*, 56(7):909–916, July 2007.

[76] R. Newell. Design and Data Security in PLDs: When Security is Non-Negotiable. Security Webinar, Microsemi corporation, 2012.

[77] NSA. TEMPEST series. http://cryptome.org/nsa-tempest.htm.

[78] T. Okamoto, E. Fujisaki, and H. Morita. PSEC: Provably Secure

Elliptic Curve Encryption Scheme. In *IEEE P1363a*, 2000.

[79] K. Okeya and K. Sakurai. Power Analysis Breaks Elliptic Curve Cryptosystems Even Secure against the Timing Attack. *Progress in Cryptology - INDOCRYPT 2000, LNCS 1977. Springer*, pages 217–314.

[80] E. Oswald. *Advances in Elliptic Curve Cryptography*, volume 317 of *London Mathematical Society Lecture Note Series*, chapter Side Channel Analysis, pages 69–86. Cambridge University Press, April 2005.

[81] D. Pamula, E. Hrynkiewicz, and A. Tisserand. Multiplication in $GF(2^m)$: area and time dependency/efficiency/complexity analysis. In *Proceedings of 10th International IFAC Workshop on Programmable Devices and Embedded Systems (PDES)* , pages 43–48, 2010.

[82] D. Pamula, E. Hrynkiewicz, and A. Tisserand. Analiza algorytmów mnozenia w ciele $GF(2^m)$. *Pomiary Automatyka Kontrola (PAK) (Measurement, Automation and Monitoring)* , 57(01/2011):58–60, 2011.

[83] D. Pamula, E. Hrynkiewicz, and A. Tisserand. Analysis of $GF(2^m)$ multipliers regarding Elliptic Curve Cryptosystem applications. In *Proceedings of 11th IFAC/IEEE International Conference on Programmable Devices and Embedded Systems (PDES)* , pages 252–257, 2012.

[84] D. Pamula and A. Tisserand. $GF(2^m)$ Finite-Field Multipliers with Reduced Activity Variations. *WAIFI 2012, LNCS 7369. Springer*, pages 152–167.

[85] B. Pochopień. *Arytmetyka systemów cyfrowych.* Skrypty Uczelniane - Politechnika Śląska. Wydaw. Politechniki Śląskiej, 2002.

[86] T. Popp, S. Mangard, and E. Oswald. Power Analysis Attacks and Countermeasures. *IEEE Design & Test of Computers*, 24(6), 2007.

[87] J. G. Proakis and D. G. Manolakis. *Digital Signal Processing.* Prentice Hall, 1996.

[88] J.-J. Quisquater. Side channel attacks, State-of-the-art. CRYPTREC

Report, 2002.

[89] J.-J. Quisquater and D. Samyde. A new tool for non-intrusive analysis of smart cards based on electro-magnetic emissions, the SEMA and DEMA methods. Presented at the rump session of EUROCRYPT'2000, 2000.

[90] S. Radack. Guide to protecting personally identifiable information. ITL Bulletin for April 2010.

[91] S. M. Radack. Security Metrics: Measurements to Support the Continued Development of Information Security Technology. ITL Bulletin, January 2010.

[92] Certicom Research. Standards for efficient cryptography, SEC 1: Elliptic curve cryptography, 2009. Version 2.0.

[93] Certicom Research. SEC 2: Recommended Elliptic Curve Domain Parameters, 2010. Version 2.0.

[94] Research Center for Information Security National Institute of Advanced Industrial Science and Technology. Side-channel Attack Standard Evaluation Board. SASEBO-G. Specification. Version 1.0 , 2008.

[95] F. Rodríguez-Henríquez, N. A. Saqib, and A. Díaz-Pérez. A fast parallel implementation of elliptic curve point multiplication over GF(2^{m}). *Microprocessors and Microsystems*, 28(5-6):329–339, 2004.

[96] F. Rodriguez-Henriquez, N. A. Saqib, A. Diaz-Perez, and C. K. Koc. *Cryptographic Algorithms on Reconfigurable Hardware*. Signals and Communication Technology. Springer, 2007.

[97] E. Savas and C. K. Koc. Finite Field Arithmetic for Cryptography. *IEEE Circuits and Systems Magazine*, 10(2):40–56, May 2010.

[98] P. Schaumont and I. Verbauwhede. A reconfiguration hierarchy for elliptic curve cryptography. In *Proc. 35th Asilomar Conference on Signals, Systems and Computers*, pages 449–453, November 2001.

[99] B. Schneier. *Applied cryptography (2nd ed.): protocols, algorithms, and source code in C.* John Wiley & Sons, Inc., New York, USA, 1995.

[100] B. Schneier. A self-study course in block-cipher cryptanalysis. *Cryptologia*, 24(1):18–33, January 2000.

[101] R. Shipsey. ECIES. Project NESSIE report NES/DOC/RHU/WP3/007/c, 2001.

[102] J. H. Silverman. *The Arithmetic of Elliptic Curves*. Number 106 in Graduate Texts in Mathematics. Springer, 1986.

[103] S.P. Skorobogatov. Side-channel attacks: new directions and horizons. In *ECRYPT2 School on Design and Security of Cryptographic Algorithms and Devices*, Albena near Varna, Bulgaria, May 2011.

[104] N. P. Smart and P.-Y. Liardet. Preventing SPA/DPA in ECC systems using the Jacobi Form. *CHES 2001, LNCS 2162. Springer*, pages 391–401, May 2001.

[105] Ch. Swenson. *Modern cryptanalysis - techniques for advanced code breaking*. Wiley, 2008.

[106] A. Tisserand. Low-power arithmetic operators. In C. Piguet, editor, *Low Power Electronics Design*, chapter 9. CRC Press, November 2004.

[107] A. Tisserand. Fast and Accurate Activity Evaluation in Multipliers. In *Proc. 42nd Asilomar Conference on Signals, Systems and Computers*, pages 757–761, Pacific Grove, California, U.S.A., October 2008. IEEE.

[108] H. C. A. van Tilborg. *Encyclopedia of Cryptography and Security*. Springer New York, Inc., Secaucus, NJ, USA, 2005.

[109] S. Vanstone. ECC holds key to next generation cryptography. *[Online].Available: http://www.design-reuse.com/articles/7409/ecchold-key-to-next-gen-cryptography.html*, March 2006.

[110] J. Wang and A. Jiang. A high-speed dual field arithmetic unit and hardware implementation. In *ASIC, 2007. ASICON '07. 7th International Conference on*, pages 213–216, October 2007.

[111] E. Wenger and M. Hutter. Exploring the Design Space of Prime Field vs. Binary Field ECC-Hardware Implementations. In *NordSec*, pages 256–271, 2011.

[112] N.H.E. Weste and D. Harris. *CMOS VLSI Design: A Circuits and*

Systems Perspective. Addison Wesley, third edition, 2004.

[113] J. Wolkerstorfer. Dual-Field Arithmetic Unit for $GF(p)$ and $GF(2^m)$. *CHES 2002, LNCS 2523. Springer*, pages 500–514.

[114] H. Wu, M. A. Hasan, I. F. Blake, and S. Gao. Finite field multiplier using redundant representation. *IEEE Transactions on Computers*, 51(11):1306–1316, November 2002.

[115] Xilinx Corporation. Virtex-II Pro and Virtex-II Pro X Platform FPGAs: Complete Data Sheet (Product Specification), 2007.

[116] Xilinx Corporation. Spartan-3E FPGA Family: Data Sheet (Product Specification), 2009.