Secure Architectures for Pairing Based Public Key Cryptography

Weibo Pan

September 2013

A Thesis Submitted to the
National University of Ireland
in Fulfillment of the Requirements for
the Degree of
Doctor of Philosophy

Supervisor: Dr. Liam Marnane
Head of Department: Prof. Nabeel A. Riza

Department of Electrical and Electronic Engineering,
University College, Cork

ABSTRACT

Along with the growing demand for cryptosystems in systems ranging from large servers to mobile devices, suitable cryptogrophic protocols for use under certain constraints are becoming more and more important. Constraints such as calculation time, area, efficiency and security, must be considered by the designer.

Elliptic curves, since their introduction to public key cryptography in 1985 have challenged established public key and signature generation schemes such as RSA, offering more security per bit. Amongst Elliptic curve based systems, pairing based cryptographies are thoroughly researched and can be used in many public key protocols such as identity based schemes. For hardware implementions of pairing based protocols, all components which calculate operations over Elliptic curves can be considered. Designers of the pairing algorithms must choose calculation blocks and arrange the basic operations carefully so that the implementation can meet the constraints of time and hardware resource area. This thesis deals with different hardware architectures to accelerate the pairing based cryptosystems in the field of characteristic two. Using different top-level architectures the hardware efficiency of operations that run at different times is first considered in this thesis.

Security is another important aspect of pairing based cryptography to be considered in practically Side Channel Analysis (SCA) attacks. The naively implemented hardware accelerators for pairing based cryptographies can be vulnerable when taking the physical analysis attacks into consideration. This thesis considered the weaknesses in pairing based public key cryptography and addresses the particular calculations in the systems that are insecure.

In this case, countermeasures should be applied to protect the weak link of the implementation to improve and perfect the pairing based algorithms. Some important rules that the designers must obey to improve the security of the cryptosystems are proposed. According to these rules, three countermeasures that protect the pairing based cryptosystems against SCA attacks are applied. The implementations of the countermeasures are presented and their performances are investigated.

CONTENTS

1. *Introduction* . . . 7
   - 1.1 Motivation . . . 7
   - 1.2 Aim of thesis . . . 8
   - 1.3 Thesis outline . . . 10

2. *Background Theory* . . . 13
   - 2.1 Introduction . . . 13
   - 2.2 Cryptography . . . 13
   - 2.3 Private and Public Key Cryptography . . . 15
     - 2.3.1 Private Key Cryptography . . . 15
     - 2.3.2 Public Key Cryptography . . . 16
     - 2.3.3 Identity Based Encryption . . . 17
     - 2.3.4 Attribute Based Encryption . . . 19
     - 2.3.5 ABE structure . . . 19
   - 2.4 Mathematical Background . . . 21
     - 2.4.1 Groups . . . 21
     - 2.4.2 Finite Fields . . . 22
     - 2.4.3 Elliptic Curves over Finite Fields . . . 23
     - 2.4.4 Point Operations over Elliptic Curves . . . 24
     - 2.4.5 Elliptic Curve Discrete Logarithm Problem . . . 26
   - 2.5 Pairing Based cryptography . . . 27
     - 2.5.1 Tate pairing over supersingular curves . . . 29
     - 2.5.2 Implementing the Pairings . . . 31
     - 2.5.3 Bilinear Pairings applied in IBE . . . 31
     - 2.5.4 Bilinear Pairings applied in ABE . . . 34
     - 2.5.5 Security of Pairing based cryptosystem . . . 36
   - 2.6 Cryptanalysis . . . 38
     - 2.6.1 Side Channel Analysis Attacks . . . 38
     - 2.6.2 Hardware Power consumption . . . 39

2.6.3 Weakness in pairing based protocols to side channel attacks. . 41
2.6.4 Countermeasures against power analysis attacks 42
2.7 Hardware Accelerators 42
2.7.1 Field Programmable Gate Arrays 43
2.7.2 SASEBO-GII board 44
2.7.3 Efficiency Evaluation of Hardware Designs 45
2.8 Conclusions 47

3. Arithmetic Architectures over the Field $GF(2^m)$ 49
3.1 Introduction 49
3.2 Choice of Finite Fields 49
3.3 Choice of Basis 50
3.4 Addition over $GF(2^m)$ 51
3.5 Multiplication over $GF(2^m)$ 52
3.5.1 Bit parallel multiplier and Bit serial multiplier 54
3.5.2 Digit serial multiplier 55
3.5.3 Karatsuba Multiplier 59
3.6 Squaring over $GF(2^m)$ 64
3.7 Inversion/Division over $GF(2^m)$ 65
3.8 Dedicated Inversion and Division architecture over $GF(2^m)$ 67
3.9 Conclusions 71

4. Implementation of the Tate Pairing over extension field $GF(2^{4m})$ 73
4.1 Introduction 73
4.2 Architecture for computations over $GF(2^{4m})$ 73
4.2.1 $GF(2^{4m})$ Multiplication 75
4.2.2 $GF(2^{4m})$ Squaring 76
4.2.3 $GF(2^{4m})$ Inversion 78
4.2.4 $GF(2^{4m})$ Frobenius Map 81
4.3 Implementing the η_T algorithm for calculating Tate pairing 82
4.3.1 Top level architecture design of η_T algorithm 82
4.3.2 Reconfiguration of the multiplications in Bus type design . . . 86
4.3.3 Implementation results of Bus type top-level architecture . . . 89
4.3.4 Implementation results of mixed type top-level architecture . . 91
4.4 Analysis of implementation result 93
4.4.1 Time analysis 93
4.4.2 Area analysis 95

4.4.3 A*T product analysis 97
4.4.4 Comparison with earlier work 98
4.5 Conclusions 100

5. Side Channel Attacks against Implementations of Tate Pairing algorithms . 102
5.1 Introduction 102
5.2 Side Channel Analysis Attacks 102
5.2.1 Correlation Power Analysis Attacks 103
5.2.2 Relationship between intermediate variables and power consumption 104
5.2.3 CPA attack setup 105
5.3 Side-channel security analysis of the η_T Pairings 107
5.3.1 Weakness of η_T pairing based IBE scheme 107
5.3.2 Weakness in addition 109
5.3.3 Weakness in multiplication 110
5.4 CPA attack against the $GF(2^m)$ operations 111
5.4.1 CPA against addition: Condition 1) $P(\alpha, \beta)$ public, attacking $Q(x, y)$ 111
5.4.2 CPA against addition: Condition 2) $Q(x, y)$ public, attacking $P(\alpha, \beta)$ 115
5.4.3 CPA against digit-serial multiplier(DSM): Condition 1) $P(\alpha, \beta)$ public, attacking $Q(x, y)$ 115
5.4.4 CPA against digit-serial multiplier(DSM): Condition 2) $Q(x, y)$ public, attacking $P(\alpha, \beta)$ 122
5.4.5 CPA against Karatsuba multiplier 125
5.4.6 CPA against register inserted Karatsuba multiplier: Condition 1) $P(\alpha, \beta)$ public, attacking $Q(x, y)$ 125
5.4.7 CPA against register inserted Karatsuba multiplier: Condition 2) $Q(x, y)$ public, attacking $P(\alpha, \beta)$ 128
5.5 Conclusions 128

6. Countermeasures against CPA attacks 130
6.1 Introduction 130
6.2 Weakness in the η_T pairing 130
6.3 Countermeasures to protect the η_T pairing 131
6.3.1 Exploiting bilinearity to protect the $GF(2^m)$ Adder 132
6.3.2 Randomized Miller variable to protect the $GF(2^m)$ Adder . . 134

6.3.3 Using projective coordinates to protect the $GF(2^m)$ Adder . . 135
6.3.4 Cost comparison between countermeasures 137
6.4 Security discussion 138
6.4.1 Security of exploiting bilinearity method 138
6.4.2 Security of randomized Miller variable method 139
6.4.3 Security of using projective coordinates method 140
6.4.4 Security operations in common 140
6.5 Bus type top-level architecture implementation result of the countermeasures 141
6.6 Mixed Type Implementation results of the countermeasures 144
6.7 Analysis of implementation result of the proposed countermeasures . 145
6.7.1 Time analysis 146
6.7.2 Time ratio analysis 147
6.7.3 A*T product analysis 149
6.8 Conclusions 150

7. Conclusions 152
7.1 Contributions to the Field 152
7.2 Future work 154
7.3 Publications 156

Appendix 157

A. Schedule of "for" loop operations 159

B. Implementation Results of Tate pairing 162

C. Features of Virtex Families 166

D. Countermeasures Algorithms Using Projective Coordinates 167

E. Calculation Time of Countermeasures Against CPA Attack 169

References 173

1. INTRODUCTION

1.1 Motivation

Throughout the development of human civilization, people all over the world have developed their own methods to protect their information in communications. In modern society, cryptosystems use secret keys to transform plaintext into ciphertext [1]. Both the encryptor and the decryptor need these secret key system to communicate. However, once the key is exposed, this system can be solved easily. The study of trying to break an encryption system without previously knowing the secret key is called cryptanalysis. The idea of cryptanalysis has existed since cryptography was first developed. Throughout history, cryptanalysis and cryptography have spurred on the development of the other. It always happens that in a war or a commercial war, breaking encrypted information of an opponent can lead to decisive events or even victory. Advanced cryptanalysis technology forces people to develop newer and stronger methods for securely encrypting and transmitting information.

Cryptography was initially used to protect secret information by governments and the military. With the development of the modern society, the demand for information security comes from many fields, such as commerces, client information protection, or personal e-mails. Since secure cryptographic systems are required in many applications, this thesis analyses and implements a highly effecient and secure public-key cryptographic scheme.

In the 1970s, cryptography became a widespread tool for securing communications. The security of modern public key cryptography relies on some mathematical problems. These problems include integer factorization [2] and the discrete logarithm problem (DLP) [3]. Among modern cryptosystems, Identity Based Encryption (IBE) [4, 5] and Attribute Based Encryption (ABE) [6] based on pairings [4] are popular public key schemes.

Pairing based IBE and ABE systems can be implemented on hardware accelerators on FPGAs. Different implementations of the cryptosystem can meet the demand for information security in many devices ranging from large servers, to mobile devices, to smart cards. Flexible platforms provide a variety of solutions for different constraints for certain demands. For example, a large server will use more hardware in exchange for a faster operation speed, while a smart card can use less hardware and a restricted operation speed. In most previous pairing based algorithm implementations, timing efficiency were considered as the first priority [7, 8, 9, 10, 11]. However, such designs do not satisfy the various demands of different applications. Thus, this work not only focuses on the the calculation speed, but also considers the hardware efficiency of the designs. In this work, different schedules of arranging the calculation modules are used, similar to those investigated in [12]. In addition, examing the calculation loop in the pairing based algorithm allows the differences between operations that happen only once and those that iterate many times to be considered for the first time. Thus in this work, different top-level architectures are implemented and their effeciency is investigated.

However, people who take such cryptosystem as a method to protect their information would always ask: "Are these schemes secure?" Thus in this work the security aspect of the chosen cryptographic algorithm implementation will be examined and perfected.

"The greatest enemy is the greatest teacher." The improvement and perfection of a cryptogarphic algorithm will not happen without the input of cryptanalysis. To examine the security aspect from point of view of an attacker helps a designer overcome the defects existing in the original cryptosystem. Side channel information leaked from the operating hardware is often related to the secret information [13, 14, 15]. A cryptosystem designer should consider all the assumptions under which the attackers may find weaknesses in the proposed cryptosystem.

If there is any weakness identified in the cryptosystem, a designer should try to overcome such defects by applying countermeasures accordingly. Countermeasures must be carefully applied so that the including hard mathematical problem still exists while the weakness is perfectly masked. Although introducing countermeasures to the original algorithm can lead to the lowering of the efficiency of the design by increasing the operation time and hardware used, it still is necessary.

1.2 Aim of thesis

The primary aim of this work is to investigate and improve the secure implementation of a public key cryptographic processor. Elliptic curve cryptography (ECC) [3, 19] was introduced by Miller and Koblitz in 1985 [20, 21]. It is based on a mathematical entity known as an elliptic curve [22] and provides a good security level making use of only small hardware resources [23]. Pairing based cryptography (PBC) is used in new forms of public key cryptography known as IBE and ABE. Among PBCs, the Tate Pairings is proved to be the most efficient one [24, 25]. In this work, the performance and security aspects of the Tate pairing, more specifically the η_T algorithm [26] for calculating the Tate pairing, will be investigated.

There are two main options for implementing the Tate pairing: software on a general purpose processor and dedicated hardware. The general purpose processors are in general electronic devices whose functions are based on the software programmes implemented, such as computers, PDAs and cell phones. However, as these processors are not optimized for implementing cryptography, a software implementation on a general purpose processor may lead to low performance [27, 28, 29, 30], and moreover, security aspects of the secret information in the general purpose processors is not guaranteed. A dedicated hardware processor for the cryptosystems can provide much better performance, not only accelerating the operation time, but also promising better security and can be tamper proof to prevent attackers compromising the cryptosystem. Thus, dedicated hardware accelerators are widely used in implementing practical cryptosystems. In this work, the algorithms are implemented on Field Programmable Gate Arrays.

There are several parameters to show the efficiency of a hardware processor. Since dedicated hardware processors are initially developed to accelerate the operation speed of the cryptography, operation time is considered as an important parameter. Area efficiency is another factor that has been considered. This is because in hardware implementations, taking more silicon resource leads to more cost. Thus, in this thesis, both time and area are taken as performance parameters.

By scheduling the architecture and the operations of the cryptographic algorithms, the flexibility of the algorithms is also considered in this thesis. In the architecture for the Tate pairing, there are arithmetic units over $GF(2^{4m})$ such as additions, squarings, multiplications and divisions. These units can be implemented as operations over $GF(2^m)$. In practical implementation of the Tate pairing algorithm, the architectures can be reconfigured for different design constraints, such as the number of multipliers used in the design and the size of multipliers. This results in trade-offs between calculation time and area. In this thesis, the architectures for the Tate pairing algorithm are described using VHDL, and then synthesized for a particular FPGA technology. A software program is used to automatically generate the VHDL implementation of the architectures for different constraints. This software along with the use of FPGAs, allows the implementation of different designs to be quickly analyzed.

Apart from cost and efficiency, security is another important aspect of a cryptosystem. A new class of attack on cryptography known as side channel analysis (SCA) attacks [14], which monitors the side channel information (electromagnetic radiation, timing, power [13, 14, 15]) of a cryptosystem, has been developed to reveal the knowledge of the secret key. In this thesis, the security of the Tate pairing is considered. Several different attacks are applied on the Tate pairing designs to test their resistance against SCA attacks. For different kinds of protocols, the security is discussed. Against SCA attacks, several countermeasures are studied, including utilizing the bilinear property of the Tate pairing [16, 17], randomizing Miller variables in the algorithm [17], and using projective coordinates to mask the operations [18]. The proposed countermeasures are implemented in this work and the operation time and area costs of their implementations are evaluated, along with the consideration of the security aspects of the countermeasures.

1.3 Thesis outline

In this thesis, η_T algorithm, a fast approach to calculate the Tate pairing is studied. Considering the constraints of time and area, the scheduling methods of implementing the η_T pairing algorithm will be discussed. For security aspect of the η_T pairing algorithm, the important applications, the pairing based IBE system [5, 4] and ABE system [6] will be considered in this work. In the IBE or ABE cryptosystem, how the η_T pairing algorithm can be attacked under certain conditions is discussed. With all these possible attacks available to an opponent, countermeasures must be considered by the cryptosystem designer. Three countermeasures are studied and applied to protect the η_T pairing algorithm.

Chapter 2 introduces the theoretical material behind private and public key cryptography. In particular, the pairing based cryptosystems and the elliptic curve on which pairings are based and their underlying fields arithmetic are covered. The details of some applications of the pairing based cryptography, for example the IBE and ABE schemes are introduced. Cryptanalysis is the ever present force finding weaknesses and spurring the development of cryptography. To improve and perfect the cryptosystem, this thesis applies correlation power analysis (CPA) attacks on the hardware implemented pairing based cryptosystems. Introduction of the mathematical theory of CPA and the FPGA platform on which the proposed cryptosystem is implemented is also given in this chapter.

In Chapter 3, the size of the Galois Field where the Elliptic curve lies on is determined. All calculations in the pairing based algorithms are operated over the chosen Elliptic curve. The basic operation blocks of Addition and Squaring are introduced. Two popular methods of implementing field Multiplication, the digit-serial multiplication and the Karatsuba Multiplier are discussed. For the implementation of field Division/Inversion, an Itoh-Tsujii algorithm which makes use of the squaring and the multiplication to implement an inversion operation over $GF(2^m)$, and a dedicated algorithm for Division over $GF(2^m)$, the Extended Euclidean Algorithm (EEA) are introduced. These operations form the basic calculation units of a pairing based algorithm.

Chapter 4 introduces the structure and design flow of the η_T pairing over select Elliptic curve. As the operation field of the η_T pairing is raised to an extended field $GF(2^{4m})$, the operations for addition, squaring, multiplication and division are introduced. The top level architectures of the implementation of the η_T pairing are presented. Since there are 7 multiplications in the main loop in the η_T pairing calculation, when using the digit-serial multiplier, these multiplications dominate the calculation time. Different schedules for arranging these 7 multiplications are applied. The implementation results of η_T pairing using both the digit-serial multiplier and the Karatsuba multiplier are shown in this chapter.

In Chapter 5 the detailed calculation steps of CPA attacks are introduced. To examine the security aspect of the η_T pairing algorithm in a more practical and more general way, the important applications, the pairing based IBE scheme and ABE scheme, are chosen as the targets of the CPA attack. The weaknesses of the η_T pairing algorithm in the IBE system are discussed in this chapter. The CPA attacks are applied against every weakness of the algorithm, including the adder and the two multipliers. With the power traces of these operations collected using an oscilloscope, this chapter presents how an attacker tries to attack such components making use of the side channel information leaked during the operations. The target components of each proposed attack, the operation steps, and the mathematical calculation results of the attacks applied are described.

Chapter 6 introduces the precautions that must be used to implement a pairing based IBE system according to the weaknesses exposed by the attacks proposed in chapter 5. These precautions can be applied through carefully arranging the intermediate variables and the operation blocks by the hardware designers. However, applying such precautions does not provide perfect security to the Tate pairing algorithm. Some defects of the pairing based IBE and ABE cryptosystems must be fixed through additional operations. For this reason, several countermeasures to protect the pairing algorithm from CPA attacks are introduced and applied to the original η_T pairing. The implementation results of such countermeasures using the same top-level architectures are shown.

2. BACKGROUND THEORY

2.1 Introduction

This chapter presents the mathematical background and some of the theories related to the topics covered in this thesis. Section 2.2 briefly introduces cryptography. Section 2.3 explains the difference between private and public key cryptography. Some popular public key schemes, the IBE scheme and the ABE scheme, are introduced in this section.

In section 2.4, the basic mathematical concepts of groups, finite fields, elliptic curves and point operations over elliptic curves are introduced. In section 2.5, the basic theory of Tate pairing and an example of pairing based IBE and ABE schemes, are presented. Discrete logarithm problems (DLPs), serving as the main security challenge in each case, are discussed.

Section 2.6 introduces some popular cryptanalysis measures of attacking public key cryptography. Mathematical analysis methods and their complexity are considered in this section. As a popular method to attack hardware implemented cryptosystems, side channel analysis attacks are introduced. The security of pairing based IBE and ABE schemes, as examples of pairing based public key schemes, is discussed in this section.

Section 2.7 introduces some dedicated hardware accelerators used to implement the pairing based cryptographic algorithms and the reason why they are used. Two different types of FPGA platforms used in the thesis are introduced in this section. The method of evaluating the power consumption of a hardware accelerator is also explained.

2.2 Cryptography

Cryptography protects information and information systems from unauthorized access or from being modified [1]. Thousands of years ago people realized that it was necessary to protect the confidentiality of important information. Secrets should always be kept secure from being accessed and tampered with by unintended recipients. Julius Caesar, the Roman Emperor [34] was credited with the invention of the Caesar cipher in approximately 50 B.C., which was created for the protection of the secret and important messages being transmitted between Caesar and his military. This Caeser Cipher encrypts a message by substituting letters in the message with letters a fixed number of positions down the alphabet. The number of places the letters were shifted along the alphabet is called the secret key. Over the years, novel ciphers were invented and used in cryptography to make cryptosystems more and more secure. World War II resulted in significant improvements in information security and marked the beginning of the professional field of modern cryptography.

As modern electronics develops, the development of the internet and personal computers calls for cryptographic protocols that are suitable for daily communications. In many applications, keeping the information transferred between public communication platforms secret can be critical to politics, business and personal interests. Digital cryptography has developed in order to meet these security needs [35].

Cryptography, as a system of protecting information, is of course not just about keeping private information from being read by an unintended recipient. As a safeguard of the modern communication via electronic media, modern cryptography encompasses Confidentiality, Integrity and Availability. Other principles, such as Authenticity and Non-Repudiation, are also considered to be very important [36].

- Confidentiality: A message should not be disclosed to unauthorized individuals or systems.

- Integrity: It is always possible to check the message has not been altered while in transit.

- Availability: A cryptosystem must remain available at all times.

- Authenticity: The identities of the sender and recipient, and the data being transmitted, are genuine.

- Non-Repudiation: The sender cannot deny having sent a particular message. The recipient cannot deny having received a particular message.

2.3 Private and Public Key Cryptography

To provide the properties mentioned above in section 2.2, different protocols have been developed. There are two different types of cryptography nowadays: private key cryptography and public key cryptography.

Private key cryptography, also called symmetric key cryptography, uses a single secret key k which is shared by both the sender and the receiver. This key is used both to encrypt and decrypt the information. Both sender and receiver need the key and the key must be kept secret from anyone else. The security of the transmission depends on how well the key is protected.

Public key cryptography uses two different keys to encrypt and decrypt: a public and a private key, respectively. Each user has its own key set and while the private key must be kept secret the public key is publicly available to everyone. Both keys are mathematically related.

2.3.1 Private Key Cryptography

A communication between Alice and Bob using the private key cryptosystem is illustrated in Fig. 2.1. In the transmission, Alice first encrypts the plaintext m using the encryption function E and the secret key k. The encrypted text is called the ciphertext c. Then Alice sends c to Bob through the insecure channel. The cipher text c is of no use to anyone except Bob because Bob holds the same shared secret key k and can recover the message from ciphertext c to plaintext m using a decryption function D. For security reasons, the primary requirement of this cryptosystem is that it must be constructed so as to prevent an eavesdropper from simply trying every possible key (known as brute force attack [37]).

Private key cryptosystems are considered very fast and, thus, are suitable for the transmissions of a large throughput of data. However, there are two problems to consider: the key distribution problem and the key management problem. The key distribution problem is how Alice and Bob agree the value of their shared secret key. A third party key generator over some secure channel is necessary to distribute the key. The key management problem is that for each pair of users in the communication network, there must be a unique key. Therefore, for a network of n users, at least $\frac{n(n-1)}{2}$ unique keys are required [38]. For a large network, the number of unique keys becomes difficult to manage.

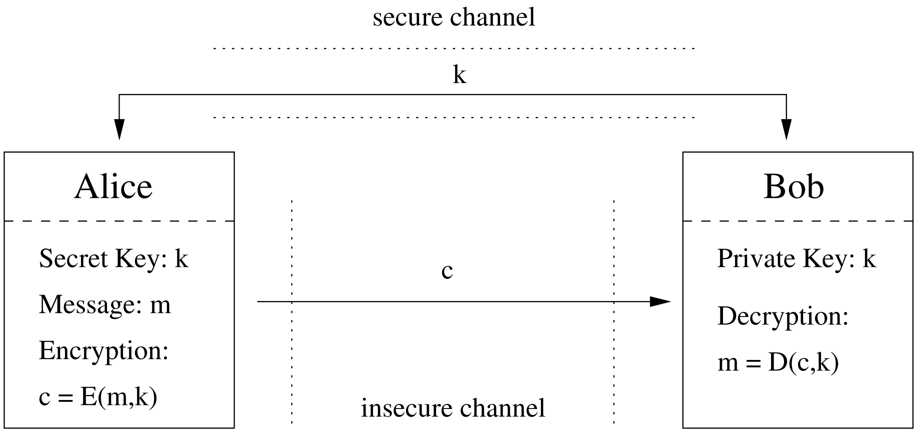


Fig. 2.1: Private key scheme communication

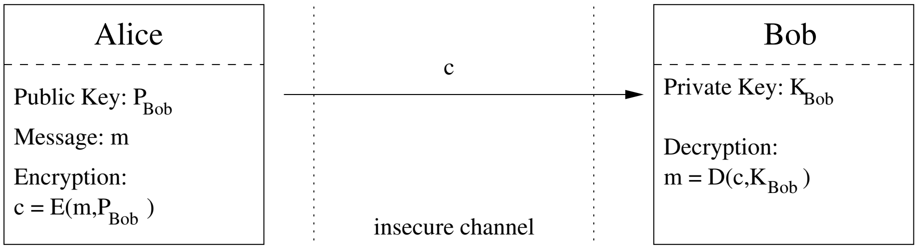


Fig. 2.2: Public key scheme communication

2.3.2 Public Key Cryptography

Consider the communication between Alice and Bob again, as shown in Fig. 2.2. In a Public key cryptosystem, Alice has a public and private key pair: (P_{Alice}, K_{Alice}), respectively. Similarly, Bob has (P_{Bob}, K_{Bob}). If Alice wants to send Bob a message m, she firstly encrypts the message with Bob's public key, P_{Bob}. The encrypted message is then sent to Bob and can only be decrypted using Bob's private key, K_{Bob}. Similarly, anyone who knows Alice's public key can send her a message by encrypting it with her public key P_{Alice}. Alice will then decrypt the message with her private key K_{Alice}.

In this scheme, the key distribution problem is solved because the public key is available to everyone. Alice and Bob do not need to agree a shared secret key. Similarly, anyone who wants to communicate with Bob can encrypt their message with the same public key P_{Bob}. Thus, key management is no longer a problem. The British government claimed that James H. Ellis, Clifford Cocks and Malcolm Williamson, members of the British Government Communicationis Headquarters (GCHQ), first developed the public key algorithms in 1973. The reader may refer to [39] for the story. However, the most famous public key scheme is RSA which was developed in 1977 by Rivest, Shamir and Adleman in the US [40].

The disadvantage of public key cryptography is that it is much more computationally complex than private key cryptography. Although public key cryptography eliminates the key distribution and key management problems, it brings some problems of its own. The main problem is the confirmation of the authenticity of the public key. In the communication model, Alice has to consider whether the public key of Bob, received from an insecure channel, is the real one or a value sent by an attacker.

In a public key cryptosystem, since Alice cannot confirm whether the public key is authentic or not, a Public Key Infrastructure (PKI) can be introduced [41]. The PKI includes a trusted third party that provides the service of a Certificate Authority (CA) [42]. CA certifies Bob's public key as belonging to him. Before Alice sends Bob a message, she has to validate the public key of Bob by contacting the CA. This adds an additional step in the communication between Alice and Bob. It is very important that the CA is trustworthy, otherwise Alice may receive false keys disguised by some attackers and any message Alice encrypted using the false keys may be easily decrypted by the attacker.

2.3.3 Identity Based Encryption

In 1984 Shamir [5] proposed a public key scheme in which the public key can be an arbitrary string. This scheme is called IBE. The first practical IBE implementation was applied by Boneh and Franklin in 2001 [4]. IBE was originally developed to simplify the certification process. It eliminates the need for the CA. Thus, a PKI is no longer necessary in the scheme. Instead, a trusted third party Public Key Generator (PKG) is introduced which is used to distribute the private key of a receiver.

The structure of an IBE scheme is shown in Fig. 2.3. In the figure, P_{Bob} and K_{Bob} represent the keys to encrypt and to decrypt respectively. An encrypted message 'm' can be readily decrypted by the intended recepient Bob only.

In an IBE system, each user is identified by a unique identity string, for example a user name or an e-mail address. The identity of Bob in Fig. 2.3 is ID_{Bob} . The IBE system can be described in 4 steps: setup, key generation, encryption and decryption.

Setup A trusted third party, the PKG, publishes the algorithm E for encryption and D for decryption, a random generator g and some related rules such as how the user's public key can be generated. All elements used in this cryptosystem are a

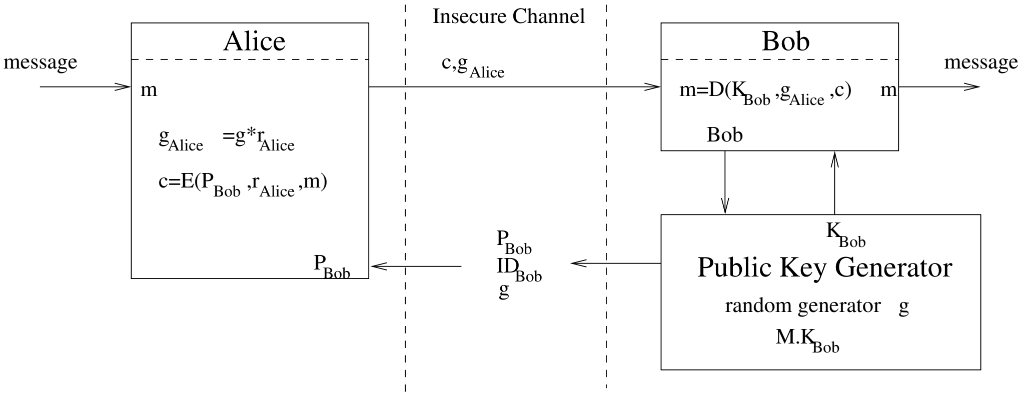


Fig. 2.3: Identity-Based Encryption scheme structure

multiple of the generator g.

Key generation The PKG randomly picks a master key $M.K_{Bob}$ for Bob. With this master key, the PKG generates a pair of keys, a public key P_{Bob} and a private key K_{Bob}, for user Bob and publishes the public key P_{Bob} of ID_{Bob} to the insecure channel.

Encryption Sender Alice in this IBE scheme can get the public key P_{Bob} according to the information published by the PKG (without certificates). Alice then picks a random number r_{Alice} and generates an identity number g_{Alice} using generator g: $g_{Alice} = g * r_{Alice}$. This g_{Alice} helps Bob recognize the sender of the received message. The rules published by the PKG must ensure that the decryptor can remove r_{Alice} using g_{Alice}. With the elements above, Alice can encrypt a message 'm' by calculation $c = E(m, P_{Bob}, r_{Alice})$. Alice sends Bob the message pair $\{c, g_{Alice}\}$.

Decryption The PKG is responsible for delivering the private key K_{Bob} to the authorised recipient Bob according to his ID. When Bob wants to decrypt the message, he has to contact the PKG and ask for his corresponding private key. On confirmation of ID_{Bob}, the PKG sends Bob his private key K_{Bob} through some secure channel. With the received message pair $\{c, g_{Alice}\}$ and the private key K_{Bob}, Bob is able to decrypt the message by calculation: $m = D(c, K_{Bob}, g_{Alice})$.

In this scheme, it is no longer necessary that the message sender contact a third party to validate the public key of a receiver. It is easier to send a message as the onus is on the receiver to contact the PKG and verify his identity to obtain the private key. This scheme also allows the PKG to control the validity of a customer's identity. If the PKG finds that the identity of Bob is no longer valid, it can change the master key and thus, the private key that Bob received before no longer works.

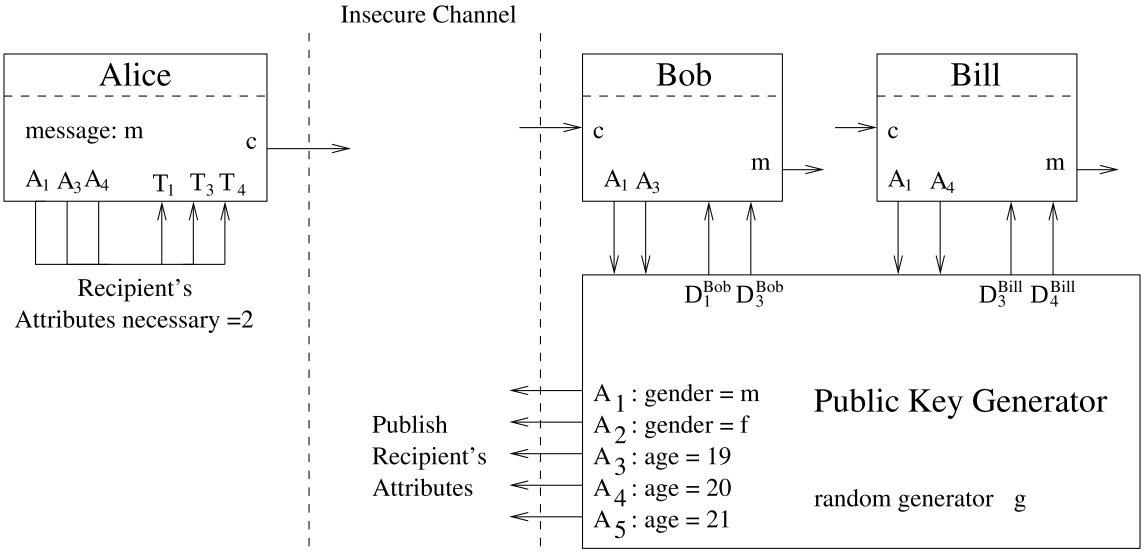


Fig. 2.4: Attribute-Based Encryption scheme structure

2.3.4 Attribute Based Encryption

ABE [6] is a new kind of public key encryption which is based on IBE [5]. In contrast to the IBE schemes, ABE is a scheme in which each user is identified by a set of attributes, for example age, gender, college, etc. Some function of those attributes is used to determine decryption ability for each ciphertext.

Sahai and Waters introduced a single authority attribute encryption scheme in [6] and named this scheme a 'Fuzzy identity-based encryption'. In this scheme, a trusted third party is needed to monitor all the attributes of the users. This authority is in charge of delivering the secret keys corresponding to each of the attributes to the intended recepients.

There are two kinds of ABE systems, Key-Policy ABE (KP-ABE) and Ciphertext-Policy ABE (CP-ABE). In KP-ABE [6, 43], every ciphertext is associated with a set of attributes and the secret key of every user is associated with a threshold access structure based on attributes. Decryption is enabled if and only if the ciphertext attribute set satisfies the access structure of the user secret key. In CP-ABE, [44, 45, 46, 47, 48], the situation is reversed: each ciphertext is associated with an access structure.

2.3.5 ABE structure

The structure of a basic ABE scheme is shown in Fig. 2.4. In the figure, A_i represents different attribute parameters. In this scheme, a sender can designate the recepient to be a set of elements holding the same specific attributes. For example, a boys' soccer team recruiting e-mail may be sent to all boys between 19 - 20 years old. In an ABE system, all members with attributes *age* = 19 or 20 and attribute *gender* = *male* will be qualified to decrypt such an e-mail. In this case, Bob and Bill, respectively 19 and 20 years of age, are able to read such an e-mail whilst the girl Eva does not get access to such information.

Similar to IBE systems, an ABE system can be described using the following steps: setup, encryption, key generation and decryption [43]. Suppose there are 5 attributes in the ABE system shown in Fig. 2.4, i.e. A_i, where $i \in \mathbb{Z}_5$. Taking Bob as an example, the operations in the ABE scheme are described as follows:

Setup This is a randomized algorithm that takes no input other than the implicit security parameter. In the ABE scheme shown in Fig. 2.4, the Key Generator publishes the attribute set γ_{total}=$\{A_i\}$ and a public key set PK=$\{T_i\}$. It keeps the master key MK which was used to generate the public key and will be used to generate the secret key SK.

Encryption This is a randomized algorithm that takes as input a message m, a set of attributes γ_e with n_e attributes and the public parameters PK. It outputs the ciphertext c and a number d. In Fig. 2.4, Sender Alice picks 3 attributes $\gamma_e = \{A_1, A_3, A_4\}$ out of the 5 attributes and uses the corresponding public key $\{T_1, T_3, T_4\}$ to encrypt the message m, into ciphertext c and defines a number $d = 2$ which means that any decryptor should hold at least 2 of the attributes in γ_e to decrypt the message.

Key Generation This is a randomized algorithm that takes as input an access structure γ_1, the master key MK and the public parameters PK. It outputs a decryption key SK=$\{D_i\}$. In Fig. 2.4, this step shows the communications between Bob and the Key Generation algorithm. On trying to get the access to the message, Bob has to contact the Key Generator with the attributes, γ_{Bob}=$\{A_1, A_3\}$, he holds and ask for the corresponding secret key. The Key Generator generates a secret key set $SK^{Bob} = \{D_1^{Bob}, D_3^{Bob}\}$ according to the master key and the attribute set input by Bob and sends SK^{Bob} back to the decryptor. In $SK^{Bob} = \{D_1^{Bob}, D_3^{Bob}\}$ the superscript '*Bob*' implies that the secret key set is randomly generated for Bob.

Decryption This algorithm takes as input the ciphertext c that was encrypted under the set γ_e of attributes, the decryption key SK=$\{D_i\}$ for access control structure γ_{Bob} and the public parameters PK. It outputs the correct message m if $|\gamma_e \cap \gamma_{Bob}| \geq d$, where '$||$' represents the number of elements. In Fig. 2.4, on receiving the secret key set $SK^{Bob} = \{D_1^{Bob}, D_3^{Bob}\}$, Bob is able to compute the plain text of the message.

No matter how many attributes Bob holds, provided at least 2 of the attributes in γ_{Bob} match with those in γ_e, the message will be correctly decrypted. Otherwise, Bob will not have the access to the plain text.

For Bill, the same steps are performed and Bill will get a secret key set $SK^{Bill} = \{D_1^{Bill}, D_4^{Bill}\}$ for decryption. Thus, this system enables an encryptor to send a message to a group of recepients who match the requirements he sets up. Note that the secret key sets, SK^{Bob} and SK^{Bill}, are randomly generated for specific decryptors. Although Decryptor1 and Decryptor2 hold 3 attributes in total ($\{A_1, A_3, A_4\}$), merging the information from Decryptor1 and Decryptor2 generates the secret key set $\{D_1^{Bob}, D_3^{Bob}, D_1^{Bill}, D_4^{Bill}\}$ rather than $\{D_1^{Bob}, D_3^{Bob}, D_4^{Bob}\}$ or $\{D_1^{Bill}, D_3^{Bill}, D_4^{Bill}\}$. This means that unqualified decryptors cannot forge secret keys, even by colluding.

2.4 Mathematical Background

In this section the necessary mathematical background is introduced. The concepts of groups, fields and elliptic curves are necessary for building the cryptosystem presented in this thesis.

2.4.1 Groups

In mathematics, a group is an algebraic structure which consists of a set of elements and an operation [49]. Such an operation is called the group operation. The group operation operates on any two of its elements to form a third element. For example, the set of integers is a group and addition and multiplications are both group operations of the integer group. A group must satisfy the following properties under the '+' operation:

- Closure: the result of the operation is still in the group, $c = a + b$, if $a, b \in G$, then $c \in G$,
- Associativity: the group operation order does not affect the operation result, $(a + b) + c = a + (b + c)$ for any $a, b, c \in G$.
- Identity element: There exists an identity element $e \in G$, such that for every element $a \in G$, the equation $e + a = a + e = a$ holds.
- Inverse element: For each $a \in G$, there exists an element $b \in G$ such that $a + b = b + a = e$.

A group G is said to be Abelian or commutative if $a + b = b + a$, $a, b \in G$. A group G is said to be finite if there is a finite number of elements in the group. The order of this group is defined as the number of elements in the group, denoted $\#G$. For example, the integer Z with addition as its group operation, is an infinite group, its order is infinity. A group made up of integers modulo p is a finite group, denoted Z_p, where p is the order of this group.

A group G is said to be cyclic if there exists an element g such that for any element $a \in G$, there exists integer k such that $[k]g = a$. Here the operation $[k]g = a$ is the k times addition chain of g, as shown in equation 2.1.

$$a = [k]g = \underbrace{g + g + ... + g}_{`+' \, k-1 \, times} \tag{2.1}$$

Here the element g is called the generator of this cyclic group G. The cyclic group generated by g is denoted $< g >$. The smallest integer n that satisfies $[n]g = e$ is the order of this group.

2.4.2 Finite Fields

Mathematically, a field is a cyclic group of elements in which nonzero elements form a group under multiplication [50]. In a field, notions of addition, subtraction, multiplication and division satisfy certain axioms. A field F must satisfy the following properties:

- Closure: $c = a + b, d = a * b$, if $a, b \in G$, then $c, d \in F$.
- Associativity: $(a+b)+c = a+(b+c)$ and $(a*b)*c = a*(b*c)$ for all $a, b, c \in F$.
- Commutative: $a + b = b + a$ and $a * b = b * a$ for all a and $b \in F$.
- Distributive: $a * (b + c) = a * b + a * c$ for all $a, b, c \in G$.
- Inverse: for all $a \in F$, there always exists an element $a^{-1} \in F$ such that $a * a^{-1} = 1$.

The most commonly used fields are the field of real numbers and the field of complex numbers which are infinite.

The Galois Field is a finite field named after Èvariste Galois. The order of a Galois Field must be equal to the positive integer power of a prime p. Èvariste Galois showed that for any prime p and positive integer m, there exists a finite field

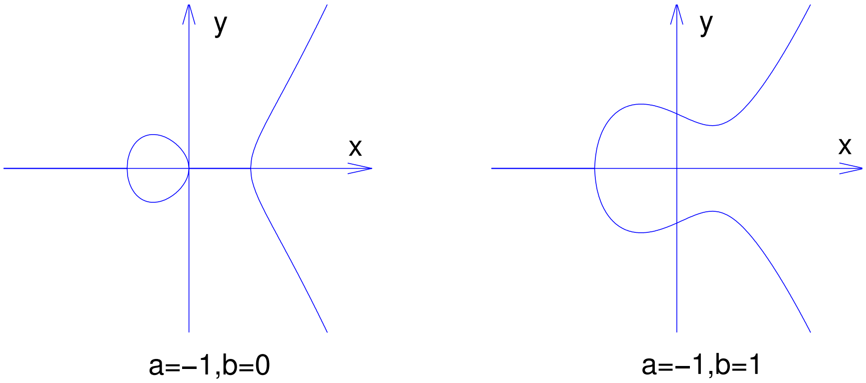


Fig. 2.5: Example of Elliptic Curve $y^2 = x^3 + ax + b$

with $q = p^m$ elements [52]. The prime p is known as the characteristic of the field $GF(p^m) = GF(q)$. In cryptographic applications the characteristics of $p = 2$, $p = 3$ and p some large prime are often used. Fields with characteristic two show better performance in area, runtime, power and energy than prime fields [53]. Page showed that finite fields of characteristic two and three result in similar performance [54]. In this work, fields of characteristic 2 with power $m \geq 163$ are used, denoted $GF(2^m)$.

2.4.3 Elliptic Curves over Finite Fields

An elliptic curve $E(GF(q))$ over $GF(q)$ can be represented in equation 2.2 [51].

$$E(GF(q)) : y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6, \tag{2.2}$$

where $a_1, a_2, a_3, a_4, a_6 \in GF(q)$. Any point P over this curve consists of two coordinates x and y, which satisfy equation 2.2 as a pair, i.e. $P(x, y) \in E(GF(q))$, $x, y \in GF(q)$. An example of an elliptic curve over the field of real numbers is shown in Fig. 2.5. The two curves in Fig. 2.5 are described by equation 2.3. For simplicity, let a and b represent a_4 and a_6 in equation 2.2, respectively.

$$E(GF(q)) : y^2 = x^3 + ax + b, \tag{2.3}$$

Restricting the elliptic curve over some special finite fields, for example $GF(2^m)$, simplifies the curve equation 2.2. Equation 2.4 describes an elliptic curve over $GF(2^m)$.

$$E(GF(2^m)) : y^2 + xy = x^3 + ax + b, \tag{2.4}$$

The number of points on the curve, denoted $\#E(GF(2^m))$, is the order of the curve. In Hasse's Theorem [52], the number of points can be calculated as $\#E(GF(2^m)) = 2^m + 1 - T_r$ where T_r is called the Trace of Frobenius. Hasse pointed out that $T_r \leq 2\sqrt{2^m}$. The calculation of T_r is introduced in [55]. For elliptic curves used in cryptographic schemes, 2^m is usually very large. Thus, $2\sqrt{2^m}$ is relatively small compare to 2^m and the order of the curve can be represented by $\#E(GF(2^m)) \approx 2^m$. In Elliptic Curve Cryptography (ECC), the order of the curve represents the number of possible values that can be used in the cryptosystem. If 2^m is divisible by T_r, the curve is supersingular. Otherwise, it is non-supersingular or ordinary. Supersingular and non-supersingular elliptic curves behave fundamentally differently in many aspects. For some certain supersingular elliptic curves, the operations can be optimized. Consequently, the calculation time of the cryptographic algorithms based on such curves can be reduced.

2.4.4 Point Operations over Elliptic Curves

There is a specified point O which represents the point at infinite. This point is also the identity element of this elliptic curve. The point at infinity, O, together with the curve itself, form a group which provides some useful properties for cryptography.

The group operation of Elliptic Curve is point addition, given by $P2 = P0 + P1$ for $Pi \in GF(2^m)$. It operates on two input points $P0$ and $P1$ and forms a third point $P2$. For the point at infinity O there exists $P + O = O + P = P$, where $P \in GF(2^m)$. The geometrical explanation of this point addition operation is given by Ian Blake in [51]. Let $P0$ and $P1$ be two distinct rational points on curve $E(GF(2^m))$. The $E(GF(2^m))$ is a cubic curve, thus, the straight line $d(x, y)$ joining $P0$ and $P1$ must intersect the curve at a third point, called $P2'$ which is a rational point on $E(GF(2^m))$. Reflecting $P2'$ with the x-axis, one obtains another rational point $P2 = P0 + P1$. Fig. 2.6(a) shows a visualization of the point addition over elliptic curve $E(GF(2^m)) : y^2 = x^3 - x + 1$.

In point addition, the special case $P0 = P1$, i.e. $P2 = P0 + P0 = [2]P0$, is called point doubling. In this case, the tangent to the curve at point $P0$, $d(x, y)$, must intersect the curve at exactly one other point $P2'$, as $E(GF(2^m))$ is a cubic curve. Again, reflecting $P2'$ about the x-axis, one obtains another rational point $P2 = P0 + P0 = [2]P0$. Fig. 2.6(b) shows a visualization of point doubling over the reals.

For an elliptic curve $E(GF(2^m))$ defined by equation 2.4, let $P0(x_0, y_0)$ and $P1(x_1, y_1)$ be the points on the curve given in affine coordinates. Assume $P0, P1 \neq O$

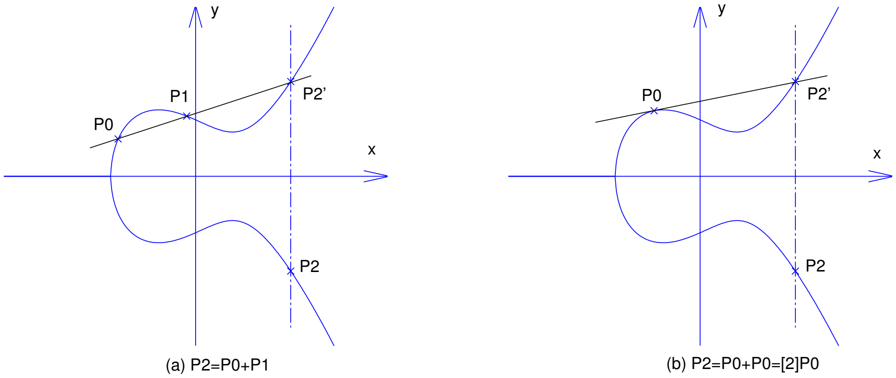


Fig. 2.6: Point Addition and Point Doubling over Elliptic curves

and $P0 \neq -P1$. The arithmetic operations for point addition $P2 = P0 + P1$ and point doubling $P2 = P0 + P0$ over the field $GF(2m)$ are given as follows.

$$\begin{aligned} &P2 = P0 + P1 \\ &\lambda = \tfrac{y_0+y_1}{x_0+x_1} \\ &x_2 = \lambda^2 + \lambda + x_0 + x_1 + a \\ &y_2 = (x_0 + x_2)\lambda + x_2 + y_0 \\ &Cost = 2M + 1I + 1S + 8A \end{aligned} \tag{2.5}$$

$$\begin{aligned} &P2 = P0 + P0 = [2]P0 \\ &\lambda = \tfrac{y_0}{x_0} + x_0 \\ &x_2 = \lambda^2 + \lambda + a \\ &y_2 = (x_0 + x_2)\lambda + x_2 + y_0 \\ &Cost = 2M + 1I + 1S + 6A \end{aligned} \tag{2.6}$$

In equations 2.5 and 2.6, λ represents the slope of the straight line $d(x, y)$ used in the chord and tangent illustrated in Fig. 2.6(a) and 2.6(b), respectively. **M**, **I**, **S** and **A** represent the operations of multiplication, inversion, squaring and addition over field $GF(2^m)$, respectively. For an elliptic curve over a finite field $GF(2^m)$, the group of points $E(GF(2^m))$ is always either a cyclic group or the product of two cyclic groups [56]. Based on point addition and point doubling, and recalling the successive additions for cyclic groups in equation 2.1, a Point Scalar Multiplication is defined as per equation 2.7:

$$Q = [k]P = \underbrace{P + P + ... + P}_{`+' \; k-1 \; times} \tag{2.7}$$

where P and Q are points on curve $E(GF(2^m))$ and k is an integer. Let ℓ be the order of point P and k can be an arbitrary integer in the range $1 \leq k \leq \ell$. Equation 2.7 can be simply carried out by repeated point additions. But, for large k, this can be slow in practice. A simple method to speed up Point Scalar Multiplication

Algorithm 2.1 Binary Method for Point Scalar Multiplication

Input: $P \in E(GF(2^m))$, n-bit integer $k = \sum\limits_{j=0}^{n-1} k_j 2^j, k_j \in \{0, 1\}$

Output: $Q = [k]P \in E(GF(2^m)))$

```
$Q \leftarrow O$
for $j = n - 1$ downto 0
  $Q \leftarrow [2]Q$
  if $k_j = 1$
    $Q \leftarrow Q + P$
  end if
end for
return Q
```

is to make use of both point addition and point doubling operations. The binary method used to calculate equation 2.7 relies on the binary expansion of k, as given in Algorithm 2.1.

This algorithm takes point P over the elliptic curve and an arbitrary integer k, consisting of n bits, as inputs. It iterates through the n bits of k. For each bit of k, a point doubling operation is performed. If the present bit of k equals '1', a point addition operation is also performed. Let W represent the number of bits that equal to '1' in the binary expansion of k, i.e. $W = \sum_{j=0}^{n-1} k_j, k_j \in \{0, 1\}$. The Point Scalar Multiplication requires $n - 1$ point doublings and $W - 1$ point additions.

2.4.5 Elliptic Curve Discrete Logarithm Problem

In a cyclic group G with a generator g and a multiplicative group operation '×', exponentiation is defined as

$$h = g^x \mod n \tag{2.8}$$

where the generated element h is an element in group G, n is the order of the group and x is some positive integer. The discrete logarithm of h is x, i.e. $\log_g h = x$. The problem of determining the smallest positive integer x that satisfies equation 2.8 for given elements h and g in group G, is defined as a Discrete Logarithm Problem (DLP). In public key schemes, the group G should be sufficiently large so that the computation of g^x can be efficiently performed. However, solving the DLP is generally intractable in such a group. If there is a fast way to solve the DLP in group G, then this group G is insecure for a public key cryptography.

The operations over an elliptic curve offer a property suitable for DLP. i.e., over an elliptic curve $E(GF(q))$, it takes polynomial time to calculate a point scalar multiplication $Q = [k]P$ given k an integer and P a point over $E(GF(q))$. However, it takes exponential time to find out the smallest k such that $Q = [k]P$, given P and Q. The problem of solving for k under this circumstance is called the Elliptic Curve Discrete Logarithm Problem (ECDLP).

The Diffie-Hellman key exchange protocol for sending messages was applied over elliptic curves by ElGamal [57]. With the operation over elliptic curves, when Alice wants to send a message m to Bob, the communication changed as follows:

- Public Key Generator posts a generator: point $P \in E(GF(q))$.
- Alice generates a random integer $k_A \in 1 \sim \ell$ and sends Bob:$[k_A]P$.
- Bob generates a random integer $k_B \in 1 \sim \ell$ and sends Alice:$[k_B]P$.
- Alice can compute $[k_A k_B]P = [k_A]([k_B]P)$
- Bob can compute $[k_A k_B]P = [k_B]([k_A]P)$

In this communication, generator P is posted by the Public Key Generator, which must be a trusted third party. NIST presented some of the recommended elliptic curves and their generators P [58]. A mathematical theorem promises that in ECC, if a curve is of a prime order $\#E$, then each of its elements, other than the identity, is of order $\#E$ and, therefore, a generator of such curve [59].

In the communications between Alice and Bob above, the information exposed to an eavesdropper is the chosen elliptic curve $E(GF(2^m))$, a point over this curve P and the multiples, $[k_A]P$ and $[k_B]P$, of the point. The underlying operation in the Diffie-Hellman protocol over elliptic curves is point scalar multiplication. An attacker can try to solve for k_A or k_B, namely ECDLP. Other attackers may try to solve $[k_A k_B]P$, namely elliptic curve Diffie-Hellman problem (ECDHP). These two problems are considered to be of the same computational complexity [60]. The efficiency of the protocol depends on the implementation efficiency of the point scalar multiplication. The security of the protocol depends on how strong the ECDLP is.

2.5 *Pairing Based cryptography*

Pairing based cryptography is a relatively new type of public key cryptographic scheme based on Elliptic Curves. It was first used to attack elliptic curve cryptosystems. Menezes et al. [61] proposed the MOV attack, also known as the Weil Descent attacks, which uses Weil pairing to reduce the ECDLP on the elliptic curve over finite field $GF(q^l)$. In this attack the integer l must be carefully selected [62, 61] so that the ECDLP on $E(GF(q))$ can be mapped to a DLP in $GF(q^l)$. A Similar attack was proposed by Frey and Ruck [63] using a Tate pairing. In these pairing attacks, an ECDLP in $< P >$ can be mapped to the DLP over $GF(q^k)$ (see Extension field in section 2.5.1), which can be solved using one of the known sub-exponential methods [51].

As an efficient public key cryptographic scheme, pairings have been applied in IBE and ABE schemes [4, 6]. A pairing operates on two points in an additive group and maps them to a point in a multiplicative group. Let G_1 be an Abelian group with point addition as a group operation and 0 as its identity element. let G_2 be a cyclic group with multiplication as a group operation and 1 as its identity element. The pairing is described as:

$$e_l : e(G_1; G_1) = G_1 \times G_1 \to G_2 \tag{2.9}$$

Pairings have properties such as bilinearity and non-degeneracy which are of interest for many applications [64, 25, 65, 26]. All pairings considered in cryptosystems have the following properties:

- Bilinearity: $\forall P, P' \in G_1$ and $\forall Q, Q' \in G_2$, there exists $e(P + P'; Q) = e(P; Q)e(P'; Q)$ and $e(P; Q+Q') = e(P; Q)e(P; Q')$, consequently $e([k]P; Q) = e(P; [k]Q) = e(P; Q)^k$.

- Non-degeneracy: $\forall P \in G_1$, with $P \neq 0$, there is some $Q \in G_2$ such that $e(P; Q) \neq 1$. $\forall Q \in G_2$, with $Q \neq 0$, there is some $P \in G_1$ such that $e(P; Q) \neq 1$.

- Computability: $\forall P \in G_1$ and $Q \in G_2$, $e(P; Q)$ can be computed efficiently.

There are several pairing based algorithms, such as the Weil pairing, the modified Weil and the Tate pairing [66, 63, 67]. Among them, the Tate pairing has proved to be the most efficient in all fields for frequently used key sizes. For a security level from 80 to 256 bits, the Tate pairing requires less calculations than the Weil pairing [25]. The Tate pairing on supersingular elliptic curves described by Kwon [68] is the focus of this work.

2.5.1 Tate pairing over supersingular curves

Now some basic definitions used in the pairings are presented. Suppose E is an elliptic curve defined over $GF(q)$ and n is a positive integer which is coprime to the characteristic of field $GF(q)$, i.e. coprime to q. Suppose n divides $\#E(GF(q))$.

- Embedding degree k_{deg}: The embedding degree of an elliptic curve E is the smallest positive integer k_{deg} such that n divides $(q^{k_{deg}} - 1)$.
- Extension field: The field $GF(q^{k_{ext}})$ is a k_{ext} extension of $GF(q)$. Conversely, $GF(q)$ is a subfield of $GF(q^{k_{ext}})$.
- Distortion map ϕ: The rule that maps a point P over $GF(q)$ to another point P' over the k_{ext} extension field $GF(q^{k_{ext}})$ is a Distortion map ϕ [69], denoted $P' = \phi(P)$.
- l-torsion: All the points of order l on $E(GF(q))$ form an l-torsion subgroup, denoted $(GF(q))[l] = \{P \in E(GF(q)) : [l]P = O\}$.

From the definitions above, it can be seen that by picking the same degree k, i.e. $k = k_{deg} = k_{ext}$, an elliptic curve $E(GF(q))$ can be mapped to its extension field $E(GF(q^k))$ through some distortion map.

As the point operations over supersingular elliptic curves are simpler than the common curves over finite fields [68, 70], designers generally like to pick suitable supersingular elliptic curves for implementing the pairing algorithms. Let E_{ss} be a supersingular elliptic curve over $GF(q)$, the mathematical calculation of Tate pairing is a map of two points over supersingular curves:

$$e_l : e(P; Q) = \psi(P, Q)^{\zeta}, \tag{2.10}$$

where $\zeta = \frac{q^k - 1}{l}$. This mathematical expression shows a non-degenerate bilinear pairing. That is, for any $P \neq O \in E(GF(q))[l]$, there exists a point $Q \in E(GF(q))[l]$ such that $e(P, Q) \neq 1$. Also, there exist $e(P_1 + P_2, Q) = e(P_1, Q)e(P_2, Q)$ and $e(P, Q_1 + Q_2) = e(P, Q_1)e(P, Q_2)$. In equation 2.10, $\psi(P, Q)$ is a rational function with two input points P and $Q \in GF(q)$.

In 1986, Miller described the calculation of a pairing in his algorithm [64], providing an efficient way of finding ψ given points P and Q. This algorithm, known as Miller's algorithm, makes use of point addition and point doubling to calculate ψ. This algorithm was improved by many researchers since pairing based cryptosystems have been proposed [25, 65, 66].

The elliptic curve $E(GF(q))$ has point addition as its group operation. The elliptic curve $E(GF(q^k))^*$ indicates the multiplicative group of $GF(q^k)$, with point scalar multiplication as its group operation and 1 as its identity element. Consider the definition of pairing introduced in equation 2.9. Choose the l-torsion subgroup of $E(GF(q))$, denoted $E(GF(q))[l]$, to be G_1, the degree k extension field $E(GF(q^k))^*$ to be G_2. The pairing described in equation 2.9 is refined here to equation 2.11. This pairing takes as input two elliptic curve points of order l and gives a third point over the degree k extension field, namely the l-th order Tate pairing.

$$e_l : E(GF(q))[l] \times E(GF(q))[l] \rightarrow E(GF(q^k))^* \tag{2.11}$$

In 2002, Barreto, Kim, Lynn and Scott [65] showed that for some specific curves, there exists a fast way of calculating the Tate pairing. They proposed an algorithm called the BKLS algorithm for the Tate pairing over supersingular curves with embedding degree $k = 2, 4, 6$. Menezes [70] pointed out that the embedding degree $k = 6$ is attained for $GF(3^m)$ and the embedding degree $k = 4$ is attained for $GF(2^m)$. For cryptographic purpose, elliptic curves over finite fields of characteristic 2, i.e. over $GF(2^m)$ where m is odd or more strongly a prime, are considered. The recommended values of the binary field size m are listed in [58].

Duursma and Lee [26] proposed a closed formula for the Tate pairing which is a faster algorithm, called the η_T pairing algorithm and Kwon [68] further improved upon it. The η_T pairing algorithm simplifies the main loop and the final exponentiation of the BKLS algorithm. This algorithm omits the denominators in Miller's algorithm. This greatly simplifies Miller's algorithm because the costly division is no longer necessary. It chooses the specific elliptic curve E_b described in equation 2.12.

$$E_b : \;\; y^2 + y = x^3 + x + b, \tag{2.12}$$

where b=0,1. Curves E_b have embedding degree $k = 4$. In [68] Kwon proved that on supersingular elliptic curve E_b over binary fields, the Tate pairing and the η_T pairing calculate the same result, i.e. $e(P; Q) = \eta_T(P; Q)$. The η_T pairing algorithm is a fast calculation of Tate pairing, and so this work continues to use the notation $e_l : e(P; Q)$ to represent an η_T pairing calculation. The Tate pairing calculation over E_b can be described as:

$$e_l : E_b(GF(2^m)) \times E_b(GF(2^m)) \rightarrow E(GF(2^{4m})) \tag{2.13}$$

Algorithm 2.2 η_T Algorithm for computing Tate pairing

```
Input: $P = (\alpha, \beta), Q = (x, y)$
Output: $C = \eta(P; Q)$
1: $C(t) \leftarrow 1$
2: $u \leftarrow x^2 + y^2 + g + \frac{m-1}{2}, v \leftarrow x^2 + 1, \alpha \leftarrow \alpha^4, \beta \leftarrow \beta^4, \gamma \leftarrow \alpha v$
3: for $i = 0$ to $m - 1$
4:     $A(t) \leftarrow \gamma + u + \beta + (\alpha + v)t + (\alpha + v + 1)t^2$
5:     $C(t) \leftarrow C(t)^2 \times A(t)$
6:     $u \leftarrow u + v, v \leftarrow v + 1, \alpha \leftarrow \alpha^4, \beta \leftarrow \beta^4, \gamma \leftarrow \alpha v$
7: endfor
8: $C(t) \leftarrow C(t)^{2^{2m}-1}$
9: return $C(t)$
```

This indicates that the Tate pairing maps two points over the same supersingular curve over $GF(2^m)$ to an element over $GF(2^{4m})$.

2.5.2 Implementing the Pairings

The η_T algorithm [68] for calculating the Tate pairing is shown in Algorithm 2.2. This algorithm takes as inputs two points $P(\alpha, \beta)$ and $Q(x, y)$ over $E(GF(2^m))$ and results in an element $C(t)$ in the extension field $GF(2^{4m})$.

In Algorithm 2.2, the coordinates of input points $P(\alpha, \beta)$ and $Q(x, y)$ are processed in step 2 to generate intermediate variables u, v, α, β and γ which are used to calculate $A(t)$ in every iteration of the 'for' loop in step 3, m times. $A(t)$ is regenerated in each iteration in step 4 by the basic operations over $GF(2^m)$ between the intermediate variables, namely the Miller variables. It should be noted that A_3 always equals 0. In step 5, symbols '×' and '^{2}' here represent a multiplication and a squaring over $GF(2^{4m})$ [77] which can be decomposed into operations in $GF(2^m)$ and will be discussed in Chapter 4. This step updates $C(t)$ in every iteration of the 'for' loop. After the 'for' loop, a final exponentiation operation of $C(t)$ over $GF(2^{4m})$ is applied in step 8. This is because the value of $C(t)$ returned at the end of the 'for' loop is in the l-th roots of unity. The final exponentiation to the power of $(2^{2m} - 1)$ provides a unique value. This algorithm returns the Tate pairing result $C(t)$.

2.5.3 Bilinear Pairings applied in IBE

Pairing based cryptography can be used in many public key protocols, for example key distribution protocols [78, 79] and IBE [4]. Interested readers are referred to the survey in [80]. In this section the IBE scheme of Fig. 2.3 is described using pairings

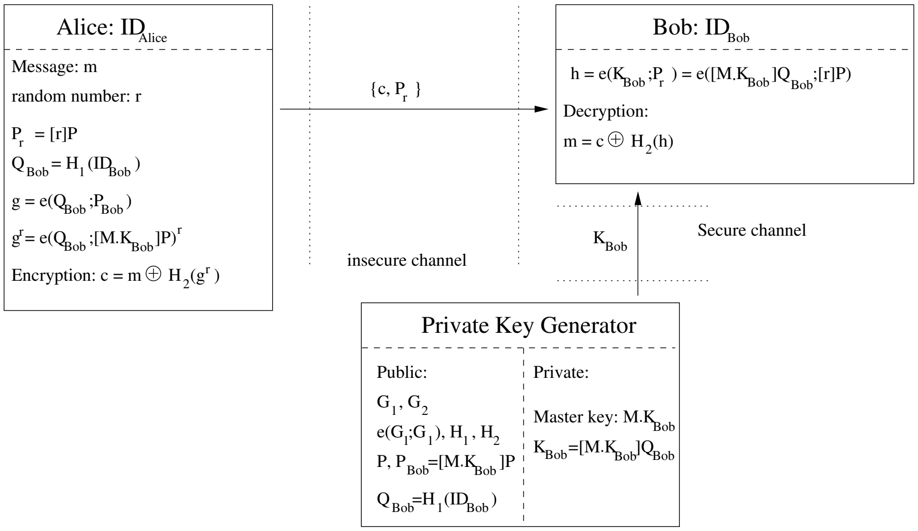


Fig. 2.7: Pairing based IBE scheme

and is illustrated in Fig. 2.7.

The main steps in the IBE scheme considered here can be summarized as follows:
Setup In this system, PKG picks a master key $M.K_{Bob}$ for user Bob and publishes system parameters set $<G_1, G_2, e, P, H_1, H_2, P_{Bob}, Q_{Bob}>$:

- Group G_1 - an elliptic curve from which the initial points are picked.

- Group G_2 - the extended field to which the two points are mapped.

- Bilinear map $e : G_1 \times G_1 \rightarrow G_2$ - the rule governing how initial points are mapped to the extended field.

- Group generator P - which generates the l-torsion group G_1.

- Cryptographic hash functions [37] $H_1 : \{0,1\}^* \rightarrow G_1$ - maps a string of any length to an element in group G_1, i.e. transform any string to a point over curve G_1. For example, Bob's ID can be transformed to an elliptic curve point over G_1.

- Hash functions $H_2 : G_2 \rightarrow \{0,1\}^n$ - maps an element of group G_2 to a string of n bits, where n is the bit-length of the plaintext message m.

- Master key $M.K_{Bob} \in Z^*$ - this master key can be a random number. It is used to generate the public key and the private key and must be kept secret.

- Point $P_{Bob} = [M.K_{Bob}]P$ - Bob's public key, generated by the PKG.

- Point $Q_{Bob} = H_1(ID_{Bob})$ - an element in group G_1, calculated using ID_{Bob} and hash function H_1.

The identity number of each entity is public (ID_{Alice}, ID_{Bob}). Thus, any party can generate Q_{Bob} of entity Bob using hash function H_1. Note that, in this system, the master key $M.K_{Bob}$ chosen by PKG is used to generate both the public key and the private key.

Key Generation Bob's private key $K_{Bob} = [M.K_{Bob}]Q_{Bob} = [M.K_{Bob}]H_1(ID_{Bob})$ is also generated by PKG, but should be kept secret and given to Bob through some secure channel.

Encryption When Alice wants to send Bob an n-bit plaintext message m, she first computes $Q_{Bob} = H_1(ID_{Bob})$ with Bob's identity information ID_{Bob} and public system parameter H_1. Alice picks a random integer $r \in Z^*$ and encrypts m into a ciphertext pair $\{P_r, c\}$:

$$\{P_r, c\}, \text{ where } \begin{cases} P_r = [r]P \in G_1 \\ c = m \oplus H_2(e(Q_{Bob}; P_{Bob})^r) \in \{0,1\}^n \end{cases} \tag{2.14}$$

This ciphertext pair is then transmitted to Bob, through an insecure channel.

Decryption On receiving the ciphertext, Bob decrypts the message as:

$$m' = c \oplus H_2(e(K_{Bob}; P_r)) \tag{2.15}$$

Expanding all elements in equation 2.15, Bob gets:

$$M' = m \oplus H_2(e(Q_{Bob}; [M.K_{Bob}]P)^r) \oplus H_2(e([M.K_{Bob}]Q_{Bob}; [r]P)) \tag{2.16}$$

Note that the bilinearity property of pairings ensures:

$$e(Q_{Bob}; [M.K_{Bob}]P)^r = e([M.K_{Bob}]Q_{Bob}; [r]P$$

The hash function H_2 is published by the PKG. The value $H_2(e([M.K_{Bob}]Q_{Bob}; [r]P))$ computed by Bob is the same as that computed by Alice. Thus, with the correct private key K_{Bob}, Bob can decrypt the plaintext message m from Alice.

Compared to the original IBE scheme illustrated in Fig. 2.3, the pairing based IBE scheme in Fig. 2.7 employs the pairing algorithm as the specific encryption and decryption algorithm. Moreover, for calculation convenience, two cryptographic hash functions, H_1 and H_2, are used to change the user's identity information into strings of suitable length.

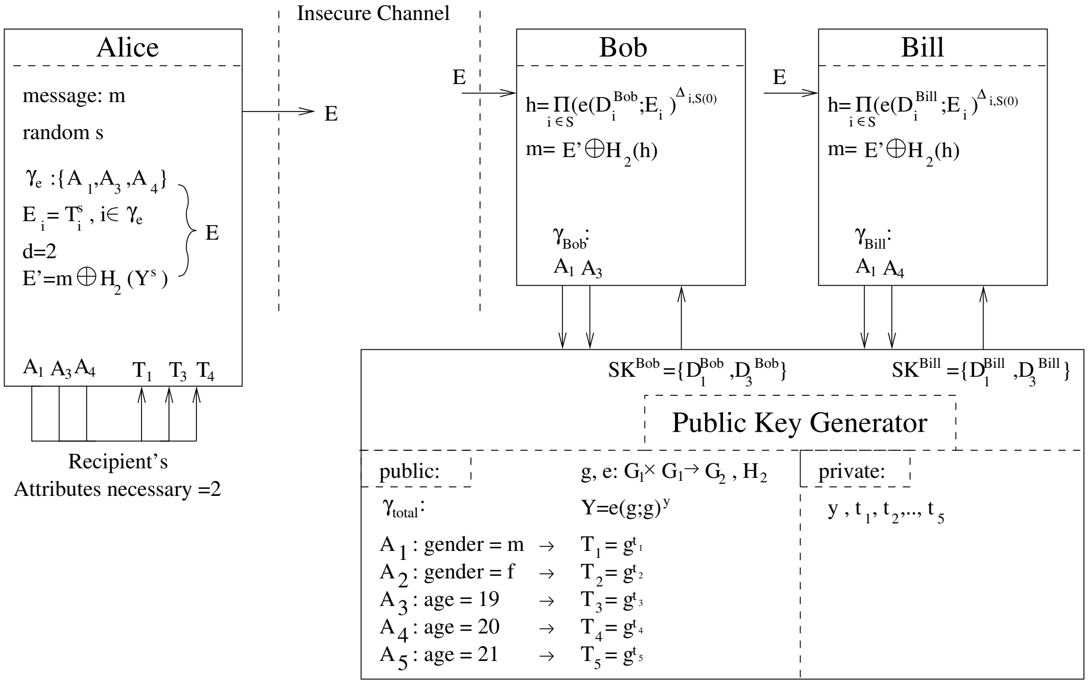


Fig. 2.8: Pairing based ABE scheme

An eavesdropper Eva may hold the public system parameters published by PKG and the ciphertext transmitted in the insecure channel. However, to solve for m, Eva needs to solve $e(Q_{Bob};[M.K_{Bob}]P)^r$ or $e([M.K_{Bob}]Q_{Bob};[r]P)$ from the system parameter set $< G_1, G_2, e, P, H_1, H_2, P_{Bob}, Q_{Bob} >$ and $[r]P$. This task is related to solving an ECDLP [4].

2.5.4 Bilinear Pairings applied in ABE

ABE scheme of Fig. 2.3 can be implemented as illustrated in Fig. 2.8 using bilinear pairings [6].

The construction of this ABE system follows:

Setup This algorithm initiates:

- Select a bilinear group G_1 of prime order p, with bilinear map $e : G_1 \times G_1 \to G_2$.

- Let $\mathbb{Z}_p$ represent the set of positive integers. Define the Lagrange coefficient $\Delta_{i,S}$ for $i \in \mathbb{Z}_p$ and a set, S, of elements in $\mathbb{Z}_p$:

$$\Delta_{i,S}(x) = \prod_{j \in S, j \neq i} \frac{x-j}{i-j}.$$

The polynomial interpolation operation of this Lagrange polynomial can be used to simplify the exponent in the decryption step.

- Let γ_{total} represent the set of all attributes. Define the universe, U, of N elements. For simplicity, take the first N elements of $\mathbb{Z}_p$ to be the universe, i.e. the integers 1,...,$N(\text{mod } p)$. Each integer in U associates with an attribute in γ_{total}.

- Choose random elements $y, t_1, ..., t_N$ in $\mathbb{Z}_p$ and a random generator g of G_1.

- Hash function $H_2 : G_2 \to \{0,1\}^n$ - maps an element of group G_2 to a string of n bits, where n is the bit-length of the plaintext message m.

To set up this scheme, the PKG computes: $T_1 = g^{t_1}, ..., T_N = g^{t_N}, Y = e(g,g)^y$. The public key is $PK =< e, g, Y, T_1, ..., T_N >$. The master key is $MK =< y, t_1, ..., t_N >$.
Encryption Encryptor processes the message m with the public identity γ_e as follows. Pick a random value $s \in \mathbb{Z}_p$, then publish the ciphertext as:

$$E = (d, \gamma_e, E' = m \oplus H_2(Y^s), \{E_i = T_i^s\}_{i \in \gamma_e})$$

Note that the identity, γ_e, is included in the ciphertext.

In the case of this ABE system, Alice has $\gamma_e = \{A_1, A_3, A_4\}$ and calculates $E_i = \{E_1, E_3, E_4\} = \{T_1^s, T_3^s, T_4^s\}$. Alice sets the attribute requirement number $d = 2$ and publishes $(d, \gamma_e, E', \{E_1, E_3, E_4\})$ to the insecure channel.
Key Generation To generate a private key for identity $\gamma_1 \subseteq U$, the following steps are taken by the PKG. A $d-1$ degree polynomial q is randomly chosen such that $q(0) = y$. Compute $D_i = g^{\frac{q(i)}{t_i}}$ for every $i \in \gamma_1$. The private key $SK = (D_i)_{i \in \gamma_1}$.
Decryption Suppose that a ciphertext, E, is encrypted with a key for identity γ_e and the decryptor has a private key for identity γ_1, where $|\gamma_e \cap \gamma_1| \geq d$. Choose an

arbitrary d-element subset, S, of $\gamma_e \cap \gamma_1$, then the ciphertext can be decrypted as:

$$\begin{aligned}
&E' \oplus (\prod_{i\in S}(e(D_i, E_i))^{\Delta_{i,S}(0)}) \\
&= m \oplus H_2(e(g,g)^{sy}) \oplus H_2((\prod_{i\in S}(e(g^{\frac{q(i)}{t_i}}, g^{st_i}))^{\Delta_{i,S}(0)})) \\
&= m \oplus H_2(e(g,g)^{sy}) \oplus H_2((\prod_{i\in S}(e(g,g)^{sq(i)})^{\Delta_{i,S}(0)})) \\
&= m \oplus H_2(e(g,g)^{sy}) \oplus H_2((e(g,g)^{s\sum\limits_{i\in S} q(i)\Delta_{i,S}(0)})) && \text{(1)bilinearity} \\
&= m \oplus H_2(e(g,g)^{sy}) \oplus H_2(e(g,g)^{sq(0)}) && \text{(2)polynomial interpolation} \\
&= m.
\end{aligned} \tag{2.17}$$

In equality (1) of equation 2.17, the bilinearity property of the pairing calculation transforms the successive multiplication into the exponent of a sum. Equality (2) is derived using polynomial interpolation in the exponents. Since the polynomial $sq(x)$ is of degree $d-1$, it can be interpolated using d points.

In the case of this ABE system, Bob holds attribute set $\gamma_1 = \gamma_{Bob} = \{A_1, A_3\}$. Bob asks the PKG for the corresponding secret set. PKG confirms Bob's identity and generates a polynomial q^{Bob} specifically and calculates and sends Bob $\{D_1^{Bob}, D_3^{Bob}\}$ accordingly. Bob picks E_1 and E_3 from the information published by Alice and calculates equation 2.17 and gets plaintext m.

Note that in the original ABE scheme introduced by Sahai and Waters in [6], cryptographic hash functions are not used. Adding hash functions improves the flexibility of ABE schemes, but on the other hand, makes the security of the entire scheme rest not only on the security of the pairing based ABE scheme but, also on the security of the cryptographic hash function used.

2.5.5 *Security of Pairing based cryptosystem*

In addition to brute force attack, researchers have developed various attacks to solve the mathematical problems in elliptic curve based cryptosystems [96, 92, 93]. However, none of these methods would be effective for a carefully chosen elliptic curve [51]. The ECDLP and other equivalent problems provide very good security. The field sizes of Elliptic Curve cryptography over $GF(2^m)$ and the equivalent security levels of symmetric key cryptography are listed in Table 2.1.

In Table 2.1, the required key length for a symmetric-key cryptosystem of equivalent security is chosen as a standard [93]. For security at 80 bit level, Iyama et al has proved that the η_T pairing algorithm provides the fastest calculation [86] amongst

Symmetric key size	η_T over field
66	$GF(2^{233})$
72	$GF(2^{283})$
80	$GF(2^{409})$
88	$GF(2^{457})$
96	$GF(2^{571})$
128	$GF(2^{1223})$

Tab. 2.1: Equivalent security level of η_T pairing algorithm

RSA [23], ECC over non-supersingular [81] and other pairing algorithms. For higher security level, for example 128 bit, η_T pairing is considered inefficient because the field size it requires is too large and the hardware resources and time required for its calculation no longer outperform the other cryptographic schemes [87].

The security of the pairing based cryptographies relies on how hard it is to solve the Bilinear Diffie-Hellman Problem (BDHP) [4]. This problem is considered as hard as an ECDLP because the pairing computation part of the BDHP can be done in polynomial time. The detail of BDHP is described here. In the l-torsion group of points $< P >$ the designer picked which is generated by $P \in E(GF(2^m))[l]$, given P, $[a]P$, $[b]P$,$[c]P \in E(GF(2^m))[l]$, solve $e(P;P)^{abc} \in E(GF(2^{4m}))*$ is a BDHP in this case. An attacker may try to solve a from P and $[a]P$ and then compute $[ab]P$ followed by $e([ab]P;[c]P) = e(P;P)^{abc}$.

Certicom, the ECC cryptography provider, stated that ECDLP over finite fields of size larger than 163-bit is believed computationally infeasible [75]. Thus, $m = 163$ will be the minimum size of a pairing based cryptography to implement. Different group order l gives different security level of ECDLP. Recommended by NIST [23], a pairing based cryptosystem should have a security level equivalent to an 80-bit symmetric key encryption. Also, Koblitz and Menezes [88] suggested that under this security level, a group $E(GF(2^m))[l]$ helps prevent the Pollard-ρ attack [92] on the DLP in the group.

Another factor that affects the system security is the size of the finite field $GF(2^m)$ on which the elliptic curve $E(GF(2^m))[l]$ is picked, i.e. m affects the security. The extension field $GF(2^{4m})$ is always expected to be large enough to prevent an index calculus [89] attack on the DLP in the field. As stated by Thome in [76], under MOV reduction, for an extended field $GF(2^{4m})$, the discrete logarithm is considered very hard when $4m$ exceeds 1000. For security equivalent to, or better than, 1024 bit RSA, $4m$ should go above 1200. Furthermore, an odd m or, more strongly, a prime

m, can make sure that the DLP on the chosen elliptic curve cannot be reduced [90]. Thus, for a security cryptosystem, the field size m should be chosen to be a prime larger than 250. For a cryptosystem of better security than 80-bit symmetric key encryption, m should be a prime larger than 300.

Thus, in this thesis, the basic field size $m = 163$ is used, with results up to $m = 571$ to show the structure and implementation of the pairing algorithm.

2.6 Cryptanalysis

Cryptanalysis is the study of methods to reveal the concealed message in a ciphertext without knowing the secret key an intended receiver used to decrypt it. For any form of ciphertext, the simplest method of attack is the brute force attack, also known as exhaustive key search. To perform such an attack, the attacker tries every possible value of the secret key to decrypt the ciphertext until the plaintext is revealed. In attacking a pairing computation over an elliptic curve $E(GF(2^m))$, where $m > 160$, this method becomes computationally infeasible.

2.6.1 Side Channel Analysis Attacks

Mathematically, choosing a large finite field provides good security for a cryptographic scheme. However, in implementing a cryptosystem, the hardware may show physical weakness that an attacker can make use of. Side channel analysis (SCA) attacks make use of the leaked information from a hardware device in which a cryptographic algorithm is operating. The attacker needs physical access to the hardware device. Any side channel information that can be measured, such as power consumption [13], timing information [14] and electro-magnetic radiation [15], can be used in such SCA attacks. These can be performed on different types of hardware devices such as FPGA [93, 16, 94] and smartcards [95]. This thesis focuses on power analysis attacks against FPGAs.

Power analysis attacks can be divided into simple power analysis (SPA), differential power analysis (DPA) and correlation power analysis (CPA). Several SCA attacks have been proposed against hardware implemented cryptographic algorithms. In 1999, Kocher et al. [18] successfully performed a SPA attack and a DPA attack on a DES implementation and revealed the secret key. In 2005, Page and Vercauteren [16] presented a theoretical DPA attack against Duursma and Lee's algorithm [26] for characteristic three. In 2004, Brier et al. [96] performed a CPA attack against FPGA implementations of DES and AES.

Here it is assumed that an attacker has access to a hardware device in which a cryptographic algorithm is operating and the power traces can be measured by the attacker. The SPA attacks determine what operations are executed by the comparing the profiles and shapes of the measured power traces. This attack can be prevented by introducing redundant operations and making the operations always the same for any secret key. For elliptic curves, several algorithms [97, 98, 99] have been proposed as countermeasures to SPA.

More advanced DPA and CPA attacks apply statistical analysis to the measured power traces to determine the secret key. DPA attack makes use of the difference of mean values of different sets of power traces [18]. CPA takes advantage of the linear correlation between hamming distance and power consumption [96]. In this work, statistical analysis using a CPA attack method is assumed.

2.6.2 Hardware Power consumption

To analyze the relationship between the intermediate variables operated upon in the FPGA and the power consumption of the related operations, the correct power consumption model that links the two aspects must be considered.

Complementary metal-oxide-semiconductor (CMOS) technology is widely used in modern ICs. The power consumption of a CMOS circuit consists of two parts, the static part and the dynamic part. CMOS circuits have very low static power consumption.

Take a CMOS inverter as a simple example, shown in Fig. 2.9. When the input is at logic 0, the n-MOS device is OFF and the p-MOS device is ON. The output is at logic 1, connected to supply voltage V_{CC}. On the contrary, when the input is at logic 1, the n-MOS is ON and the p-MOS is OFF. The Output voltage is at logic 0 and connected to GND.

There are no appreciable current flows into the gate terminal and no significant dc current path from V_{CC} to GND. However, a leakage current exists due to reverse-bias leakage between diffused regions and the substrate. Thus, the static power consumption P_S can be calculated as in equation 2.18 [100]:

$$P_S = V_{CC} \times I_{CC} \tag{2.18}$$

where:
V_{CC} = supply voltage
I_{CC} = current into a device (sum of leakage currents)

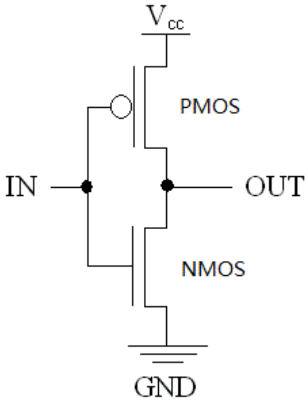


Fig. 2.9: A general CMOS inverter

This power exists in all CMOS components in an FPGA device. When a CMOS device stays static and does not switch, this power contributes to the system noise in power measurement and power analysis.

When a CMOS device switches from one logic state to another, one of the MOS transistors turns on while the other turns off at the same time. During this brief transition, there exists a switching current. This current causes a transient power consumption P_T which dominates the dynamic power consumption of a CMOS device. In addition, when a CMOS device switches, the external capacitance is charged. This causes current and capacitive-load power consumption P_L which adds to the dynamic power consumption as well. The dynamic power consumption P_D can be calculated as in equation 2.19 [100]:

$$P_D = P_T + P_L = (C_{pd} \times N_{SW} + \sum C_L) \times V_{CC}^2 \times f \tag{2.19}$$

where:
C_{pd} = power-consumption capacitance
f = system frequency
V_{CC} = supply voltage
N_{SW} = total number of bits switching.
C_L = load capacitances at each output.
It can be abstracted from equation 2.19 that in the case when there is no output, during a single clock period, the dynamic power consumption of a CMOS IC caused by bits switching can be calculated as in equation 2.20:

$$P_{SW} = C_{pd} \times V_{CC}^2 \times N_{SW} \times f \tag{2.20}$$

Equation 2.21 gives a general idea of the relationship between power consumption of digital circuits and the data being operated upon inside such circuits, i.e.: under certain technology (C_{pd} is constant) and stable power supply (V_{CC} is constant), the power dynamic consumption is linearly related with, N_{SW}, the total number of bits switching on the board.

$$P_{SW} \propto N_{SW} \tag{2.21}$$

The more bits switching at a time, the more power consumed at that time. This is a weak point of practical digital IC design and provides the power analysis attackers with an opportunity to reveal the secrets in hardware cryptosystems.

2.6.3 Weakness in pairing based protocols to side channel attacks.

The pairing based systems were designed for use in public key cryptography, for example the pairing based IBE schemes described in section 2.5.3. Any protocol applied on an insecure channel will have to face a problem: an ill-intentioned attacker may intercept the message and forward the altered message to the intended receiver.

Take the pairing based public key schemes, sender Alice has to send the cypher text and some useful information for decryption, namely a $\{U, V\}$ pair. This $\{U, V\}$ pair can have specific names in different public key schemes and will be addressed in IBE and ABE schemes in detailed discussions in section 5.3. Assume Eva is a side channel attacker who can get physical access to Bob's hardware devices. Thus, Eva can intercept $\{U, V\}$, the public value exposed in the insecure channel and forward the altered values $\{U', V'\}$ to Bob. Also, Eva can monitor and obtain the power traces of Bob's decryption process in the hardware devices. Originally, for decryption, Bob computes $e(K_{Bob}; U)$ when he receives a ciphertext $\{U, V\}$ from Alice. When the cipertext was distorted to $\{U', V'\}$ by Eva, Bob computes $e(K_{Bob}; U')$. This will result in an incorrect plaintext which will be abandoned by Bob's end automatically. However, by doing the same operation a number of times using different $\{U', V'\}$ (usually N times, where $500 \leq N \leq 10000$ [13, 101, 102]), Eva can collect the power consumption traces of Bob's 'no sense' operations. Under this condition, Eva knows the structure of the pairing algorithm e, the value of U' and the power consumption of the N calculations. Eva is then able to perform a power analysis attack trying to reveal the secret operator K_{Bob} according to the above information. The details of how Eva can perform a CPA attack and get Bob's private key K_{Bob} can be found in Chapter 5.

2.6.4 Countermeasures against power analysis attacks

Because SCA attacks can be applied on hardware implementations of cryptosystems to reveal the secret information, designers of the cryptographic algorithms have put forward some countermeasures. The weakness mentioned above exists because in the operation $e(K_{Bob}; U')$, operand U' is known by attacker Eva. Designers should try to make it impossible for Eva to use the knowledge she holds to perform an SCA attack.

Several countermeasures to SCA attacks have been suggested for affected cryptosystems. [16, 17, 18, 103, 104, 105] The architecture level countermeasures can be implemented by noise insertion [18], random clock frequency [103], or randomization of the instruction streams [104]. Although these methods succeed in changing the implementation's power consumption pattern, it is still possible for an attacker to reconstruct the original pattern [18, 105].

The algorithm level countermeasures operate on the intermediate variables in the calculation flow. Page and Vercauteren [16] pointed out that the bilinearity property of the pairing algorithms can be used. Scott [17] proposed a countermeasure which multiplies a randomly picked variable with the input point to mask it. Coron [18] pointed out that using projective coordinate rather than the conventional affine coordinates helps hide the secret information. However, projective coordinate solutions to the η_T pairing algorithm requires additional operations. Thus, in implementing the original η_T pairing algorithm, such solutions are not included. Instead, these methods are considered as countermeasures against SCAs to help improve the security of the η_T pairing and their implementations are investigated in Chapter 6.

As the algorithm level countermeasures all introduce additional operations based on the original algorithm, area and calculation time overheads follow. The extra security should not come at the expense of excessively increasing the hardware resources required. Therefore, apart from the enhanced security aspect, the overhead cost of the countermeasures must be considered.

2.7 Hardware Accelerators

As can be seen in the pairing based IBE scheme and ABE scheme in section 2.5.3, there are pairing operations in the encryption step and corresponding pairing operations in the decryption step. This makes the cryptosystem very computationally intensive because a pairing requires many finite field operations. Such schemes can be applied by implementing software on a general purpose processor. However, general purpose processors perform operations serially. This does not suit the finite field operations required in pairings. Moreover, a general purpose processor processes a word length of 32 or 64 bits, while the word length of an element in the pairing is suggested to be at least 160 bits [23].

A dedicated hardware accelerator can easily solve the problems a general purpose processor meets [37]. The word length of a dedicated hardware accelerator is assigned by the designer. Dedicated hardware accelerators can be designed to exploit the parallelism and improve the system performance substantially. By carefully arranging the hardware, techniques such as loop unrolling and pipelining can help speed up the operations in a pairing calculation. Take the η_T pairing algorithm as an example. The η_T design over the finite field $GF(2^{307})$ on a Pentium IV processor with an operating frequency of 3000 MHz computes the algorithm in 3500 μs [106], while a similar design on a Xilinx FPGA Virtex-II Pro with an operating frequency 156 MHz computes the pairing algorithm over $GF(2^{313})$ within only 213 μs [107]. The hardware processor accelerates the calculation by 16 times which is a significant improvement.

2.7.1 Field Programmable Gate Arrays

A Field Programmable Gate Array (FPGA) is an integrated circuit that can be programmed and reconfigured by the customers to carry out some specific logic functions. What makes the FPGA ‘field programmable’ are the individual Configurable Logic Blocks (CLBs) and a hierarchy of reconfigurable interconnects that allow the CLBs to be ‘wired together’. The function of the blocks can be specified by the customers using a Hardware Description Language (HDL). Software synthesis tools allow a designer to take a high level design description and transform it into a programming file for the FPGA.

The leading FPGA manufacturers are Xilinx [108] and Altera [109]. In this work, the designs are implemented on Xilinx Virtex-V [84] FPGAs. An FPGA consists of the following components:

- Input/Output Blocks (IOBs): the simplest block that provides interconnection between the FPGA and other devices on the board. Each IOB supports bidirectional data flow and 3-state operation.

- Configurable Logic Blocks (CLBs): basic functional part of an FPGA which contains flexible look-up tables (LUTs) that can implement logic. Each CLB

Model	IOBs	CLBs(row × col)	Slices	Registers	LUTs	BRAM(Kbits)
xc5vlx50	560	120×30	7,200	28,800	28,800	1,728
xc3s400A	311	40×24	3,584	7,168	7,168	360

Tab. 2.2: Xilinx Virtex-V xc5vlx50 and Spartan-3A xc3s400A product specifications

contains several interconnected slices. Within each slice, there are LUTs and flip flops (FFs).

- Block RAM: provides data storage in the form of dual-port blocks. These block RAMs can be used to synchronously store large amounts of data.

In the design of dedicated hardware accelerators, a reduction in computation time is always the aim. In the meantime, using less circuit area is also a goal. To speed up the calculation of a hardware accelerator, new technology can bring higher operation frequency and faster calculation speed. Other than using different technologies, a designer can minimize the number of clock cycles required using appropriately optimized algorithms. Also, using more hardware resources to increase parallelism can decrease calculation time, at the cost of greater resources used. When implementing an algorithm on a hardware accelerator, usually there are many different methods of parallelizing the operations, resulting in different operation times and circuit areas. Typically, there exists a trade-off between area and speed of a hardware design.

2.7.2 SASEBO-GII board

The SASEBO-GII evaluation board [110] was designed and developed specifically for side-channel attack experiments. The board has two Xilinx FPGAs: the Virtex-5 xc5vlx50 as a cryptographic FPGA and a Spartan-3A XC3S400A [111] as a control FPGA. These two FPGAs are connected using a 38-bit general purpose input/output common bus. In this work, the hardware accelerators are mounted on the Virtex-5 FPGA. The Spartan-3A acts as a connector between the Virtex-5 FPGA and the PC. The main specifications of the FPGAs are shown in Table 2.2. There is a shunt resistor of only 1Ω inserted between the V_{DD} side of the cryptographic FPGA and the power supply for measuring power traces. Fig. 2.10 shows a block diagram of the SASEBO-GII board and the experimental setup.

In power analysis experiments, the control FPGA communicates with the host PC through a USB connector. The cryptosystem is implemented on the crypto FPGA. All input and output operations between the host computer and the crypto FPGA are transferred through the control FPGA. This prevents direct communications

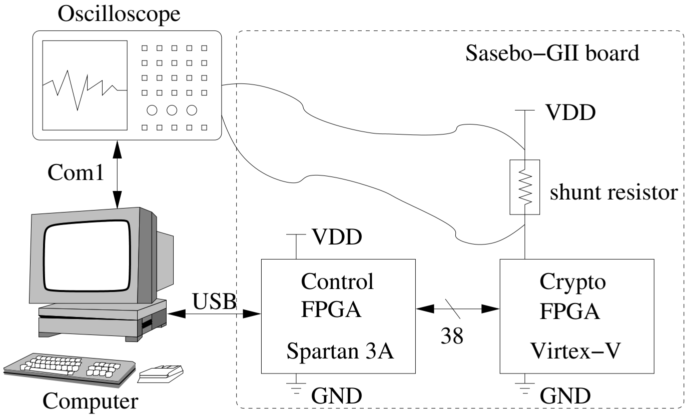


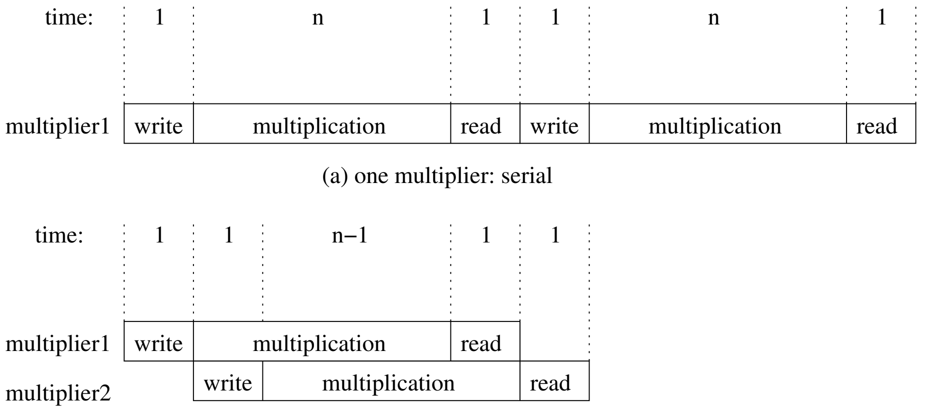
Fig. 2.10: SASEBO-GII board block diagram

between the host computer and the crypto FPGA (which is always the target of the power analysis attack) and reduces the environmental noise in the power analysis attacks.

The board has an external power source which supplies the on-board power regulators and the FPGAs with 5.0V. The on-board oscillator provides a clock signal of 24MHz and there is a port which supports external clock input of different frequencies. Different operations in the cryptographic algorithm lead to different numbers of transistors switching. According to equation 2.20 the difference in the number of switching transistors can cause different dynamic power consumption values. A differential probe of the oscilloscope, connected to the two ends of the shunt resistor, records the voltage difference values. These values are linearly related to the dynamic current through the shunt resistor and can be used for a power analysis of the cryptosystem.

2.7.3 Efficiency Evaluation of Hardware Designs

The most common aspect of a hardware accelerator of concern to the designer is the computation time or speed. Generally, this can be calculated using the total number of clock cycles needed per computation and the maximum clock frequency. The calculation flow of a cryptographic algorithm is decided by the algorithm mathematically. Once an algorithm is chosen, the computation flow is fixed. The designer's work is to minimize the clock cycles required for each sub-operation and to optimize the hardware architecture for a higher operating frequency. To minimize the clock cycles required, a popular method is to make operations run in parallel with more hardware resources. For example, two multiplications, where each calculates the re-



(a) one multiplier: serial

(b) two multipliers: parallel

Fig. 2.11: Time scheduling for different number of multipliers

sult in n clock cycles, can be operated in $2n+4$ clock cycles using only one multiplier, as shown in Fig. 2.11. Using a parallel method, by adding one more multiplier and some control units in hardware, the multiplication can be calculated in $n + 3$ clock cycles. This method reduced the calculation time by $n + 1$ clock cycles. Another useful method to speed up a hardware accelerator is to shorten the critical path.

In FPGA design, a critical path indicates the longest path a signal encounters from one register to another. This usually happens where complicated logic cells are required. A popular way to optimize such a case is called pipelining and is achieved by inserting registers between the logic cells and split the original 'one clock cycle operation' into two clock cycles. Fig. 2.12 shows the logical path of a calculation $x = (a \times b) + (c \times d)$. The longest path here is from a (Flip-Flop 1) to x (Flip-Flop 5), i.e. route1 + gate1 + route2 + gate2 + route3. By adding intermediate registers, the longest path in this block should be either route4 + gate1 + route5 or route6 + gate2 + route7. This method does not precisely half the original critical path, but shortens it. The operation $x = (a \times b) + (c \times d)$ is then divided into two stages, the '$\times$' stage and the '$+$' stage and requires 2 clock cycles for calculation. This can lead to more clock cycles and may require more hardware resource, but will increase the maximum frequency at which the entire system can be operated. However, by applying the pipeline method, when the '$\times$' stage is finished, the system can put the next operation, $x_2 = (a_2 \times b_2) + (c_2 \times d_2)$, into the structure rather than wait until the calculation of x is finished. This pipeline method can save clock cycles and improve the hardware efficiency.

The area of the architecture is also important. For FPGAs the area is usually measured in terms of slices or look-up tables (LUTs). The tristate buffers, block

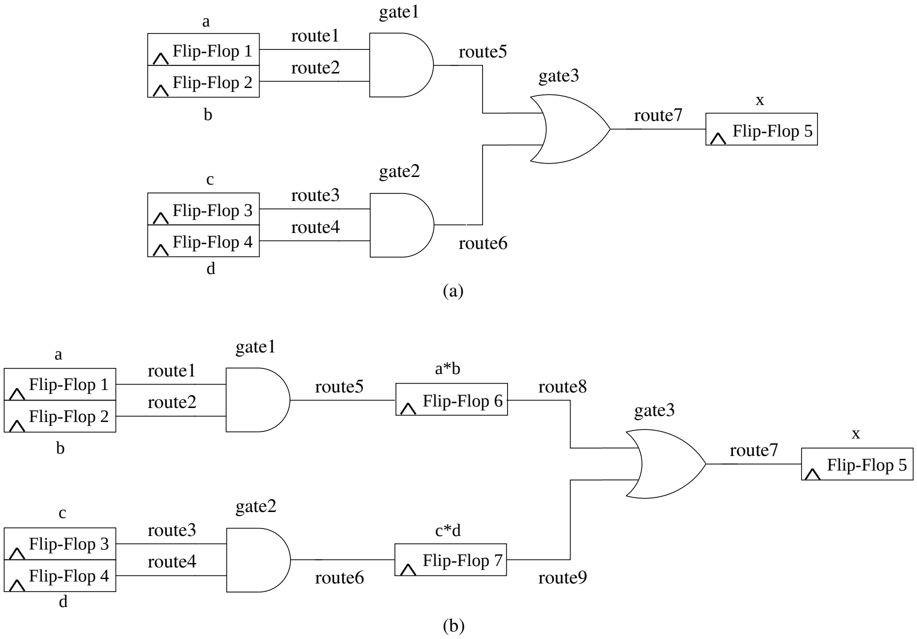


Fig. 2.12: Cutting down the critical path

RAMs and ROMs etc. are also important parts of an FPGA. Basically, the more such resources are required the larger the area an FPGA design takes. Slices are the underlying units in a Xilinx FPGA. Each slice contains a number of LUTs and several flip-flops. For example, in the Xilinx Virtex-V FPGA [84], each slice contains 4 LUTs and 4 flip-flops. Thus, when listing the implementation result, it is important to attach the FPGA type. Comparison between different type of FPGA chips is meaningless.

As can be seen from the methods to optimize the calculation speed, most ways to decrease the calculation time will increase the area taken by the architecture. In such a case a trade-off is desired. In this work the area-time (A*T) product is introduced as a parameter to compare designs. This is the product of the computation time and the required area. Increasing the parallelism in the design decreases the computation time but simultaneously requires more area. The A*T product judges whether the increase in area outweighs the decrease in computation time. The design with the minimum A*T product is said to provide the best trade-off between area and speed.

2.8 Conclusions

In this chapter, the background information required for the rest of the thesis has been provided. The IBE and ABE schemes are two popular public key schemes that satisfy the demands of security nowadays. The IBE scheme provides an approach to peer to peer communication and the ABE scheme provides a peer to multi-peer solution to the morden communication. Elliptic curve cryptography and, in particular, pairing based cryptography can be used in both the IBE and ABE schemes in practical applications. The pairing based cryptography is chosen in this work as an efficient way of implementing the public key schemes. Considering both security and efficiency aspects, implementations of the η_T pairing algorithm over fields of size m between 163 and 571 will be investigated.

SCA attacks are considered in this thesis to investigate and improve the security of the proposed public key cryptosystem. Using cryptanalysis methods to test and find the weaknesses existing in the cryptosystem and applying countermeasures to protect the system is an efficient way to improve the security aspect of the cryptosystems. This methodology is used in this thesis. Both the hardware power consumption model and the usage of the cryptographic algorithm in the public key schemes are considered to find out the possible weaknesses in these schemes.

A SASEBO-GII evaluation board which consists of both the FPGA components and the power measurement ports is used for implementation and SCA attacks in this thesis. The flexibility of FPGAs makes them highly suitable for accelerating cryptographic applications as the processors can easily be reconfigured to support different design parameters. This makes it a good platform for implementing accelerators for the cryptosystems. The shunt resistor provides convenience for cryptanalysis. In this thesis, the power analysis attacks are based on the power collected from this platform.

3. ARITHMETIC ARCHITECTURES OVER THE FIELD $GF(2^M)$

3.1 Introduction

As introduced in Chapter 2, pairings are basically operations on two points over an elliptic curve. Thus, the efficiency of the reconfigurable hardware design of a pairing algorithm depends on the efficiency of the basic operations, including addition, squaring, multiplication and division, over the finite field. Hardware architectures of these arithmetic operations over the field of characteristic 2, i.e. over $GF(2^m)$, are presented in this chapter.

Section 3.2 discusses the targeted finite field size m. The irreducible polynomials that generates such fields are listed. Section 3.3 introduces the basic addition operations over finite fields $GF(2^m)$. Section 3.4 presents some popular $GF(2^m)$ multiplication structures, including the digit-serial multiplier and the Karatsuba multiplier. Section 3.5 presents a parallel structure for implementing the $GF(2^m)$ squaring. In sections 3.6 and 3.7, two methods of implementing the $GF(2^m)$ division/inversion are introduced. For all the $GF(2^m)$ operation blocks, the details of the structures are analyzed and simple examples are illustrated to highlight the calculation flow in the operations.

3.2 Choice of Finite Fields

An element $a(x)$ of the finite field $GF(2^m)$ in this work is represented using a degree $(m-1)$ binary polynomial, $a(x) = \sum\limits_{i=0}^{m-1} a_i x^i, a_i \in \{0,1\}$ which maps well to hardware. There are basic operations of addition, squaring, multiplication and division/inversion over field $GF(2^m)$. If the basic operations generate a result of degree higher than $m-1$, this result must go through a reduction operation, modulo $f(x)$, which reduces the degree down to $m-1$ or lower. This $f(x)$ is the irreducible polynomial of the corresponding finite field, of degree m. The value of $f(x)$ depends on the field size m. For efficient field operations, polynomials of low hamming weight are preferred. Normally an irreducible trinomial $f(x) = x^m + x^k + 1$ will be chosen, if one exists. A trinomial is preferred in the implementations as it requires the least hardware resources. However, if no irreducible trinomial exists, then a pentanomial $f(x) = x^m + x^a + x^b + x^c + 1$ is selected. The particular pentanomial chosen has the following properties: the second term x^a has the lowest degree among all irreducible pentanomials of degree m; the third term x^b has the lowest degree among all irreducible pentanomials of degree m and second term x^a; and the fourth term x^c has the lowest degree among all irreducible pentanomials of degree m, second term x^a and third term x^b [1].

As introduced in chapter 2, the suggested field size m of $GF(2^m)$, where elliptic curves are picked, varies from 163 to 571, as suggested by NIST in [93]. The curves over binary fields are carefully chosen because the coefficients of these specific curves and underlying field were selected to optimize the efficiency of the elliptic curve operations. Here are the polynomials for the field sizes of interested:
$m = 163 : f(x) = x^{163} + x^7 + x^6 + x^3 + 1$
$m = 233 : f(x) = x^{233} + x^{74} + 1$
$m = 283 : f(x) = x^{283} + x^{12} + x^7 + x^5 + 1$
$m = 571 : f(x) = x^{571} + x^{10} + x^5 + x^2 + 1$
Note that the coefficients of the irreducible polynomials selected for the corresponding field sizes all follow the rules introduced in last paragraph. To find an irreducible polynomial, or to test the irreducibility of a polynomial, designers can refer to [112, 113, 114, 115].

3.3 Choice of Basis

In order to perform operations on elements of binary fields and their corresponding extension fields, a basis representation must be defined first. Different bases including polynomial basis [116], normal basis or dual basis [117] are available to represent elements over binary fields.

In polynomial basis, let a be an element over $GF(2^m)$. The polynomial basis of $GF(2^m)$ is:

$$\{1, a, a^2, ..., a^{m-1}\} \tag{3.1}$$

The set of elements over of $GF(2^m)$ in polynomial basis is:

$$\{0, 1, a, a^2, ..., a^{2^m-2}\} \tag{3.2}$$

Note that in binary fields, an irreducible polynomial can be used to reduce for terms of degree higher than m to at most $m-1$. When using binary strings to represent elements in polynomial basis, for example in $GF(2^4)$, the most significant bit represents a^3, the least significant bit represents a^0 i.e. 1. A binary string "1011" over $GF(2^4)$ represents the element "a^3+a+1".

In normal basis, let b be an element over $GF(2^m)$. The normal basis of $GF(2^m)$ is:

$$\{b, b^2, b^{2^2}, ..., b^{2^{m-1}}\} \tag{3.3}$$

Different from polynomial basis, the most significant bit in normal basis represents b^8, the least significant bit represents b. The binary string "1011" over $GF(2^4)$ in normal basis represents the element "b^8+b^2+b".

A dual basis is not a concrete basis like polynomial basis or normal basis. It only provide a way to easily communicate between systems using different bases and convert elements from one basis to another. In this work the operands are distributed by the same PKG. Thus, only one basis is needed in the cryptosystem and dual basis is not considered.

The IEEE standards for public key cryptography [32] recommend polynomial or normal basis. For both types of basis, there are different architectures for field operations including addition, multiplication, squaring and division/inversion. Beth [118] investigated the differences between the three bases mentioned above and proved that they all show some advantages in certain conditions. Normal basis needs less steps in calculating exponentiations, but the advantage is not obvious. When area constraints are considered, polynomial basis shows an advantage against normal basis. Moreover, polynomial basis offers hardware multiplication with lower complexity than normal basis [119]. Thus, polynomial basis is chosen in this thesis.

3.4 Addition over $GF(2^m)$

Addition operation over $GF(2)$ is the same as the logic operation exclusive-or (XOR). Note that in binary finite field operations, as $0+0=0$ and $1+1=0$, there exists $-a=a$ for all $a \in GF(2^m)$ i.e. subtraction equals addition.

As the addition operation can be expressed by a logical XOR operation, in a hardware implementation, a one bit addition is performed by a two input XOR gate, and an addition of two m-bit elements is performed by an array of m two input XOR gates, as shown in Fig. 3.1.

In a hardware implementation, because XOR gates do not require large area, designers may prefer to trade operation time with area. For time efficiency, an m-

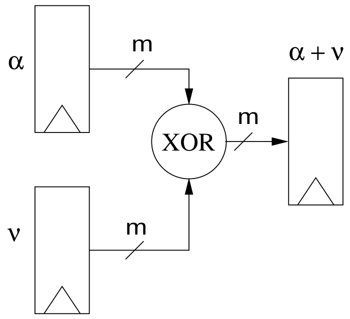


Fig. 3.1: Architecture of an adder

XOR-gate chain can be used to perform an m-bit addition operation in parallel. Typically, in a hardware implementation of an adder, each XOR gate is mapped to a LUT in FPGA. Thus, for field size m, a $GF(2^m)$ Adder requires m LUTs. When the field size m increases, the area taken by the adder increases as well.

The calculation time is equal to the delay through one XOR gate because all m XOR gates operate in parallel. In the Virtex-5 FPGA, 65 nm technology is applied. The time for a signal to propagate through an XOR gate, plus the time for the target register to setup and hold, is 1.249 ns. The maximum operation frequency is 800 MHz. When implementing the $GF(2^m)$ operation blocks, one would find that an Adder takes least area and calculation time. The critical path of a crypto processor with many units will generally not include an adder.

3.5 Multiplication over $GF(2^m)$

Over the basic binary field $GF(2^m)$, a multiplication of two elements $a(x) = \sum_{i=0}^{m-1} a_i x^i$ and $b(x) = \sum_{i=0}^{m-1} b_i x^i$ can be represented as per equation 3.4. The multiplication operation can be divided into two stages: the polynomial multiplication and the reduction stage modulo the irreducible polynomial $f(x)$, i.e. mod $f(x)$ in the following sections.

$$\begin{aligned} & z(x) = a(x) \cdot b(x) = a(x) \cdot \sum_{i=0}^{m-1} b_i x^i \\ & = a(x) b_{m-1} x^{m-1} + \cdots + a(x) b_2 x^2 + a(x) b_1 x + a(x) b_0 = \sum_{i=0}^{2m-2} z_i x^i \\ & c(x) = a(x) \cdot b(x) \bmod f(x) = z(x) \bmod f(x) \end{aligned} \tag{3.4}$$

The maximum degree of the multiplication result $z(x)$ is $(2m-2)$. It must be mapped to an element of $GF(2^m)$ through the reduction operation $\mathrm{mod} f(x)$. The reduction process repeatedly adds the irreducible polynomial and its transforms to $z(x)$ until the result is reduced to degree $(m-1)$ or less.

To illustrate the process of the multiplication operation and how the modulo stage mod $f(x)$ works, consider the field $GF(2^4)$ with irreducible polynomial $f(x) = x^4 + x + 1$. The multiplication of elements $a(x) = x^3 + 1$ and $b(x) = x^3 + x + 1$ is computed. The polynomial multiplication stage is performed as per equation 3.5.

$$\begin{aligned} z(x) &= a(x) \cdot b(x) \\ &= (x^3+1) \cdot (x^3+x+1) \\ &= x^6 + x^3 + x^4 + x + x^3 + 1 \\ &= x^6 + x^4 + x + 1 \end{aligned} \tag{3.5}$$

Note that $x^3 + x^3 = 0$, thus, only 4 terms remain in the result of the degree 6 polynomial $z(x)$. Since $f(x) = 0$ over $GF(2^m)$, terms of degree higher than 3 can be reduced using the transforms of the irreducible polynomial $f(x) = 0$ as per equation 3.6.

$$\begin{aligned} &f(x) = x^4 + x + 1 = 0 \\ &\rightarrow x^4 = x + 1 \\ &\rightarrow x^5 = x^2 + x \\ &\rightarrow x^6 = x^3 + x^2 \end{aligned} \tag{3.6}$$

$$\begin{aligned} c(x) &= z(x) \bmod f(x) \\ &= x^6 + x^4 + x + 1 \bmod f(x) \\ &= \underbrace{(x^3+x^2)}_{x^6} + \underbrace{(x+1)}_{x^4} + x + 1 \\ &= x^3 + x^2 \end{aligned} \tag{3.7}$$

Using these transforms of $f(x)$, $z(x)$ can be reduced to $c(x)$ of degree less than $(m-1)$, as per equation 3.7. This maps the degree 6 element $z(x)$ back to an element over $GF(2^m)$. From equation 3.7 it also can be seen that the reduction of terms x^6 and x^4 can be done at the same time. Thus, in a hardware implementation over FPGA platforms, all terms of degree higher than $(m-1)$ can be reduced in parallel using XOR gates, i.e. addition operations in binary fields, as shown in line 3 of equation 3.7. Also, the reduction result can be output after the delay of the XOR gates in the same clock cycle.

Algorithm 3.1 Right-to-Left Shift-and-add field multiplication over $GF(2^m)$

Input: binary polynomials $a(x) = \sum_{i=0}^{m-1} a_i x^i$ and $b(x) = \sum_{i=0}^{m-1} b_i x^i$

Output: $c(x) = a(x) \cdot b(x) \bmod f(x)$

```
If $b_0 = 1$ then $c \leftarrow a$; else $c \leftarrow 0$;
for $i$ from 0 to $m-1$ loop
    $a = a \cdot x \bmod f(x)$
    if $b_i = 1$ then $c \leftarrow c + a$
end for
return $c(x)$
```

3.5.1 Bit parallel multiplier and Bit serial multiplier

As mentioned in previous sections, the range of field sizes chosen in this work is $163 \leq m \leq 571$. For large finite fields, there are many different architectures suitable for implementing $GF(2^m)$ multiplications.

A bit-parallel multiplier architecture, introduced in [120, 121, 107, 122, 123], performs the multiplication in one clock cycle. It multiplies $a(x)$ by all the bits of $b(x)$ in the same clock cycle and produces a $(2m-2)$ degree polynomial $z(x)$, which then goes through the reduction block modulo $f(x)$ and transforms into the $(m-1)$ degree polynomial $c(x)$, namely the product. This bit-parallel multiplier produces the product in only one clock cycle. It is a very fast method, but requires large circuit area to perform the calculation of $a(x) \cdot b(x) = z(x)$ in one clock cycle for field sizes required in this thesis. For an m-bit multiplication, the number of gates required for this architecture is proportional to m^2. This bit-parallel architecture is not suitable for use in area constrained applications because of the its large area requirement.

A smaller sized multiplier, the original Right-to-left shift-and-add method [52] for field multiplication, is described as in Algorithm 3.1. This method was mentioned in [124, 120, 125] and was named Bit Serial Multiplication.

This algorithm takes as inputs $a(x)$ and $b(x)$ and returns the product $c(x)$ modulo the irreducible polynomial $f(x)$. This architecture is based on the observation of equation 3.4 that $a(x)$ is multiplied by each bit of $b(x)$. As can be seen in Algorithm 3.1, element $a(x)$ is shifted one bit in every iteration. It takes $(m-1)$ iterations to complete this multiplication. A reduction modulo $f(x)$ is performed in each of the iterations to make sure that $a(x)$ is of, at most, degree $(m-1)$. In the hardware architecture of Algorithm 3.1, as the Hamming weight of $f(x)$ is always small, only at most 5 XOR gates are needed to complete the process of the reduction operation.

				⟵	
B[0]	b(3)	b(2)	b(1)	b(0)	↓
B[1]	b(7)	b(6)	b(5)	b(4)	
B[2]	b(11)	b(10)	b(9)	b(8)	
⋮	⋮	⋮	⋮	⋮	
B[40]	0	b(162)	b(161)	b(160)	

Tab. 3.1: 163 bit element $b(x)$ divided by digit $d = 4$

Thus, the bit-serial multiplier requires only a small amount of hardware resource. They are suited for use in area constrained environments where long calculation time is acceptable in order to minimize area.

3.5.2 Digit serial multiplier

Other than the two extremes of finite field multipliers, a digit-serial architecture for calculating $GF(2^m)$ multiplications is proposed by Song and Parhi in [126]. It provides a trade-off between the min-area max-time and the max-area min-time designs. In this digit-serial multiplier (DSM), element $b(x)$ divided into several blocks, each is of d bits, where d is named the digit size of the multiplier. Take $m = 163$ as an example, if $d = 4$, element $b(x)$ is then divided into 41 blocks (B[0]-B[40]), as shown in Table 3.1. Note that element $b(x)$ is divided into $n = \lceil \frac{m}{d} \rceil$ blocks. For the last block B[40], the empty bits of B-blocks are filled with '0'.

In the DSM, element $a(x)$ is multiplied by B[i], i.e. d bits of element $b(x)$ from right to left, in clock cycle i. It takes $n = \lceil \frac{m}{d} \rceil$ clock cycles to finish the calculation. Choosing a larger digit size d requires more hardware resources but, at the same time, reduces the calculation time required. Larger d brings smaller n. This is the trade-off between area and speed. The bit-serial and the bit-parallel, multipliers can be seen as two extreme conditions of the digit-serial multiplier (DSM): $d = 1$ is equivalent to a bit-serial multiplier whilst $d = m$ is equivalent to a bit-parallel multiplier.

The digit-serial multiplication algorithm is given in Algorithm 3.2. Similar to Algorithm 3.1, this algorithm takes as input $a(x)$ and $b(x)$ and computes the product $c(x)$. As mentioned in Table 3.1, if the field size m is not an integer multiple of the digit size d, it is necessary to fill the extra bits with '0' and form a new element $b'(x)$ of m' bits, as shown in definition 1 of Algorithm 3.2. Then this new element $b'(x)$ is divided into n B-blocks (B[0]∼B[n-1]), each B-block is of d bits.

In Algorithm 3.2, $c'(x)$ is initiated as $0 \in GF(2^m)$, followed by $n = \lceil \frac{m}{d} \rceil$ iterations.

Algorithm 3.2 Digit-Serial multiplication over $GF(2^m)$

Input: binary polynomials $a(x) = \sum_{i=0}^{m-1} a_i x^i$ and $b(x) = \sum_{i=0}^{m-1} b_i x^i$

Output: $c(x) = a(x) \cdot b(x) \bmod f(x)$

Define1: $m' = m + (m \bmod d), b'(x) = \sum_{i=0}^{m'-1} b'_i x^i, b'_i = \begin{cases} 0, i > m-1 \\ b_i, i \leq m-1 \end{cases}$

Define2: $n = \lceil \frac{m}{d} \rceil$, $B[i] = \sum_{k=0}^{d-1} b'(i * d + k)$

```
1: $c'(x) \leftarrow 0$
2: for $i$ from $n-1$ downto 0 loop
3:   $c(x) \leftarrow c'(x) + a(x) \cdot B[i] \bmod f(x)$
4:   if $i > 0$ then
5:     $c'(x) \leftarrow c(x) \cdot x^d \bmod f(x)$
6:   end if
7: end for
8: return $c(x)$
```

Within each of the iterations i of the 'for' loop in steps 2-7, the element $a(x)$ is multiplied by a B-block . The products of $a(x)$ and B-blocks are degree $(m+d-2)$ polynomials. These polynomials then go through a reduction block modulo $f(x)$ and return as a degree $(m-1)$ polynomial $c(x)$. Except for the last iteration, this $c(x)$ is then extended by appending d zero bits from the least-significant end and becomes a degree $(m+d-1)$ polynomial $(c(x) \cdot x^d)$, i.e. in step-5,

$$c'(x) = c(x) \cdot x^d \bmod f(x) = \sum_{i=0}^{m+d-1} c'_i x^i \bmod f(x), c'_i = \begin{cases} 0, c < d \\ c_i, i \leq d \end{cases}$$

Another reduction is applied here to reduce $c'(x)$ to a degree $(m-1)$ polynomial. Note that this reduction is different from the one in step-3, which is a $(m+d-2)$ to $(m-1)$ reduction. After n iterations each B-block is processed and the result is returned in $c(x)$.

To illustrate the mechanism of Algorithm 3.2, consider again the example in equation 3.5, the calculation of $a(x) \cdot b(x) = (x^3+1) \cdot (x^3+x+1)$ over $GF(2^4)$ with $f(x) = x^4 + x + 1$. Assume $d = 2$ and $n = 2$. As $m = 4$ and $(m \bmod d = 0)$, there exists $b'(x) = b(x)$. $b'(x)$ here can be divided into 2 B-blocks and each of them contains 2 bits: $B[1] = x + 0$, $B[0] = x + 1$.

Table 3.2 shows the calculation flow and the values of the multiplication over $GF(2^4)$. After 2 iterations, the correct result $(x^3 + x^2)$ is contained in $c(x)$. In a hardware implementation, the corresponding architecture of Algorithm 3.2 is shown

$a(x)$	$x^3 + 1$	
$b(x)$	$x^3 + x + 1$	
$b'(x)$	$x^3 + x + 1$	
$B[1]$	x	
$B[0]$	$x + 1$	
Iteration number i	$i = 1$	$i = 0$
$a(x) \cdot B[i]$	$x^4 + x$	$x^4 + x^3 + x + 1$
$a(x) \cdot B[i] \bmod f(x)$	1	x^3
$c(x) \leftarrow c'(x) + a(x) \cdot B[i] \bmod f(x)$	1	$x^3 + x^2$
$c(x) \cdot x^d$	x^2	n/a
$c(x) \leftarrow c(x) \cdot x^d \bmod f(x)$	x^2	n/a

Tab. 3.2: Calculation flow of Digit-Serial Multiplication example over $GF(2^4)$

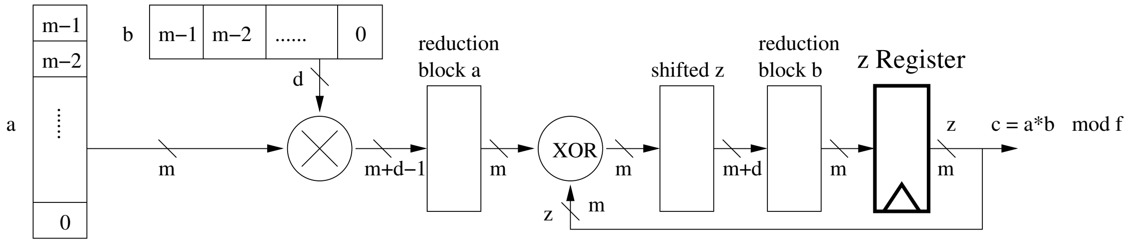


Fig. 3.2: Architecture of a Digit-Serial multiplier

in Fig. 3.2.

In Fig. 3.2, $\otimes$ represents a multiplication block over a binary field, takes as inputs the m-bit element a and d-bits of operand b and returns $c(x)$ consisting of $(m+d-1)$ bits. The multiplication block requires $m * d$ AND gates and $(m-1)*(d-1)$ XOR gates in total, no matter how the field polynomial is picked. The two extreme cases, the bit-serial ($d = 1$) and the bit-parallel ($d = m$), require m AND gates and m^2 AND gates + $(m-1)^2$ XOR gates respectively. Note that in a real bit-serial multiplier, unlike in the digit-serial structure, the multiplication block is not necessary and a multiplexor is applied instead.

This $c(x)$ output from the multiplication block then goes through the first reduction block and is reduced to m bits. The reduction blocks are XOR networks based on the irreducible polynomial $f(x)$. For the input polynomial, each term of degree higher than $(m-1)$ is mapped to 3 (for trinomial) or 5 (for pentanomial) lower degree terms and XORed with the lower bits of the input polynomial. Thus, the hardware resources taken by the reduction blocks depend on the field size and the digit size.

The result output from 'reduction block a' is then accumulated with previous result from the last iteration through an XOR chain, where an array of m XOR gates are used. A finite state machine controls the execution of Algorithm 3.2, the architecture being illustrated in Fig. 3.2. Except for the last iteration, the output of the XOR chain is extended by d '0' bits from the least significant side and sent through another reduction block, the 'reduction block b', which reduces the extended result back to degree of, at most, $(m-1)$. The multiplication result is stored in an m-bit register and is accumulated with the output of 'reduction block a' in every iteration. After n clock cycles, the calculation is finished and the correct result is stored in the 'z Register'. The multiplier then sets a 'done' signal to inform the system controller that the calculation is completed. For different field size and digit size pairs, the calculation time varies. Thus, in hardware designs where this 'done' signal is applied, the system always takes the 'done' signal as permission to read the result from the 'z Register'.

Theoretically the digit size d can be chosen between 1 and m. It is obvious that when using larger digit size d, the number of clock cycles required for the DSM is reduced. In [52] digit size d=4 and 32 are discussed for DSM calculation over $GF(2^{163})$. In [127] digit size d ranging from 4 to 16 was discussed for its area and timing performance. In addition, in different Tate pairing implementations using DSMs [8, 7, 12, 128], digit sizes of d = 1, 2, 4, 8, 16 and 32 were used for different field sizes. To show the trends of how the calculation performances of the DSMs change when digit size d changes, in this work the digit sizes d = 1, 2, 4, 8, 16 and 32 are all applied.

The implementation results of DSM with different field size m and digital size d are shown in Table 3.3. As can be seen, for the same field size m, the DSMs require almost the same number of registers, which is about 3 times the field size m. However, the size of the multiplication block $a \times B[i]$ is determined by both the field size and the digit size. Along with the increase of the digit size d, the number of LUTs required increases too. This results in a complicated critical path and, thus, decreases the maximum frequency of the DSM. The calculation times of the DSMs are listed. A*T represents the Area*Time product, which is a parameter chosen to show the total resource required for the implementations. As can be seen, when digit size d doubles, the number of clock cycles halves. However, in the same time, the clock frequency of the DSM reduces as a general trend. Note that, due to the look-up table structures in the FPGAs, there can be exceptions in such trend (m=163, d=2 and m=233, d=2). The increase of d helps decrease the A*T product because the multiplication block '$\otimes$' in the DSM structure does not dominate the

m	d	n	LUTs	Regs	Freq. (MHz)	Time (ns)	A*T (LUT*μs)
163	1	163	345	500	397.3	410	141.5
	2	82	512	502	506.5	162	82.9
	4	41	691	502	398.3	103	71.1
	8	21	880	502	336.2	62	55.0
	16	11	1587	509	302.8	36	57.7
	32	6	2940	543	263.5	23	66.9
233	1	233	485	710	455.0	512	248.4
	2	117	721	710	528.4	221	159.6
	4	59	964	711	438.4	135	129.7
	8	30	1208	714	387.3	77	93.6
	16	15	2185	713	358.2	42	91.5
	32	8	4084	728	295.2	27	110.7
283	1	283	591	861	446.7	634	374.4
	2	142	874	861	445.8	319	278.4
	4	71	1162	860	368.8	193	223.7
	8	36	1482	863	334.8	108	159.4
	16	18	2659	863	308.4	58	155.2
	32	9	5138	861	254.4	35	181.8
571	1	571	1170	1726	427.2	1337	1563.8
	2	286	1744	1726	440.7	649	1131.8
	4	143	2325	1726	396.7	360	838.1
	8	72	2914	1728	336.0	214	624.4
	16	36	5252	1727	307.3	117	615.3
	32	18	9937	1726	269.8	67	663.0

Tab. 3.3: $GF(2^m)$ Digit Serial Multiplier implementation Results on Virtex-V FPGA

area when d is small. However, when d is large enough (d=8 for $GF(2^{163})$ and d=16 for $GF(2^{233})$, $GF(2^{283})$ and $GF(2^{571})$), the A*T product reaches a minimum value. In this circumstance the '$\otimes$' block dominates the area of the DSM. The calculation time reduction accrued by increase of d over 16 does not compensate for the corresponding area increment. Using d = 32 does not give a better resource efficiency (A*T product) than using d = 16.

3.5.3 Karatsuba Multiplier

Another popular multiplier, the Karatsuba multiplier, was proposed by Karatsuba et al in 1963 [77]. This multiplier was used in many Pairing designs [129, 130]. Let A and B be 2n-bit integers. Split A and B into halves as $A = a_1 2^n + a_0$ and $B = b_1 2^n + b_0$. The product P of A and B, using polynomial multiplication, is given

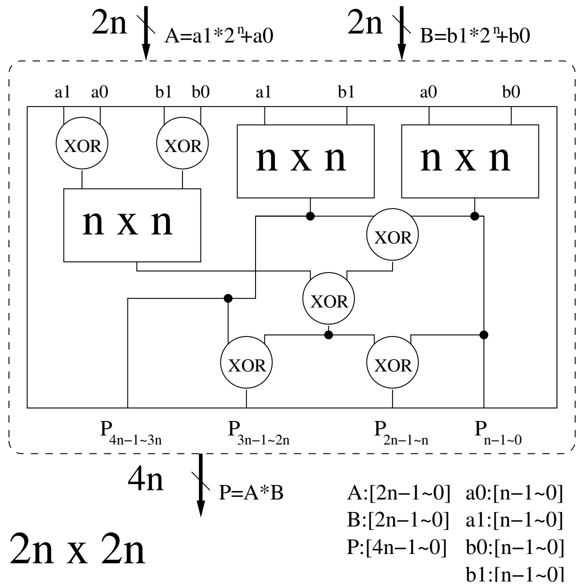


Fig. 3.3: Karatsuba multiplier of $2n$-bit×$2n$-bit

by

$$\begin{aligned} P = A \cdot B &= (a_1 2^n + a_0) \cdot (b_1 2^n + b_0) \\ &= a_1 b_1 2^{2n} + (a_1 \cdot b_0 + a_0 \cdot b_1) 2^n + a_0 \cdot b_0. \end{aligned} \tag{3.8}$$

The Karatsuba method makes use of the relationship between the most and least significant digits, that is $(a_1 \cdot b_0 + a_0 \cdot b_1) = (a_1 + a_0) \cdot (b_1 + b_0) - (a_1 \cdot b_1 + a_0 \cdot b_0)$ and product P can be calculated as per equation 3.9. The calculation of an m-bit multiplication consists of 3 $\frac{m}{2}$-bit multiplications and 6 additions. Examining one of the three $\frac{m}{2}$-bit multiplications reveals that it consists of three $\frac{m}{4}$-bit multiplications and 6 additions etc..

$$P = a_1 b_1 2^{2n} + ((a_1 + a_0) \cdot (b_1 + b_0) - (a_1 \cdot b_1 + a_0 \cdot b_0)) 2^n + a_0 \cdot b_0. \tag{3.9}$$

The original Karatsuba multiplier computes the multiplication of any bitwidth in only one clock cycle. A one level Karatsuba multiplier is shown in Fig. 3.3. This structure calculates a $2n \times 2n$ multiplication. Both inputs are of $2n$ bits and the output is a $4n$-bit array. There are 6 XOR chains and 3 sub-multipliers which, together calculate the multiplication of $n \times n$ size. The $n \times n$ sub-multiplication blocks in Fig. 3.3 are of the same structure as the $2n \times 2n$ multiplier, but the bitwidth of the buses is halved. This structure makes use of equation 3.9 by replacing an $n \times n$ sub-multiplier by 6 XOR chains. This reduces the hardware resources used.

m	$n \times n$	# of sub-mults	sub-mult for MSBs
163	22×22	27	9×9
233	15×15	81	8×8
283	18×18	81	13×13
571	18×18	243	13×13

Tab. 3.4: Sub-mult stratagy of Karatsuba multiplier over $GF(2^m)$

The Karatsuba multiplier can be successively split into smaller sub-units until the bitwidth is sufficiently small and the cost of an $n \times n$ sub-multiplier is equal to or less than the 6 XOR chains. In this case, an $n \times n$ will not be split using a Karatsuba structure again, but will use the bit-parallel multiplier structure. This smallest bitwidth equals 9 in Xilinx Virtex 5 FPGAs, i.e. implementing an 18×18 multiplier using the Karatsuba multiplier structure, or using the bit-parallel structure described in section 3.5.1, consumes equal area. However, considering the complicated critical path brought by the Karatsuba structure, as suggested by Rebeiro [130], a sub-multiplier bitwidth of approximately 20 gives the best hardware efficiency. The trade off between sub-multiplier level and size is discussed in Rebeiro's work.

By calculating all inputs in the same clock cycle, the Karatsuba method requires substantial hardware resources. Let n be the number of input bits of a sub-multiplier block and thus, for field size m, rather than consuming hardware resources proportional to m^2 (as a bit-parallel multiplier does), the hardware resources required for a Karatsuba multiplier is proportional to $n^2 \times 3^{\log_2 \frac{m}{n}}$. In most implementations using Karatsuba multipliers [129, 130, 9] n is the smallest sub-multiplier size. i.e. if the $n \times n$ sub-multiplier is replaced by another Karatsuba structure which uses three $\frac{n}{2} \times \frac{n}{2}$ sub-multipliers and some XOR chains, the hardware efficiency will be lower. Normally n is a constant or varies within a constant range for specific technologies. For example, 4 levels of sub-multipliers of size $n = 18$ shows the best efficiency for Virtex-V technology in this work. In the case of field size $m = 283$ and sub-multiplier block size $n = 18$, 81 such sub-multiplier blocks are required. However, the 81 18×18 sub-multiplier blocks provide a multiplication of 288×288. The most significant 5 bits are not necessary. In this case, the size of the sub-multiplier block which calculates the most significant bits (MSBs) of the input array is given by $n = 13$, i.e. 80 18×18 sub-multiplier blocks and one 13×13 sub-multiplier block. For other field sizes m, the sub-multiplier block for the MSBs are treated alike. The size and number of sub-multiplier blocks used in this work are listed in Table 3.4

In addition to the large area required, this Karatsuba structure results in a long

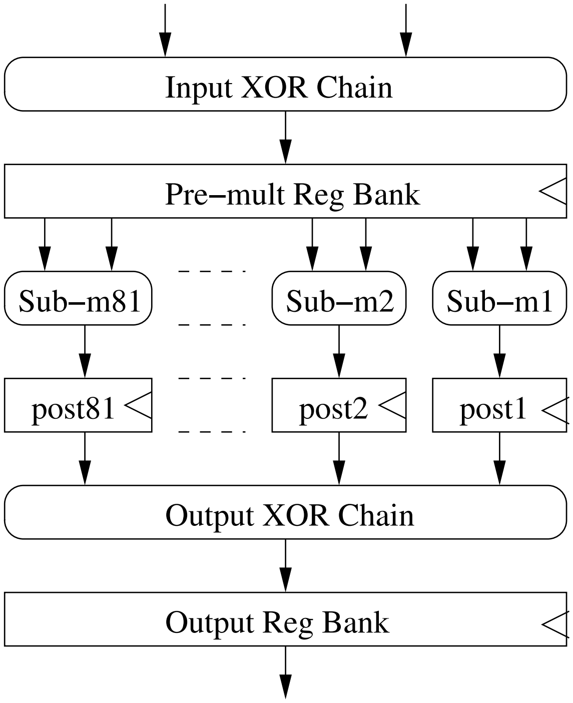


Fig. 3.4: Top level architecture of Karatsuba Multiplier with Registers inserted

critical path. In the worst case, a signal goes through 3 XOR chains in each level down from m to n. A popular method of solving this problem is to insert registers into the Karatsuba multiplier to shorten the critical path. Beuchat et al in [9] proposed a general method which improves the operation speed of this Karatsuba multiplier by inserting registers before and after the sub-multiplier blocks. The top-level architecture of the improved Karatsuba multiplier is shown in Fig. 3.4. In the figure, 'Sub-m1∼Sub-m81' represents the sub-multiplier blocks. The 'Pre-mult Reg Bank' represents the registers inserted before the sub-multiplier blocks. The results of the sub-multiplier blocks go directly into another set of registers, called the post multiplication registers, represented as 'post1∼post81'. This method improves the critical path of the Karatsuba multiplier. Since this multiplier usually contains the most complex critical path in a design, which leads to the lowest frequency of the Pairing system, this method improves the whole Pairing calculation.

The implementation results of the Karatsuba multiplier are shown in Table 3.5. As can be seen, inserting registers increases the maximum frequency by about 70%. Both methods of inserting and not inserting registers require large amounts of LUTs. The number of registers required for the register insertion method is almost triple that of the original Karatsuba multiplier. However, because there are equal numbers of LUTs and registers in a slice of a Virtex-V FPGA, this makes use of the unused

m	Regs Inserted?	LUTs	Regs	Freq. (MHz)	Single Run (ns)	Pipeline (ns)	A*T (μ*LUT)
163	N	7151	1247	201.8	5.0	n/a	35.4
	Y	6937	3923	338.9	8.9	3.0	$20.5^{(p)}$
233	N	12049	1797	206.2	4.8	n/a	58.4
	Y	12269	5662	345.6	8.7	2.9	$35.5^{(p)}$
283	N	16269	2187	188.9	5.3	n/a	86.1
	Y	16316	6884	328.1	9.1	3.0	$49.7^{(p)}$
571	N	53441	6353	174.3	5.7	n/a	305
	Y	49873	20544	292.7	10.2	3.4	$170^{(p)}$
2^{nd}-level	N	18345	4593	175.1	28.5	n/a	524
571	Y	18702	10344	359.2	19.5	n/a	364

Tab. 3.5: FPGA implementation result of Karatsuba multipliers, over Xilinx Virtex-V

registers in the slices and does not consume extra slices.

As can be seen from Table 3.5, when registers are inserted into the Karatsuba multiplier structures, the number of clock cycles required for a single Karatsuba multiplication increases from one to three. However, by applying the pipeline method, at most three multiplications can share the same Karatsuba multiplier. The first multiplication operation requires 3 clock cycles. After that each multiplication requires only one additional clock cycle for its calculation. If there are n multiplications, a total of $n + 2$ clock cycles will be required for their calculation. In Table 3.5, superscript '$^{(p)}$' indicates that the A*T product corresponds to the pipelined design.

In a Xilinx Virtex-V LX50 FPGA, there are only 28800 LUTs. For design over a field size $m = 571$, larger FPGA chips or smaller multipliers should be chosen. In pairing systems, the Karatsuba multiplier is usually around 300-bit size because the Karatsuba multiplier requires too many hardware resources. Making use of equation 3.9, a 571-bit multiplication can be achieved by performing three 286-bit multiplications (80 18×18 sub-multiplier blocks and one $16{\times}16$ sub-multiplier block) and 4 field additions (which simply require XOR-chains). This structure is called a 2^{nd}-level 571-bit Karatsuba multiplier. The implementation results of such a multiplier are also listed Table 3.5. In the structure of a 2^{nd}-level Karatsuba multiplier, the operations before and after the three 286-bit multiplications each require one clock cycle for the calculation. Thus, a 2^{nd}-level Karatsuba multiplier requires 5 clock cycles for its calculation if there are no registers inserted, or 7 clock cycles with registers inserted (with the pipeline method applied on the three 286-bit multiplication). This 2^{nd}-level Karatsuba multiplier structure reduces the area requirement of the large size Karatsuba multiplier at the cost of more clock cycles.

3.6 Squaring over $GF(2^m)$

The squaring operation over $GF(2^m)$, $c(x) = a^2(x) \bmod f(x)$, can be performed by simply assigning the two inputs to the same value using the multipliers described in the previous section and calculating a multiplication. However, there are dedicated squarer architectures that provide significantly greater efficiency when calculating $c(x) = a^2(x) \bmod f(x)$ over a binary field.

The squaring operation over binary fields is a linear operation [52]. The operation of squaring a binary polynomial can be calculated in a very short time with little cost when compared with performing a binary multiplication. The squaring over $GF(2^m)$ can be described as: if $a(x) = \sum\limits_{i=0}^{m-1} a_i x^i$, then $a^2(x) = \sum\limits_{i=0}^{m-1} a_i x^{2i}$. In binary representation, $a^2(x)$ is obtained by simply inserting '0' bits between consecutive bits of the binary representation of $a(x)$. Fig. 3.5 shows how the m-bit polynomial is extended into a $(2m-1)$-bit polynomial. In a hardware implementation, this step is wiring, in which no logical operation is performed. To form an individual operation block over $GF(2^m)$, after '0' bits are inserted, the extended result must go through a reduction block modulo irreducible polynomial $f(x)$. The output of the reduction block, an m bits polynomial, is the result of the squaring operation.

To illustrate this bit parallel squaring structure, consider the following example. Let $a(x) = x^3 + x^2 + x + 1 = \text{‘}1111'$ be a 4-bit polynomial over $GF(2^4)$ and the irreducible polynomial $f(x)$ be $f(x) = x^4 + x + 1$. The calculation flow of $c(x) = a^2(x) \bmod f(x)$ is given in equation 3.10. Firstly, '0' bits are inserted into the input $a(x)$, forming array '1010101', represented in binary polynomial form as given in equation 3.10. $a^2(x)$ then goes through the reduction modular block. Considering the transforms of the field polynomial $f(x)$ in equation 3.6, the output $c(x)$ can be calculated as per equation 3.11.

$$a^2(x) = x^6 + x^4 + x^2 + 1 \tag{3.10}$$

$$\begin{aligned} & c(x) = a^2(x) \bmod f(x) \\ & = x^6 + x^4 + x^2 + 1 \bmod f(x) \\ & = \underbrace{(x^3 + x^2)}_{x^6} + \underbrace{(x + 1)}_{x^4} + x^2 + 1 \\ & = x^3 + x \end{aligned} \tag{3.11}$$

The implementation results of the bit-parallel squarer architecture on FPGA are

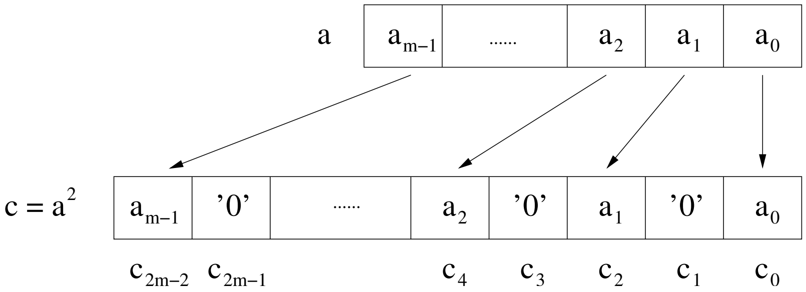


Fig. 3.5: Calculating $a^2(x)$ is achieved by inserting ‘0’ bits into $a(x)$

m	LUTs	Freq.(MHz)	Time(ns)	A*T(LUT*μs)
163	163	253.2	3.95	0.64
233	153	277.8	3.60	0.84
283	283	248.1	4.03	1.14
571	571	264.4	3.78	2.16

Tab. 3.6: FPGA implementation result of bit-parallel squarer, over Xilinx Virtex-V

presented in Table 3.6. When implementing the squarer, because the reduction block takes up most of the area, the complexity of the irreducible polynomial decides the hardware consumption. As can be seen in Table 3.6, for field size of m = 163, 283 and 571, the field polynomials are all pentanomial. The larger the field size is, the more hardware resources are required. However, for field size of m = 233, because the field polynomial is a trinomial, the squarer takes less area than that of m = 163.

3.7 *Inversion/Division over* $GF(2^m)$

In this section, the binary polynomial $a(x)$ is denoted by a for simplicity. Recall that in section 2.4.2, the inverse of a nonzero element $a \in GF(2^m)$ is the unique element g such that $a \cdot g = 1 \in GF(2^m)$, i.e. $a \cdot g \bmod f \equiv 1$. This element g is denoted $a^{-1} \bmod f$ or a^{-1} for simplicity. Division in $GF(2^m)$ can be described as $c = \frac{a}{b} \bmod f$. The division operation is equivalent to an inversion $\frac{1}{b} \bmod f$ followed by a multiplication $c = a.\frac{1}{b} \bmod f$. Or, on the contrary, the inversion can be performed using a division operation whose first operand is set equal to $1 \in GF(2^m)$. Therefore, division and inversion are interchangeable.

Based on the observation that for element a over field $GF(2^m)$, there exists $a = a^1 = a^{2^m}$ (see Fermat's Little Theorem [131]), there exists $a^{-1} = a^{2^m-2}$. This relationship shows that the inversion operation can be equivalently performed using only multiplications and squarings rather than a dedicated inversion operation. An

Algorithm 3.3 Itoh-Tsujii Inversion over $GF(2^m)$

```
Input: binary polynomials $a(x) = \sum_{i=0}^{m-1} a_i x^i$ and $b(x) = m - 1 = \sum_{i=0}^{i=r-1} b_i 2^i$
Output: $a(x)^{-1} \in GF(2^m)$
Initialize :$\chi = a; k = 1$
for $i \leftarrow r - 2$ to 0 do
    $\mu = \chi$
    for $j \leftarrow 1$ to $k$ do
        $\mu = \mu^2$
    endfor
    $\chi = \mu \cdot \chi$
    $k = 2 \cdot k$
    if $b_i = 1$ then
        $\chi = \chi^2 \cdot a$
        $k = k + 1$
    endif
endfor
return $a^{-1} = \chi^2$
```

algorithm was developed by Itoh and Tsujii [132] as shown in Algorithm 3.3. This algorithm allows the calculation of the inverse of an element over binary field $GF(2^m)$ using the existing operation blocks for multiplication and squaring.

It can be seen from Algorithm 3.3 that the calculation requires $(m-1)$ squarings and at least $r - 1 + s - 1$ multiplications, where r is the bit length of the binary representation of $(m - 1)$ and s is the Hamming weight of the binary representation of $(m - 1)$. Performing the squarings and multiplications in parallel is not possible because their results are used in the next iteration. In FPGA implementations, at least 2 clock cycles are necessary to operate a squaring operation, one clock cycle to load the operand into the squaring block and the other to calculate the result. As discussed in previous subsection 3.3.2, it takes $n = \lceil \frac{m}{d} \rceil$ clock cycles to calculate a $GF(2^m)$ multiplication. Thus, at least $2 * (m - 1) + n * (r + s - 2)$ clock cycles are needed to perform an inverse operation using Algorithm 3.3. Table 3.7 shows the cost of the Itoh-Tsujii (IT) Inversion for different field sizes. As can be seen, the calculation time of this IT Inversion depends on the scheme of multipliers used in the algorithm.

m	$GF(2^m)$ Mults	$GF(2^m)$ Sq
163	9	162
233	10	232
283	11	282
571	13	570

Tab. 3.7: Operational cost of Itoh-Tsujii Inversion

3.8 Dedicated Inversion and Division architecture over $GF(2^m)$

Apart from the Itoh-Tsujii algorithm which utilizes existing operations, there are dedicated architecture for calculating $c = \frac{a}{b} \bmod f$ over finite fields. In [133, 134, 135], the classical Euclidean algorithm was introduced. This algorithm can calculate field division and inversion (by setting input a to $1 \in GF(2^m)$). However, a disadvantage of the classical Euclidean algorithm is that it does not involve a fixed number of iterations for computing c in a given field. This makes it difficult to implement on hardware platforms. To overcome this disadvantage, the Extended Euclidean Algorithm (EEA) [134, 135, 136, 137, 138, 139] was developed. The EEA can calculate the inversion [134, 135, 136, 137] and division [138, 139] directly. The EEA division algorithm is based on the extended expression of division below. Let a and b be two elements in $GF(2^m)$ defined by the primitive polynomial f and $c = \frac{a}{b} \bmod f$, then there exists:

$$b \cdot c + f \cdot d = a \tag{3.12}$$

for some element d in $GF(2^m)$ and c is the inverse of b when $a = 1$. In [140] both an inverter and divider based on the EEA were prototyped on an FPGA. Both architectures compute the results in $2m$ clock cycles. It was found that the divider requires less hardware resources and gives a higher maximum clock frequency. Consider that, as mentioned before, a divider can perform inversion simply by setting operand a to $1 \in GF(2^m)$, while an inverter needs to perform an inversion followed by a multiplication to calculate a division. Thus, in a hardware implementation, a divider gives better performance.

A clarified version of the EEA algorithm described in [138] is given as in Algorithm 3.4. This algorithm is also presented in [140]. It efficiently makes use of properties of the binary field. There are two pairs of polynomials, (r, s) and (u, v). Note that (u, v) are degree $(m - 1)$ polynomials and (r, s) are of degree m. A 'for' loop which runs $2m$ times makes up the main body of the algorithm. In each iteration the two

Algorithm 3.4 EEA division over $GF(2^m)$

Input: $a(x), b(x) \in GF(2^m)$

Output: $c(x) = \frac{a(x)}{b(x)} \bmod f(x) \in GF(2^m)$

```
1: Init: $s^{(0)} = f, r^{(0)} = b, u^{(0)} = a, v^{(0)} = 0, \delta^{(0)} = 1$
2: for $i \leftarrow 1$ to $2m$ do
3:   if $r_0^{(i-1)} = 0$ then
4:     $r^{(i)} = (r^{(i-1)})/x$
5:     $s^{(i)} = s^{(i-1)}$
6:     $u^{(i)} = (u^{(i-1)})/x \bmod f(x)$
7:     $v^{(i)} = v^{(i-1)}$
8:     $\delta^{(i)} = \delta^{(i-1)} + 1$
9:   else
10:    if $\delta^{(i-1)} > 0$ then
11:      $r^{(i)} = (r^{(i-1)} + s^{(i-1)})/x$
12:      $s^{(i)} = r^{(i-1)}$
13:      $u^{(i)} = (u^{(i-1)} + v^{(i-1)})/x \bmod f(x)$
14:      $v^{(i)} = u^{(i-1)}$
15:      $\delta^{(i)} = -\delta^{(i-1)} + 1$
16:    else
17:      $r^{(i)} = (r^{(i-1)} + s^{(i-1)})/x$
18:      $s^{(i)} = s^{(i-1)}$
19:      $u^{(i)} = (u^{(i-1)} + v^{(i-1)})/x \bmod f(x)$
20:      $v^{(i)} = v^{(i-1)}$
21:      $\delta^{(i)} = \delta^{(i-1)} + 1$
22:    endif
23:  endif
24: endfor
25: return $c(x) = \frac{a(x)}{b(x)} = v$
```

polynomial pairs are updated independently of each other.

In Algorithm 3.4 superscripts are used to indicate which iteration the polynomial is from, i.e. $r^{(i-1)}$ indicates the value of polynomial r from the previous iteration.

In Algorithm 3.4, division by x is sometimes required. For terms of degree higher than 1, a division is simply a one bit right shift in hardware. However, for a degree 1 term, i.e. 1, a division by x requires a modulo $f(x)$ operation in addition. For example, in step 6 the operation $u^{(i-1)}/x$ can create a polynomial with term $\frac{1}{x}$. This term must be mapped back to $GF(2^m)$ modulo $f(x)$ as follows. Let $f(x) = \sum_{i=0}^{m} f_i x^i$. Recall that $f(x) = 0$ and $f_0 \equiv 1 \in GF(2^m)$. Therefore, $f_0 = 1 = \sum_{i=0}^{m} f_i x^i$. This yields:

i	δ	$r(x)$	$s(x)$	$u(x)$	$v(x)$
0	1	x^3+x+1	x^4+x+1	x^3+x^2	0
1	0	x^3+x^2	x^3+x+1	x^2+x	x^3+x^2
2	1	x^2+x	x^3+x+1	$x+1$	x^3+x^2
3	2	$x+1$	x^3+x+1	x^3	x^3+x^2
4	-1	x^2	$x+1$	x	x^3
5	0	x	$x+1$	1	x^3
6	1	1	$x+1$	x^3+1	x^3
7	0	1	1	x^3+1	x^3+1
8	1	0	1	0	x^3+1

Tab. 3.8: Calculation flow of an EEA Division over $GF(2^4)$

$$\frac{1}{x} = \sum_{i=0}^{m-1} f_{i+1}x^i \tag{3.13}$$

For example, in $GF(2^4)$ generated by $f(x) = x^4 + x + 1$, there exists:

$$\begin{aligned} &f(x) = x^4 + x + 1 = 0 \\ &1 = x^4 + x \\ &\tfrac{1}{x} = x^3 + 1 \end{aligned} \tag{3.14}$$

Table 3.8 the operations over $GF(2^4)$ with primitive polynomial $f(x) = x^4+x+1$ are taken as an example to illustrate how the division algorithm works. The division being calculated here is $c(x) = \frac{a(x)}{b(x)} \bmod f(x)$, where $a(x) = x^3 + x^2$ and $b(x) = x^3 + x + 1$. The values of the variables in every iteration are shown in Table 3.8. The correct result $c(x) = x^3 + 1$ can be derived from $v(x)$ after $2m$ clock cycles, i.e. 8 iterations are required in the division over $GF(2^4)$.

As mentioned above, in Algorithm 3.4 the intermediate polynomial pairs (r, s) and (u, v) update independently of each other. A slice-wise architecture described in [140] is applied here for the EEA divisor. In this architecture, a calculation slice chain of $(m + 1)$ slices, the RS slice chain, is used to update each bit of the r and s polynomials. Another chain of m slices, the uv slice chain, updates each bit of the u and v polynomials. The structures of the rs slice and the uv slice are illustrated in Fig. 3.6 and Fig. 3.7 respectively. The bit of the irreducible polynomial $f(x)$ being '0' or '1' affects the output of the corresponding slice. To optimize the hardware consumption, $f(x)$ can be hard coded into the design of the architecture. The input bit 'd' in Fig. 3.6 and 3.7 represents the δ signal and the input bit q is generated by

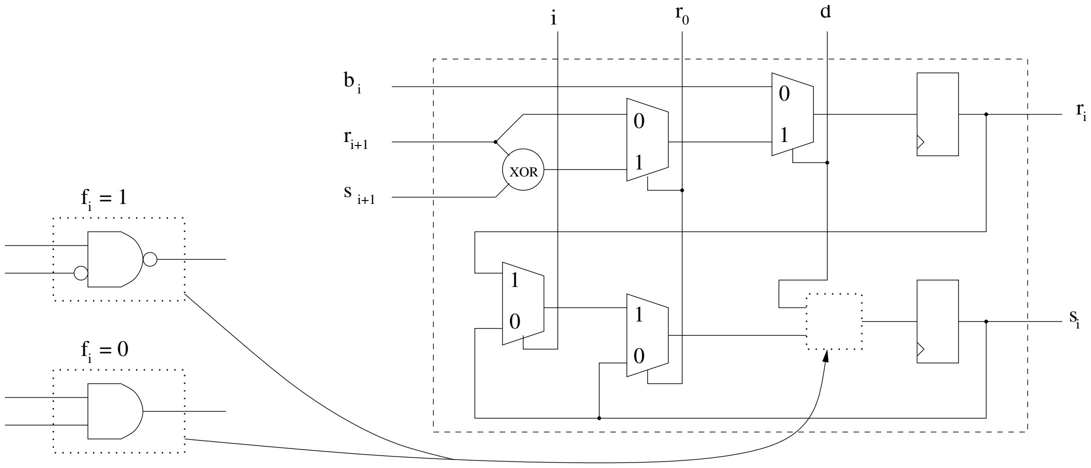


Fig. 3.6: rs slice of EEA divider

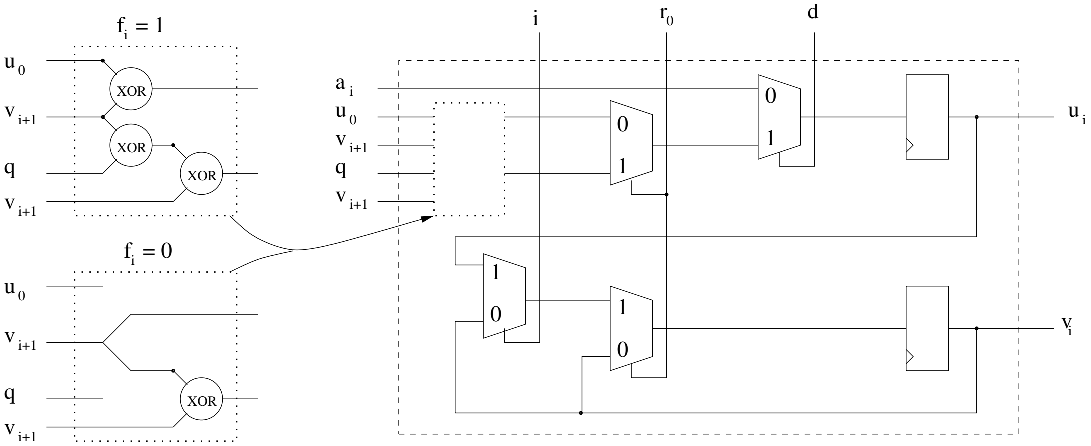


Fig. 3.7: uv slice of EEA divider

$$q = u_0 \text{ XOR } v_0 \tag{3.15}$$

The architecture of the $GF(2^m)$ EEA divisor is shown in Fig. 3.8. The hardware resources required by this architecture depends on the field size m. The larger the field size the more rs slices and uv slices are needed. A $2m$ bit counter is required in the controller to indicate when the calculation of the divisor is completed. The divider requires $(2m + 2)$ clock cycles to perform the division.

The implementation results for the EEA divisor architecture on the Virtex-V FPGA are shown in Table 3.9. Since in the rs slices and the uv slices, the bits of the polynomials are updated independently, the adjacent bits don't affect each other. Thus, the EEA divider architecture results in a high operating frequency.

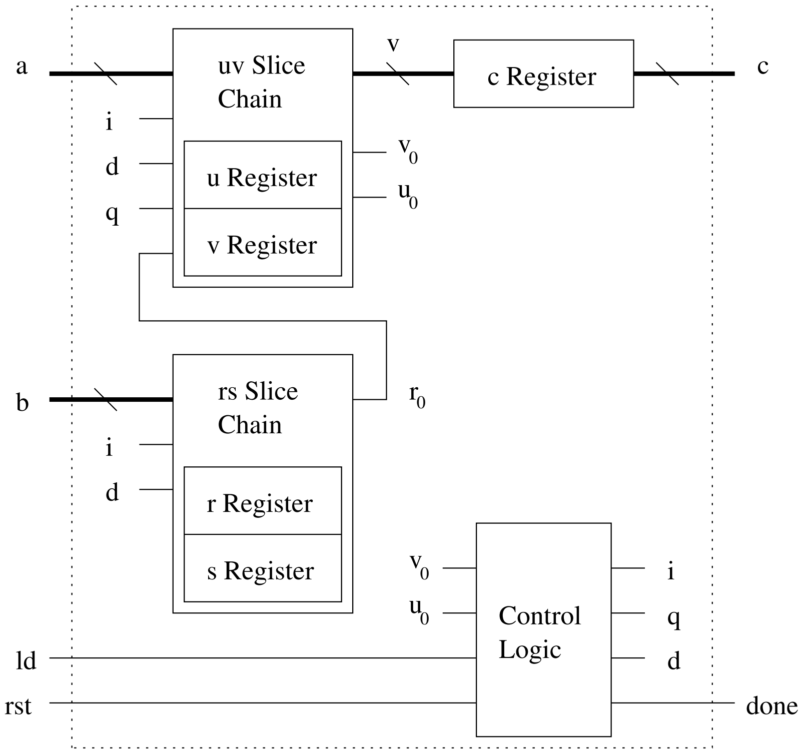


Fig. 3.8: $GF(2^m)$ EEA Divisor architecture

m	LUTs	Regs	Freq.(Mhz)	Time(μs)	A*T(LUT*μs)
163	836	991	429.7	0.76	638
233	1184	1411	430.0	1.09	1288
283	1438	1712	398.3	1.43	2051
571	2880	3441	308.7	3.71	10673

Tab. 3.9: $GF(2^m)$ EEA divider implementation Results, over Xilinx Virtex-V

3.9 Conclusions

In this chapter hardware architectures for performing $GF(2^m)$ arithmetic were discussed. The particular architectures required for operations in the η_T pairing algorithm include adders, multipliers, squarers and dividers/inverters. Among them the adder is basically an m-bit XOR chain. Similar to the adder, the squarer consists of a wiring part and a reduction block. These adder and squarer architectures are simple and require only one clock cycle for calculation.

The $GF(2^m)$ Multiplier is much more complicated than the adder and the squarer. A digit-serial multiplier was presented for calculating $GF(2^m)$ multiplication. This architecture provides a trade-off between area and speed for it has a variable digit size. This is advantageous as it allows processors based on such a multiplier to be tailored to resource constrained systems. Contrary to the reconfigurable DSM, a fully parallel Karatsuba multiplier was introduced. This multiplier is fast, but requires substantial hardware resource, suitable for large cryptosystems which take speed as their first priority.

The algorithm for an IT inversion over $GF(2^m)$ using existing squarer and multiplier operations was introduced. The performance of this inversion algorithm depends on the squarer and the chosen multiplier in the designs. Another method, the dedicated EEA divider architecture, was also discussed. Either method can be used to implement a $GF(2^m)$ division. In the Tate pairing implementation in this work, the EEA divider architecture is chosen because this architecture requires extra hardware resources and introduces more noise into the power analysis. From the attacker's point of view, the more complicated the cryptosystem that can be successfully attacked, the stronger the attack method is.

Implementation results including calculation time, area and A*T products of the dedicated architectures of the $GF(2^m)$ operations are presented in this chapter. Among them the divider consumes the longest calculation time and the highest A*T product, i.e. the most hardware resources. The multiplier architectures require less hardware resources than the divider does. However, a digit-serial structure still requires several clock cycles for its calculation. On the contrary, a Karatsuba multiplier of fully parallel structure can perform a $GF(2^m)$ multiplication in a very short time but requires extra area. The adder and squarer structures require the least hardware resources and calculation time for their calculations.

4. IMPLEMENTATION OF THE TATE PAIRING OVER EXTENSION FIELD $GF(2^{4M})$

4.1 Introduction

Chapter 3 introduced the architectures of the arithmetic operation blocks over the field $GF(2^m)$. The Tate pairing requires Multiplication, Squaring and Division/Inversion over the extension field $GF(2^{4m})$. These operations over the extension field consist of the basic operations over $GF(2^m)$. There have been many designs presented in the literature that consider extension field arithmetic. This chapter introduces the architecture of the $GF(2^{4m})$ operations and how they can be composed using the basic operation blocks. For simplicity, in this chapter the basic operations of addition, multiplication, squaring and division/inversion over $GF(2^m)$ are represented as **A**, **M**, **S** and **D/I** respectively.

Section 4.2 introduces the calculation flows of the $GF(2^{4m})$ operations. The schemes for managing the basic $GF(2^m)$ operations in the calculations are described. Section 4.3 analyzes the top-level architecture of the pairing based algorithms and two architectures are chosen for the implementation of the Tate pairing algorithm in this work. Implementation results of reconfigurable designs for different field size m, different number of multipliers and choice of multipliers are listed. Section 4.4 analyzes the implementation results and compares the results with earlier Tate pairing implementations. How the changes in architectures, components and other parameters affect the performances of the implementations are discussed.

4.2 Architecture for computations over $GF(2^{4m})$

In polynomial basis, an element over field $GF(2^{4m})$ consists of $4m$ bits. The most straight forward representation would be to consider the field $GF(2^n)$ where $n = 4m$. In this case, the arithmetic over $GF(2^{4m})$ can be implemented on the same architecture as over $GF(2^m)$. However, the resource requirement would differ a lot. For example, a multiplication over $GF(2^{4m})$ will be 16 times larger than that over

$GF(2^m)$.

As suggested by Blake et al [71], in this work the field $GF(2^{4m})$ will be represented as $GF(2^m)^4$ which is an isomorphism of $GF(2^{4m})$. The only difference between them is that the elements are represented in a different way. From now on $GF(2^{4m})$ can be considered as shorthand for $GF((2^m)^4)$. Elements of $GF(2^{4m})$ are represented in the form of polynomials of degree 3 with coefficients in $GF(2^m)$:

$$a(t) = \sum_{i=0}^{3} a_i t^i \in GF(2^{4m}), \tag{4.1}$$

where $a_i \in GF(2^m)$.Arithmetic operations over $GF(2^{4m})$ are defined modulo a degree 4 irreducible polynomial $p(t)$ whose coefficients are in $GF(2^m)$. However, if 4 does not divide m, the coefficients p_i of the irreducible polynomial can be chosen from $\{0, 1\}$ [72]. This greatly simplifies the modular reduction process.

In the η_T pairing algorithm shown in Algorithm 2.2, the intermediate variable $C(t) \in GF(2^{4m})$ is represented as

$$C(t) = C_0 + C_1 t + C_2 t^2 + C_3 t^3,$$

where $C_i \in GF(2^m)$ and $t^4+t+1 = 0$. $C(t)$ is initiated as $C(t) = 1+0t+0t^2+0t^3 = 1$ in step 1 of Algorithm 2.2. Similarly, $A(t)$ can be expressed as

$$A(t) = A_0 + A_1 t + A_2 t^2 + A_3 t^3,$$

where $A_i \in GF(2^m)$. In this work the irreducible polynomial $p(t) = t^4 + t + 1$ is chosen as is used in works [74, 8, 119].

To map an element in $GF(2^m)$ to the extension field $GF(2^{4m})$, a distortion map must be applied, shown in equation 4.2.

$$Q^* = R(Q(x, y)) = (x + t^2 + t + 1, y + (t^2 + t)x + t), \text{ where } t^4 + t + 1 = 0 \tag{4.2}$$

This equation indicates that with the polynomial $p(t) = t^4 + t + 1$, a point $Q(x, y)$ with coordinates x and y in $GF(2^m)$ can be mapped to a point Q^*, whose coordinates are in $GF(2^{4m})$. In equation 4.2, t is the generator which generates the cyclic group $GF(2^{4m})$ of order 15. The mapping operations in equation 4.2 extends two coordinates on the elliptic curve over field $GF(2^m)$ to two coordinates over field $GF(2^{4m})$. operations over the extension field $GF(2^{4m})$ can be seen as the operations between polynomials as shown in equation 4.1.

The computation of the pairings involves multiplications over field $GF(2^{4m})$. The η_T algorithm transforms such multiplications into several multiplications over field $GF(2^m)$. Moreover, the η_T algorithm integrates the operations in the distortion map into the point operations. This makes it possible for the distortion map operations to be calculated together with the other operations in the algorithm in parallel and improves the efficiency.

4.2.1 $GF(2^{4m})$ Multiplication

The multiplication of two $GF(2^{4m})$ elements can be performed using polynomial multiplication, $c(t) = a(t) \cdot b(t) = (a_3t^3 + a_2t^2 + a_1t + a_0) \cdot (b_3t^3 + b_2t^3 + b_1t + b_0)$, where a_i and $b_i \in GF(2^m)$. Naively expanding the polynomial multiplication and computing the result modulo the irreducible polynomial $p(t)$ above would cost 16**M** and 9**A**. The Karatsuba-Ofman algorithm [77] provides a less resource consuming method because addition and squaring require less hardware resources than multiplications. As introduced in section 3.5.3, a Karatsuba multiplier can calculate a $2n{\times}2n$ multiplication at the cost of 3 $n \times n$ multiplications and 6 additions. By the same method, a $4n{\times}4n$ multiplication requires 3 $2n{\times}2n$ multiplications and thus, 9 $n \times n$ multiplications instead of 16 when the naive approach is used.

For the $GF(2^{4m})$ extension field, there are 4 coefficients in each element in $GF(2^{4m})$. Rewriting the multiplication as in equation 4.3

$$\begin{aligned} c(t) &= a(t) \cdot b(t) \bmod p(t) \\ &= \sum_{i=0}^{3} a_i t^i \cdot \sum_{j=0}^{3} b_j t^j \bmod p(t) \\ &= \sum_{i=0}^{6} z_i t^i \bmod p(t) \\ &= \sum_{i=0}^{3} c_i t^i \end{aligned} \tag{4.3}$$

The polynomial multiplication consists of two parts, a multiplication part followed by a reduction operation modulo $p(t)$. Using the Karatruba method, the partial products and the coefficients of c_i are shown in equation 4.4.

$$\begin{aligned}
q_0 &= a_0b_0 & z_0 &= q_0 \\
q_1 &= a_1b_1 & z_1 &= q_0 + q_1 + q_4 \\
q_2 &= a_2b_2 & z_2 &= q_0 + q_1 + q_2 + q_5 \\
q_3 &= a_3b_3 & z_3 &= \textstyle\sum_{i=0}^{8} q_i \\
q_4 &= (a_0 + a_1)(b_0 + b_1) & z_4 &= q_1 + q_2 + q_3 + q_6 \\
q_5 &= (a_0 + a_2)(b_0 + b_2) & z_5 &= q_2 + q_3 + q_7 \\
q_6 &= (a_1 + a_3)(b_1 + b_3) & z_6 &= q_3 \\
q_7 &= (a_2 + a_3)(b_2 + b_3) \\
q_8 &= (\textstyle\sum_{i=0}^{3} a_i)(\textstyle\sum_{i=0}^{3} b_i)
\end{aligned} \tag{4.4}$$

With intermediate product $z(t)$ generated, the next step is to perform the reduction modulo $p(t)$ operation, where $p(t) = t^4 + t + 1$. Similar to equation 3.6 in section 3.5, the irreducible polynomial $p(t)$ can be extended to terms greater than 4:

$$\begin{aligned}
p(t) &= t^4 + t + 1 = 0 \\
t^4 &= t + 1 \\
t^5 &= t^2 + t \\
t^6 &= t^3 + t^2
\end{aligned} \tag{4.5}$$

This operation gives the coefficients of the final result $c(t)$, as shown in equation 4.6. The total cost of a multiplication over field $GF(2^{4m})$ is 9**M** and 22**A**.

$$\begin{aligned}
c_0 &= q_0 + q_1 + q_2 + q_3 + q_6 \\
c_1 &= q_0 + q_4 + q_6 + q_7 \\
c_2 &= q_0 + q_1 + q_5 + q_7 \\
c_3 &= q_0 + q_1 + q_2 + q_4 + q_5 + q_6 + q_7 + q_8
\end{aligned} \tag{4.6}$$

The architecture for multiplication in extension field $GF(2^{4m})$, designed by Shantz et al [77] and presented in [141], is shown in Fig. 4.1. Using 9 multipliers achieves the highest computation speed at the cost of largest area. Reducing the area (number of multipliers) will result in an increase in computation time.

4.2.2 $GF(2^{4m})$ Squaring

Similar to the $GF(2^m)$ Squarer, although a squaring can be performed using a multiplier by setting both inputs the same, a dedicated $GF(2^{4m})$ squaring is much more efficient in both timing and area. To compute a $GF(2^{4m})$ squaring $c(t) =$

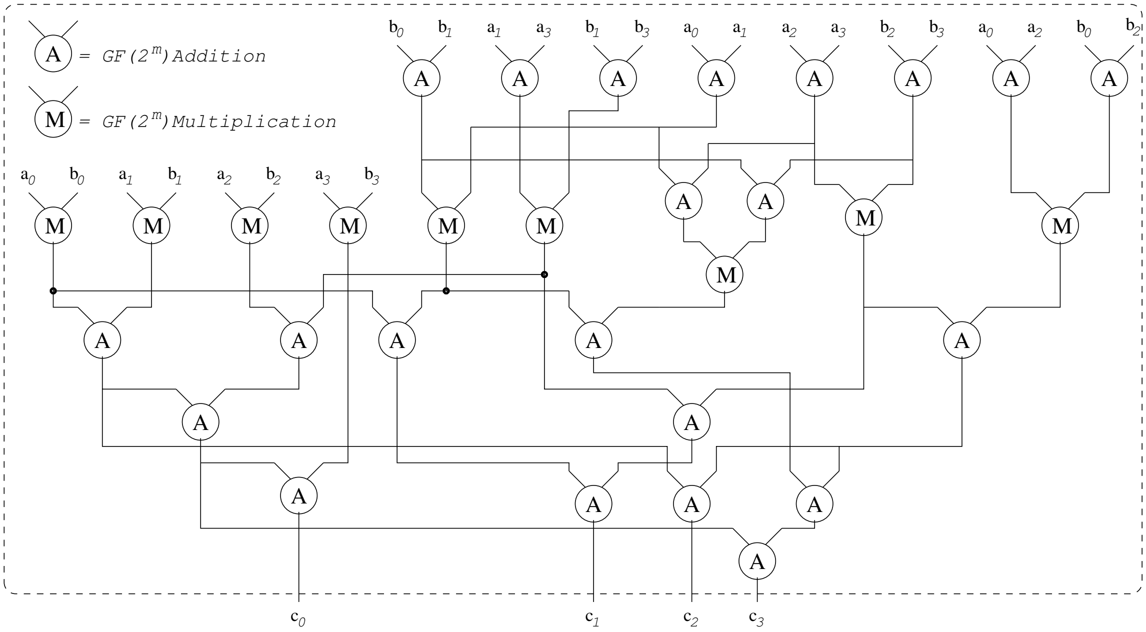

A = GF(2^m)Addition
M = GF(2^m)Multiplication
b0 b1 a1 a3 b1 b3 a0 a1 a2 a3 b2 b3 a0 a2 b0 b2
a0 b0 a1 b1 a2 b2 a3 b3
A
M
c0
c1
c2
c3

Fig. 4.1: $GF(2^{4m})$ Multiplier

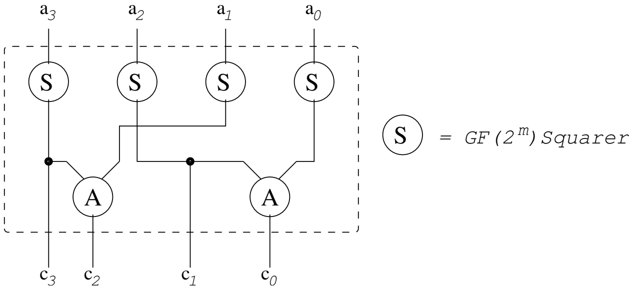


Fig. 4.2: $GF(2^{4m})$ Squarer

$z(t) \bmod p(t) = a^2(t) \bmod p(t)$, first compute the squaring stage as per equation 4.7. The result $z(t)$ is then reduced modulo $p(t)$, as shown in equation 4.5. The final result is given in equation 4.8.

$$\begin{aligned} z(t) &= a^2(t) = (a_3t^3 + a_2t^2 + a_1t + a_0)^2 \\ &= a_3^2t^6 + a_2^2t^4 + a_1^2t^2 + a_0^2 \end{aligned} \tag{4.7}$$

$$C(t) = z(t) \bmod p(t) = a_3^2t^3 + (a_3^2 + a_1^2)t^2 + a_2^2t + (a_2^2 + a_0^2) \tag{4.8}$$

Fig. 4.2 shows the architecture of a $GF(2^{4m})$ squarer which computes $c(t) = a^2(t) \bmod p(t)$. Both $GF(2^m)$ squaring and addition require only 1 clock cycle. When compared with the $GF(2^{4m})$ multiplication architecture, the squarer is much more efficient in both timing and area.

Algorithm 4.1 $GF(p^n)$ Inversion Algorithm

Input: $a(t)$, $p(t)$ such that $deg(a(t)) < deg(p(t))$
Output: $c(t) = a(t)^{-1} \bmod p(t)$
Initialize: $b = 0; c = 1; p = p(t); g = a(t)$

```
while $deg(p) \neq 0$ do
    if $deg(p) < deg(g)$ then
        exchange $p$, $b$ with $g$, $c$ respectively
    end
    $j = deg(p) - deg(g)$
    $\alpha = g_{deg(g)}^2$
    $\beta = p_{deg(p)} \cdot g_{deg(g)}$
    $\gamma = g_{deg(g)} \cdot p_{deg(p)-1} - p_{deg(p)} \cdot g_{deg(g)-1}$
    $p = \alpha.p - (\beta.t^j + \gamma.t^{j-1}).g$
    $b = \alpha.b - (\beta.t^j + \gamma.t^{j-1}).c$
    if $deg(p) = deg(g)$ then
        $p = g_{deg(p)} \cdot p - p_{deg(p)} \cdot g$
        $b = g_{deg(p)} \cdot b - p_{deg(p)} \cdot c$
    end
end
return $c(t) = a(t)^{-1} = b = p_0^{-1}.b$
```

4.2.3 $GF(2^{4m})$ Inversion

An inversion over $GF(2^{4m})$ is required in the final exponentiation step in the η_T pairing algorithm. An algorithm for inversion over $GF(p^n)$, presented in [119, 142], is shown as Algorithm 4.1.

This algorithm is a general purpose algorithm for inversion operations over any base field $GF(p)$ and any extension degree n. In the case of this work, for the field $GF(p^n)$, p is set to 2^m and $n = 4$. In Algorithm 4.1, function $deg(p)$ returns the degree of the polynomial $p(t)$. The input polynomial $p(t)$ is set to $t^4 + t + 1$, i.e. initially of degree 4. The quantity $a(t)$ in this algorithm is considered as a degree 3 polynomial. The while loop in this algorithm reduces the degree of p by one or more in every iteration until the degree of p reaches zero. It is obvious that only additions, multiplications and substractions, which are treated the same as additions, are necessary in the 'while' loop. At the end of this algorithm, a division and 4 multiplications are performed to get the result $c(t)$ such that $(a(t) * c(t)) \bmod p(t) = 1$, where $a(t) = \sum_{i=0}^{3} a_i t^i$.

Examining the algorithm for the case where the field is $GF(2^{4m})$, this algorithm takes as inputs 4 coefficients of $a(t)$, where each is over $GF(2^m)$. For optimization, the irreducible polynomial $p(t)$ of the extension field can be pre-stored in the registers. Rewriting $p(t)$ here: $p(t) = t^4 + t + 1 = \sum_{i=0}^{4} p_i t^i = p_4 t^4 + p_1 t^1 + p_0 t^0$, thus, there exists $p_4 = p_1 = p_0 = 1 \in GF(2^m)$ and $p_3 = p_2 = 0 \in GF(2^m)$ pre-stored in the registers. The 'while' loop between steps 1 and 15 runs a total of 3 times in this $GF(2^{4m})$ inversion. The intermediate variables are all elements over $GF(2^m)$. They are initiated at the start of the algorithm and are updated in every iteration. In the beginning of this computation, g is assigned to values of input $a(t) = \sum_{i=0}^{3} a_i t^i$, $b(t) = \sum_{i=0}^{3} b_i t^i$ is set to 0 and $c(t) = \sum_{i=0}^{3} c_i t^i$ is set to $0t^3 + 0t^2 + 0t + 1$, as shown in equation 4.9.

Initiation:

$$\begin{array}{llll} g_3 = a_3 & p_4 = 1 & b_3 = 0 & c_3 = 0 \\ g_2 = a_2 & p_3 = 0 & b_2 = 0 & c_2 = 0 \\ g_1 = a_1 & p_2 = 0 & b_1 = 0 & c_1 = 0 \\ g_0 = a_0 & p_1 = 1 & b_0 = 0 & c_0 = 1 \\ & p_0 = 1 & & \end{array} \tag{4.9}$$

On the first iteration, intermediate polynomials p and b will be updated as given in equation 4.10 and c and g will remain unchanged. This iteration reduces the degree of p by two. Thus, the computation of this inversion requires only three iterations of the 'while' loop.

Iteration 1:

$$\begin{array}{lll} \alpha = g_3^2 & p_4 = 0 & b_3 = 0 \\ \beta = g_3 & p_3 = 0 & b_2 = 0 \\ \gamma = g_2 & p_2 = \beta \cdot g_1 + \gamma \cdot g_2 & b_1 = \beta \\ & p_1 = \alpha \cdot p_1 + \beta . g_0 + \gamma \cdot g_1 & b_0 = \gamma \\ & p_0 = \alpha \cdot p_0 + \gamma \cdot g_0 & \end{array} \tag{4.10}$$

The updates of the intermediate polynomials in the second and the third iteration are shown as in equation 4.11 and equation 4.12. Note that, except for the first iteration, each iteration of the ' while' loop reduces the degree of p by one. At the end of the third iteration, the function $deg(p)$ returns 0, which indicates the end of the 'while' loop.

Iteration 2:

$$\begin{array}{llll}
\alpha = p_2^2 & p_4 = 0 & g_2 = p_2 & b_3 = 0 \\
\beta = g_3 \cdot p_2 & p_3 = 0 & g_1 = p_1 & b_2 = \beta \cdot b_1 \\
\gamma = g_2 \cdot p_2 + g_3 \cdot p_1 & p_2 = 0 & g_0 = p_0 & b_1 = \beta \cdot b_0 + \gamma \cdot b_1 \\
 & p_1 = \alpha \cdot g_1 + \beta . p_0 + \gamma \cdot p_1 & c_1 = b_1 & b_0 = \alpha + \gamma \cdot b_0 \\
 & p_0 = \alpha \cdot g_0 + \gamma \cdot p_0 & c_0 = b_0 &
\end{array} \tag{4.11}$$

Iteration 3:

$$\begin{array}{lll}
\alpha = p_1^2 & p_4 = 0 & b_3 = \beta \cdot b_2 \\
\beta = g_2 \cdot p_1 & p_3 = 0 & b_2 = \beta \cdot b_1 + \gamma \cdot b_2 \\
\gamma = p_1 \cdot g_1 + g_2 \cdot p_0 & p_2 = 0 & b_1 = \alpha \cdot c_+ \beta \cdot b_0 + \gamma \cdot b_1 \\
 & p_1 = 0 & b_0 = \alpha \cdot c_0 + \gamma \cdot b_0 \\
 & p_0 = \alpha \cdot p_0 + \gamma \cdot g_0 &
\end{array} \tag{4.12}$$

Following the three iterations of the ‘while’ loop, finally step 16 computes:

$$c_i = p_0^{-1} \cdot b_i, \quad i = 0, 1, 2, 3 \tag{4.13}$$

Note that to speed up this $GF(2^{4m})$ inversion computation, in the derivation of equations 4.9,4.10,4.11,4.12 and 4.13 the degree of the input polynomial $a(t)$ is assumed to be 3. In the case where $a_3 = 0$, i.e. the assumption is untrue, the iterations above will be different and the operation flow of such inversions must be re-derived. In practical cryptosystems, the probability that this occasion happens is $\frac{1}{2^m}$ for field size m. This value is sufficiently low that the benefit of the assumption far outweighs its cost. When this inverter structure is used in the Tate pairing architecture, it is possible to have the Tate pairing architecture check that the term a_3 of input $a(t)$ equals to $0 \in GF(2^m)$. If $a_3 = 0$, then that particular Tate pairing computation can be discarded. An alternative way is to apply detection mechanism in the division structure. In the case of $a_3 = 0$, input $a(t)$ can be multiplied by t, i.e. left shift by one digit. After the division algorithm, the result is then divided by t, i.e. right shift by one digit.

From equations 4.9, 4.10, 4.11, 4.12 and 4.13, it can be seen that the $GF(2^{4m})$ inversion is very time consuming. A total number of 16**A**+4**S**+34**M**+1**I** are required.

$c = c^{2^0}$	c^{2^1}	c^{2^2}	c^{2^3}	c^{2^4}	c^{2^5}
c_3	c_3^2	$c_3^{2^2}$	$c_3^{2^3}$	$c_3^{2^4}$	$c_3^{2^5}$
c_2	$c_1^2 + c_3^2$	$c_3^{2^2} + c_3^{2^2}$	$c_1^{2^3}$	$c_2^{2^4}$	$c_1^{2^5} + c_3^{2^5}$
c_1	c_2^2	$c_3^{2^2} + c_3^{2^2}$	$c_2^{2^3} + c_3^{2^3}$	$c_1^{2^4}$	$c_2^{2^5}$
c_0	$c_0^2 + c_2^2$	$c_3^{2^2} + c_3^{2^2} + c_3^{2^2} + c_3^{2^2}$	$c_0^{2^3} + c_1^{2^3}$	$c_0^{2^4}$	$c_0^{2^5} + c_2^{2^5}$

Tab. 4.1: Successive squaring over $GF(2^{4m})$

4.2.4 $GF(2^{4m})$ Frobenius Map

When implementing the Tate pairing, there is a final exponentiation which requires raising an extension field element $c(t)$ to the power of $q = 2^m$, i.e. calculating $c(t)^{2^m}$. This operation is known as the Frobenius map. It is equivalent to m successive squarings. There are dedicated operations for efficiently computing this Frobenius map.

In extension fields, successive squarings which raise $GF(2^{4m})$ elements to the power of 2^i can be optimized. Table 4.1 illustrates the results of successively squaring a $GF(2^{4m})$ element $c = c(t) = \sum_{i=0}^{3} c_i t^i$, where $c_i \in GF(2^m)$. It is noted that the coefficients of the calculation results are equivalent for the case of c^{2^0} and c^{2^4}, likewise for c^{2^1} and c^{2^5}, i.e. the coefficients of $c(t)$ raised to power of 2^k and 2^{k+4} are the same. Thus, for all the cases considered here, the exponentiation coefficients result in 4 cases. The coefficients of c^{2^k} depends on the value of $k \mod 4$. Consider the property for elements over binary fields $a^{2^m} = a$ for field $GF(2^m)$, when $k = m$, the terms $c_i^{2^k}$ are equivalent to c_i or sum of c_i.

The calculation result of the frobenius map is illustrated in Table 4.2. The results depend on the field size m mod 4. The field size is always chosen to be a prime for security reasons (see chapter 2). The results of field sizes modulo 4 are respectively: 3, 3, 1 and 3 for the chosen field sizes 163, 233, 283 and 571 in this work. From Table 4.2 it can be seen that this mapping operation requires only 2 additions over $GF(2^m)$. From the hardware point of view, this operation requires only $2 * m$ XOR gates. This is a very efficient method to raise an element over $GF(2^{4m})$ to the power of 2^m.

Consider the final exponentiation operation $C(t) \leftarrow C(t)^{2^{2m}-1}$ in step 8 of Algo-

$c = a^{2^m}$	m mod 4			
	0	1	2	3
C_3	a_3	a_3	a_3	a_3
C_2	a_2	$a_1 + a_3$	$a_2 + a_3$	a_1
C_1	a_1	a_2	$a_1 + a_3$	$a_2 + a_3$
C_0	a_0	$a_0 + a_2$	$a_0 + a_1 + a_2 + a_3$	$a_0 + a_1$

Tab. 4.2: Computing $c = a^{2^m}$ over $GF(2^{4m})$

rithm 2.2. This operation can be expanded into:

$$
\begin{aligned}
C^{2^{2m}-1} &= C^{(2^m+1)(2^m-1)} \\
&= (C^{(2^m+1)})^{(2^m-1)} \\
&= (C^{2^m} \times C)^{(2^m-1)} \\
&= \tfrac{((C^{2^m} \times C))^{2^m}}{((C^{2^m} \times C))}
\end{aligned}
\tag{4.14}
$$

Equation 4.14 uses C to represent $C(t)$ for simplicity. The result of equation 4.14 shows that making use of the Frobenius map, the final exponentiation step can be simplified to several $GF(2^{4m})$ operations: a frobenius map , a $GF(2^{4m})$ multiplication followed by a $GF(2^{4m})$ division. Here the $GF(2^{4m})$ division can be calculated through a $GF(2^{4m})$ inversion followed by a $GF(2^{4m})$ multiplication, introduced in section 4.2.3 and 4.2.1 respectively. Expanding them into $GF(2^m)$ operations, this final exponentiation operation requires 64**A**, 4**S**, 52**M** and 1**I**.

4.3 Implementing the η_T algorithm for calculating Tate pairing

Implementing the Tate pairing amounts to properly managing the operation blocks and the operation flow of the design to complete the calculation of the η_T algorithm, as introduced in section 2.5.2. Here the implementation of the η_T algorithm is introduced.

4.3.1 Top level architecture design of η_T algorithm

As shown in Algorithm 2.2, the ‘for’ loop in steps 3-7, which will be executed m times in each η_T pairing calculation, consists of 24**A**, 8**S** and 7**M** in $GF(2^m)$. To perform the operations in the ‘for’ loop, the operation blocks must be reused in each iteration. Apart from the ‘for’ loop, there are other operations over $GF(2^m)$. The pre-‘for’ operations in step 2 require 1**A**, 6**S** and 1**M**. The final exponentiation operation in step 8 requires 64**A**, 4**S**, 52**M** and 1**I**. The total number of $GF(2^m)$ operations for

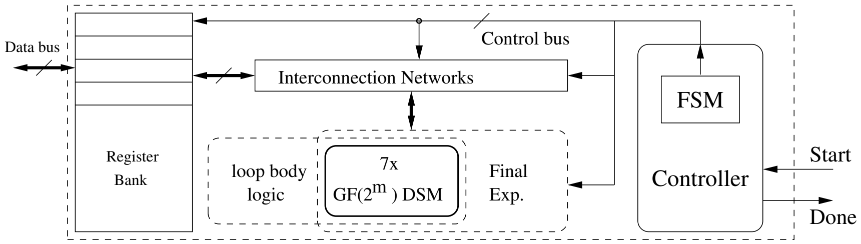


Fig. 4.3: Top-level architecture of Shu's design [7]

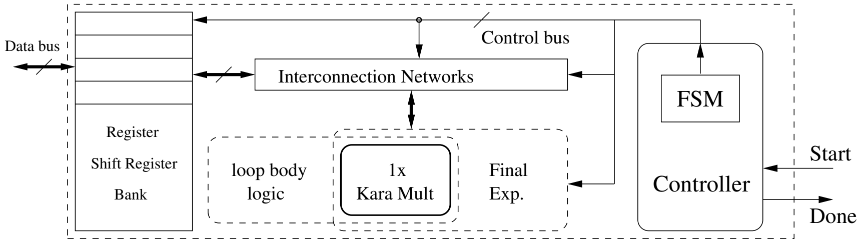


Fig. 4.4: Top-level archigtecture of Ghosh's design [10]

the η_T algorithm calculation is (24m+65)**A**, (8m+10)**S**, (7m+53)**M** and 1**I**. Many accelerators of η_T algorithm over $GF(2^m)$ have been designed [7, 8, 9, 10, 11, 12].

Review of other η_T Pairing Architectures

Shu et al in [7] and Li et al in [8] proposed an ALU for the operations of the 'for' loop in step 3-7 and the final exponentiation in step 8, as shown in Fig. 4.3. In this design, all the addition and squaring operations are expanded in the combinational logic. This requires a large amount of hardware resources, but computes all operations, other than multiplications, in a short time. 7 digit serial multipliers are used in this design where all the multipliers are reused by the 'for' loop operation in step 3-7 and the final exponentiation operation in step 8. A controller, which mainly consists of a finite state machine (FSM) and several selector signals, controls the interconnection networks to choose the right inputs from the register bank, input them into the operation blocks and output the results back into the register bank for the next iteration.

Ghosh et al in [10] and Beuchat et al in [11] proposed another type of implementation of the Tate pairing using the Karatsuba multiplier, as shown in Fig. 4.4. Similar to Shu's design, Ghosh and Beuchat expand the operations and used dedicated combinational logic to calculate the 'for' loop and the final exponentiation. The difference is that Ghosh's work uses a Karatsuba multiplier which consumes large area but is very time efficient. Thus, only 1 multiplier is used in Ghosh's

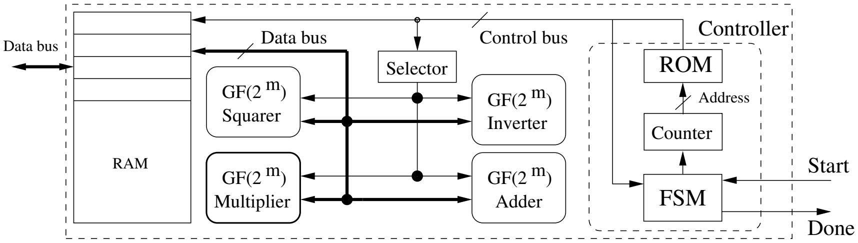


Fig. 4.5: Top-level archigtecture of Keller's design [12]

design. This multiplier is reused by both the 'for' loop logic and the final exponentiation logic. Selectors are used to choose the inputs of the multiplier from the register bank. This structure considers only the fastest implementation method and requires significantly large area. In this work, more reconfigurable and flexible methods to implement the Tate pairing algorithm will be considered.

In [12], Keller et al designed a bus type architecture, as shown in Fig. 4.5. In this design, all the calculations are broken down into basic operations over $GF(2^m)$. As can be seen from Fig. 4.5, only an adder, a squarer, a multiplier and an inverter can be found in the top-level architecture. All these blocks calculate the very basic operations over $GF(2^m)$. A data bus connects all these blocks and the register bank. This structure requires less hardware resources than the combinational logic structure. In the mean time, the controller plays a major role. The operation flow is stored in the ROM which connects to all the $GF(2^m)$ operation blocks through the control bus. The control bus decides which block is activated, assigns the inputs of the $GF(2^m)$ operation blocks and reads the results back when the operations finish. In this way, all the operation blocks are reused throughout the pairing calculation. The price of saving hardware resources is that more clock cycles are required. Since the data bus cannot assign the inputs to two blocks at the same time, this results in the calculation of the pairing in a serial manner. The additions and squarings each require two clock cycles, one for the assignment and the other for reading back the result to the register bank.

The earlier Tate pairing implementations used different top-level architectures and different components. However, the basic components of these designs are the same.

- Data access mechanism: A design can use register banks or block RAMs to save the data. Register banks require extra slice flip-flops, i.e. extra hardware resources. Block RAMs does not take up extra slice flip-flops, but only two variables stored in the block RAM can be read at a time. Thus, the Block RAM suits the bus type structure better than the combinational logic structure.

- Controller: A controller usually consists of an FSM and a control bus or some control signals. In Keller's work [12], a ROM is used to control the Control bus with pre-stored instructions. This helps reduce the complexity of the FSM.

- Selector: In combinational logic designs, all the registers are connected to the corresponding operation logic. Multiplexors and enable signals (namely interconnection networks in Fig. 4.4 and 4.3), together with the control bus, control the operation flow of the design. In bus type designs, a Data bus connects all the operation blocks with the block RAM. A selector controlled by the controller chooses which operation block is to be enabled.

- Operation logic: The combinational logic structure expands all operations required in parallel. If some logic in the structure can be reused by two or more operations, a multiplexor is usually used to select the current inputs of the logic. This structure is fast, but results in large area. The bus type structure uses operation blocks for each independent function. Such blocks are connected with the block RAM through a data bus. When the block RAM presents different data on the bus, the inputs of the blocks are chosen. Thus, the operation blocks in the bus type structure can be reused by any input. However, the block RAM can only present two variables on the Data bus. This increases the calculation time of the bus type structure designs.

Top level architecture of Bus type

Fig. 4.6 shows the top level architecture for implementing the Tate pairing using the η_T algorithm. As can be seen from the figure, the bus type top-level architecture applied in this work is similar to Keller's architecture in Fig. 4.5. The architecture is based on the $GF(2^m)$ modules. A block RAM is used to store the inputs and intermediate variables required in the algorithm. The $GF(2^m)$ blocks and memory are controlled by a FSM controller, which iterates through an instruction set, pre-stored in ROM. No $GF(2^{4m})$ modules can be seen on the top level because the $GF(2^m)$ function blocks are reused for each of the $GF(2^{4m})$ operations. The instructions for the $GF(2^{4m})$ operations are stored in the ROM and all the operations have access to all the $GF(2^m)$ blocks so that the design will reach a high utilization of all its hardware.

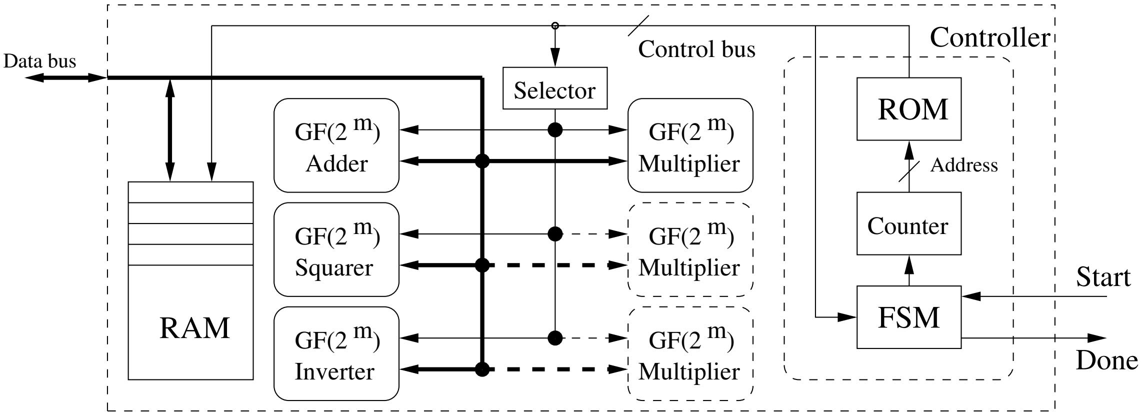


Fig. 4.6: FPGA architecture of Tate pairing

In contrast with Keller's bus type top-level architecture, there can be more than one $GF(2^m)$ Multiplier block in the architecture of this work. Using the parallel method introduced in section 2.7.3, the $GF(2^m)$ Multipliers can operate at the same time. In each clock cycle, the data bus can only arrange the inputs to one operation block, thus having one more parallel multiplier in the architecture saves $n-1$ clock cycles, at the cost of one more $GF(2^m)$ Multiplier and one more m-bit selector array. By the same method, there can be more than one Adder, Squarer or Inverter. The $GF(2^m)$ addition and squaring operation both require only 2 clock cycles for their calculation in the bus-type architecture. Thus, even if the designer increases the number of $GF(2^m)$ Adder or $GF(2^m)$ Squarer block, the data bus does not have spare clock cycles to write to and read from the additional $GF(2^m)$ Adder or $GF(2^m)$ Squarer block. Thus, there is only one $GF(2^m)$ Adder and one $GF(2^m)$ Squarer block in the bus type architecture of this work. For the Inverter block, because there is only one $GF(2^m)$ Inversion operation in the whole η_T algorithm calculation, it is not necessary to put mores $GF(2^m)$ Inverter block in the architecture.

4.3.2 Reconfiguration of the multiplications in Bus type design

In the bus type architecture applied in this work, the steps in Algorithm 2.2 are calculated serially. Operations in steps 1, 2 and 8 in the algorithm are executed only once in the η_T pairing calculation, while operations in the 'for' loop in steps 3-7 are executed m times and dominate the calculation time of the whole design. The arrangement of the operations in the 'for' loop affects the calculation time of the η_T algorithm.

Note that in step 4 of Algorithm 2.2, the intermediate variable $GF(2^{4m})$ element

$A(t)$ contains only 3 terms, i.e. the coefficient a_3 of the degree 4 term a_3t^3 equals 0. This makes it possible to optimize the $GF(2^{4m})$ Multiplication, the operation of $C(t) \leftarrow C(t)^2 \times A(t)$ in step 5 of Algorithm 2.2.

The calculation of $C(t) \leftarrow C(t)^2 \times A(t)$ consists of two parts, the $GF(2^{4m})$ squaring part and the $GF(2^{4m})$ multiplication part. Expanding the operations and rewriting step 5 of Algorithm 2.2 results in equation 4.15:

$$\begin{aligned} &C(t) \leftarrow C_{(i)}(t)^2 \\ &C_{(i+1)}(t) \leftarrow C(t) \times A(t) \end{aligned} \tag{4.15}$$

In equation 4.15, $C_{(i)}(t)$ represents the variable value for the current iteration, derived from previous iteration. $C(t)$ is an intermediate variable and $C_{(i+1)}(t)$ is the calculated value of this iteration, ready for the calculation in the next iteration. The squaring calculation in equation 4.15 can be calculated in 4 **S** and 2 **A**, as introduced in section 4.2.2. Here the calculation of $C_{(i+1)}(t) \leftarrow C(t) \times A(t)$ is discussed. Let

$$A(t) = a_0 + a_1t + a_2t^2,$$

where $a_0 = \gamma + u + \beta, a_1 = \alpha + v, a_2 = a_1 + 1$. $C(t)$ and $C_{(i+1)}(t)$ can be written as follows:

$$\begin{aligned} &C(t) = c_0 + c_1t + c_2t^2 + c_3t^3, \text{where } c_i \in GF(2^m) \\ &C_{(i+1)}(t) = c_{0(i+1)} + c_{1(i+1)}t + c_{2(i+1)}t^2 + c_{3(i+1)}t^3, \text{where } c_{j(i+1)} \in GF(2^m). \end{aligned}$$

Expand the calculation $C_{(i+1)}(t) \leftarrow C(t) \times A(t)$, yields

$$\begin{aligned} &c_{0(i+1)} + c_{1(i+1)}t + c_{2(i+1)}t^2 + c_{3(i+1)}t^3 \\ &= (c_0 + c_1t + c_2t^2 + c_3t^3) \times (a_0 + a_1t + a_2t^2) \end{aligned} \tag{4.16}$$

where

$c_{0(i+1)} = c_0a_0 + (c_2 + c_3)a_2 + c_3,$

$c_{1(i+1)} = c_0a_0 + (c_1 + c_2 + c_3)a_0 + (c_0 + c_2 + c_3)(a_0 + a_2) + c_3a_2 + c_0 + c_3,$

$c_{2(i+1)} = c_0a_0 + (c_1 + c_2 + c_3)a_0 + (c_0 + c_2 + c_3)(a_0 + a_2) + (c_1 + c_2)(a_0 + a_2) + c_1,$

$c_{3(i+1)} = (c_1 + c_2 + c_3)a_0 + (c_1 + c_2)(a_0 + a_2) + c_2.$

The expressions above show the calculation of equation 4.15 in detail. In each iteration of the 'for' loop in steps 3-7 of Algorithm 2.2, $6\mathbf{A} + 4\mathbf{S} + 1\mathbf{M}$ operations are required to update $A(t)$. Also, to update $C(t)$ in each iteration, $18\mathbf{A} + 4\mathbf{S} + 6\mathbf{M}$ are required. In the bus type top-level architecture, if there is more than one multiplier,

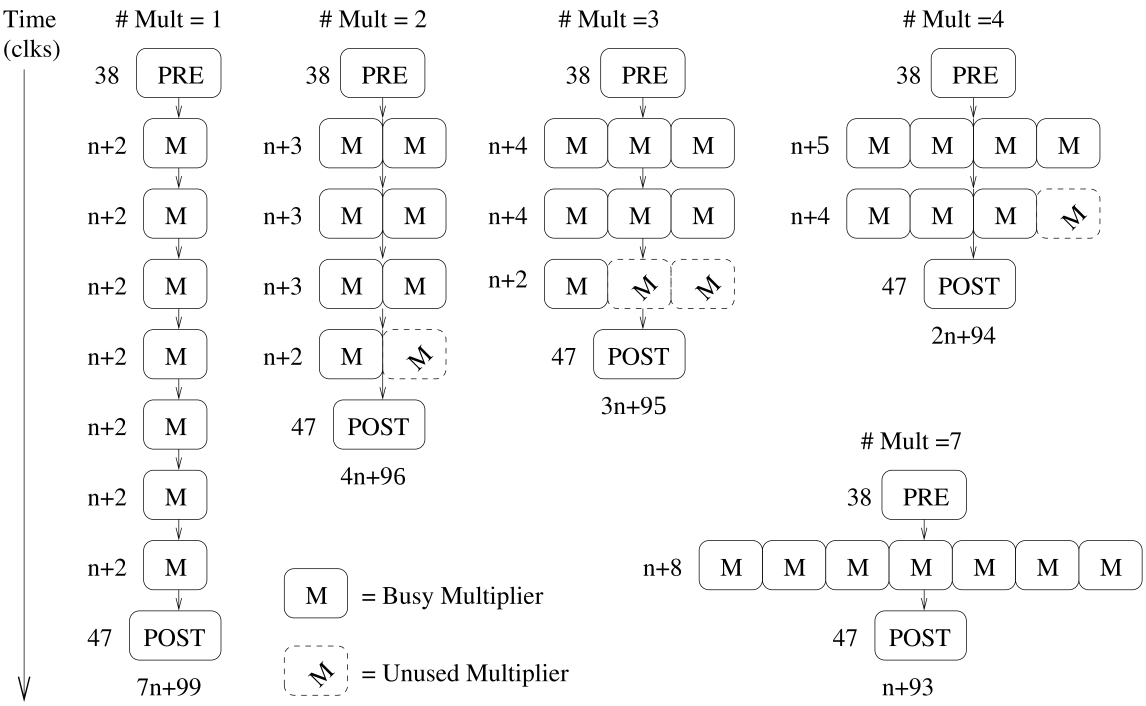


Fig. 4.7: Schedules for different numbers of Multipliers

calculation of the 7 multiplications in $A(t)$ and $C(t)$ can be carried out at the same time. By using different numbers of multipliers, there can be different schedules for computing the multiplications in steps 3-7 in Algorithm 2.2, as shown in Fig. 4.7.

In Fig. 4.7, blocks ‘PRE’ and ‘POST’ represent the pre- and post-computations of the 7**M**. The pre-computations consist of $11\mathbf{A} + 8\mathbf{S}$ and require 38 clock cycles. The post-computations consist of 13**A**, some RAM writing operations and some finite state machine operations, requiring 47 clock cycles. The detail of operation flows inside the ‘PRE’ and ‘POST’ blocks in Fig. 4.7 can be found in Fig. A.1 and A.2 in Appendix A. The operation of a single **M** requires $n = \lceil \frac{m}{d} \rceil$ clock cycles for the DSM, with one additional clock cycle reading the product from the output into memory. In this work, the processor will always spend 85 clock cycles on the ‘PRE’ and ‘POST’ operations, which are performed in serial. In this case, only 1 Adder and 1 Squarer are needed in the architecture.

With different number $\#Mult$ of multipliers, the design can deal with at most $\#Mult$ multiplications at the same time. Numbers #Mult=1, 2, 3, 4 and 7 are used in this work. Using 5 and 6 multipliers does not help reduce the calculation of the 7 multiplications. Since only 7 multiplications are needed in each iteration, putting more multipliers in the design will not speed up the operation time. As can be seen in Fig. 4.7, adding multipliers linearly reduces the required number of clock cycles. Using only one DSM, which gives a minimum area design, referred to as a minimum DSM design in later discussions, takes $7n + 99$ clock cycles. While using 7 DSMs, which requires the largest area, referred to as a maximum DSM design in later discussions, calculates the operations in the shortest time of $n + 93$ clock cycles. The details of the operation flow using different $\#Mult$ can be found in Fig. A.3, A.4, A.5, A.6 and A.7 in Appendix A.

4.3.3 Implementation results of Bus type top-level architecture

In this implementation, the Tate pairing processor is written in VHDL language. The design is simulated using ModelSim XE III 6.3c and the target FPGA technology is Xilinx Virtex-V. The designs presented in this section use a bus type top-level architecture. Different numbers of DSMs ($\#Mult$ = 1, 2, 3, 4 and 7) of different digit sizes (d = 1, 2, 4, 8, 16 and 32) are used to implement the target algorithm at the field size of m = 163, 233, 283 and 571. The Karatsuba multipliers, with and without registers inserted, are also applied in the designs because they calculate $GF(2^m)$ multiplications in very few clock cycles. Due to the large area required, each Karatsuba design contains only one Karatsuba multiplier. For different parameters the hardware designs require different area and result in different maximum clock frequency. The implementation results are for m=571 are given in Table 4.3. For convenience implementation results for field sizes m=163, 233 and 283 are listed in Appendix B, Tables B.1, B.2 and B.3.

Because there are 4 registers and 4 LUTs in each slice of the Xilinx Virtex-V xc5vlx50 FPGA and in the designs, the number of registers never exceeds that of LUTs, the number of the LUTs represents the amount of hardware resources used. As can be seen from the tables, the area and time required varies when using different values of the digit size d and different numbers of multipliers. In this work the hardware resources required (Area), the calculation time of each design (Time) and Area*Time (A*T) product are taken as the parameters to judge the performance of different designs. The A*T product represents the total resources used to perform a Tate pairing algorithm. Here a lower A*T product means that less hardware and timing resources are used.

In the design over $GF(2^{571})$, because a Karatsuba multiplier of 571 bits exceeds the area resources available, a 2^{nd}-level Karatsuba multiplier is applied. i.e. making use of equation 3.9, in calculating the 571-bit multiplication, 3 sub-multiplications, each of 286 bits, are used. As introduced in section 3.5.3, each 286-bit Karatsuba multiplier contains 80 18×18 sub-multiplier blocks and one 16×16 sub-multiplier

Parameters			Results				
#Mult	d	n	Regs	LUTs	Freq. (MHz)	Time (μs)	A*T (LUT*s)
1× DSM	1	571	7504	7598	232.4	10182.6	77.37
	2	286	7504	8162	232.4	5215.9	42.57
	4	143	7503	8745	232.4	2723.9	23.82
	8	72	7506	9886	232.4	1486.6	14.70
	16	36	7504	12748	232.4	859.2	10.95
	32	18	7513	16441	230.0	551.2	9.06
2× DSM	1	571	9231	8766	226.9	6063.2	53.15
	2	286	9233	9905	226.9	3155.4	31.25
	4	143	9228	11060	226.9	1696.4	18.76
	8	72	9236	13339	226.9	972.0	12.97
	16	36	9233	19071	226.9	604.7	11.53
	32	18	9240	25581	226.9	421.1	10.77
3× DSM	1	571	10959	11658	226.5	4609.2	53.73
	2	286	10959	13365	226.5	2427.4	32.44
	4	143	10955	15099	226.5	1332.6	20.12
	8	72	10964	18493	226.5	789.1	14.59
	16	36	10960	25400	226.5	513.5	13.04
	32	18	10965	35353	226.9	375.0	13.26
4× DSM	1	571	12685	16816	214.1	3348.0	56.30
	2	286	12685	17380	212.4	1817.0	31.58
	4	143	12680	21402	214.1	1027.1	21.98
	8	72	12692	25935	214.1	642.1	16.65
	16	36	12689	37365	214.1	446.9	16.70
	32	18	12692	49512	215.6	346.8	17.17
7× DSM	1	571	17864	17464	212.4	1829.0	31.94
	2	286	17863	21447	213.4	1037.8	22.26
	4	143	17865	28360	212.4	648.2	18.38
	8	72	17876	38361	212.4	452.3	17.35
	16	36	17870	52695	213.1	351.8	18.54
	32	18	17867	69990	209.1	308.1	21.56
Kara.	no Reg	5	8649	24808	176.6	419.7	10.41
Kara.	with Reg	7	16133	25119	252.9	325.1	8.17

Tab. 4.3: Implementation results of Tate pairing, bus type, Xilinx Virtex-V, m=571

block. In this case, a 286-bit Karatsuba multiplier plus 5 clock cycles, rather than a 571-bit Karatsuba multiplier plus 1 clock cycle, are required. When the registers for reducing the critical path are inserted, 7 clock cycles are required for this multiplication. The two additional clock cycles, in this case, are required for the signals to go through the inserted registers in the multiplier. This 2^{nd}-level structure requires

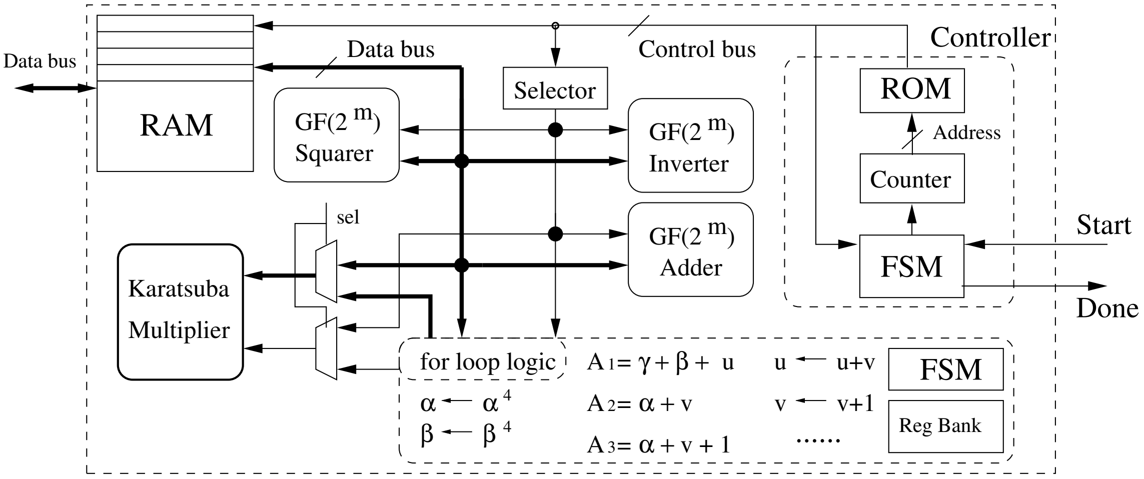


Fig. 4.8: Mixed type architecture of Tate pairing

more operating time than using original Karatsuba multipliers, as in designs for m =163, 233 and 283. In the designs for m=571, the Karatsuba multiplier does not result in a much faster calculation speed than the fastest DSM designs. However, as a medium size, but fast multiplier, the 2^{nd}-level Karatsuba multiplier provides the best A*T product amongst the designs over $GF(2^{571})$.

4.3.4 Implementation results of mixed type top-level architecture

For the m=163, #*mult*=7, d=16 and d=32 designs, the calculation time of the multiplications halves when d increases from 16 to 32. However, the calculation time of the whole Tate pairing does not decrease significantly. This is because the 'small' operations, including additions and squarings, dominate the calculation time when the calculation times of the multiplications are reduced. The two clock cycles required for addition and squaring in the bus type structure become the speed bottle neck.

In Algorithm 2.2, steps 1, 2 and 8 operate only once. The addition and squaring operations in these steps require some time, but not a significant amount. The bus-type design with instructions pre-stored in the ROM reduces the hardware resources requirement. In contrast, the addition and squaring operations in steps 3-7, i.e. the 85 clock cycles for addition and squaring operations in the 'for' loop as illustrated in Fig. 4.7, are operated m times in the Tate pairing calculation. One may think of reducing this calculation time of 85*m clock cycles by implementing the operations in parallel, i.e. in the form of combinational logic, using the same structure as used in the designs illustrated in Figs. 4.3 and 4.4.

Parameters			Results				
m	Type	n	Regs	LUTs	Freq. (MHz)	Time (μs)	A*T (LUT*s)
163	no Reg	1	10781	16243	153.1	13.4	0.22
	with Reg	1	14003	16395	240.1	9.9	0.16
233	no Reg	1	15251	24698	153.1	18.4	0.45
	with Reg	1	19916	24587	240.4	13.7	0.34
283	no Reg	1	18586	32604	146.1	23.1	0.75
	with Reg	1	24256	32345	238.5	16.5	0.53
571	no Reg	5	41341	49321	176.7	128.7	6.35
	with Reg	7	47094	49666	240.1	128.4	6.38

Tab. 4.4: Implementation results of Tate pairing, mixed type architecture, Xilinx Virtex-V

In the architecture illustrated in Fig. 4.8, the addition and squaring operations in steps 3-7 are parallel implemented in the '**for loop logic**' block. These operations can be calculated in only one clock cycle. Since the '**for loop logic**' block operates on all intermediate variables in the same clock cycle, such variables are stored in the '**Reg Bank**' of the '**for loop logic**' block and written back into RAM after the 'for' loop calculation ends. Operations in steps 1, 2 and 8 are the same as in the bus-type design, i.e. an adder, a squarer and a divider connected to the data bus calculate the operations in steps 1, 2 and 8 in serial form, with instructions pre-stored in the ROM in the controller. In controlling the multiplier, a 'sel' signal selects the controller of the Karatsuba multiplier. In steps 1, 2 and 8, the inputs of the Karatsuba multiplier can be arranged by the data bus, controlled by the ROM. In steps 3-7, i.e. the 'for' loop, the inputs of the Karatsuba multiplier can be arranged by the **for loop logic** block. This architecture significantly reduces the calculation times of the designs with fast implementation such as when the Karatsuba multiplier is used. The implementation results are shown in Table 4.4.

In Table 4.4, the Karatsuba multipliers with and without registers inserted, are used. Similar to using DSMs, the parameter n in this table indicates the number of clock cycles required to calculate a multiplication. In the case without registers inserted, for designs with m=163, 233 and 283, the 7 multiplications are input into the Karatsuba multiplier serially and read out one clock cycle after the input serially as well. For example, in clock cycles $\{1, 2, 3, 4, 5, 6, 7\}$ the inputs of the 7 multiplications are read from registers in the '**for loop logic**' block and sent into the multiplier. After one clock cycle, i.e. in clock cycles $\{2, 3, 4, 5, 6, 7, 8\}$, the calculation results are read and written back to the corresponding registers respectively. The number of clock cycles required for each iteration of the 'for' loop is 9.

In the case with registers inserted, a single Karatsuba multiplier requires 3 clock cycles for the calculation. However, according to its structure as illustrated in Fig. 3.4, for designs with m=163, 233 and 283, the multiplications can be pipelined in the Karatsuba multiplier. For example, in clock cycles $\{1, 2, 3, 4, 5, 6, 7\}$ the inputs of the 7 multiplications are serially sent to the multiplier. After three clock cycles, i.e. in clock cycle 4, the result of the first multiplication can be read from the output register of the Karatsuba multiplier. In clock cycles $\{5, 6, 7, 8, 9, 10\}$, the results of the rest 6 multiplications are read from the output and written back to their corresponding registers respectively. Thus, the number of clock cycles required for each iteration of the 'for' loop is 11.

Because the m=571 design uses a 2^{nd}-level Karatsuba multiplier, in which the 286-bit multiplier is reused 3 times, the schedule for the design with m=163, 233 and 283 cannot be applied to the m=571 design. In the mixed type m=571 designs, it requires 5 and 7 clock cycles for the Karatsuba multipliers without and with registers, respectively.

It must be noted that the hardware resources required for designs with field size m=283 and 571 have exceeded the constraint of Virtex-V xc5vlx50 FPGA in this work (28800 Regs, 28800 LUTs). The area requirements of these designs are only theoretical values after synthesis. These designs can only be implemented in larger FPGA technologies.

4.4 Analysis of implementation result

The results of implementations using different top-level architectures and different parameters show different performances. For each m, when the parameters ($\#Mult$ and d) change, the trends of how the implementation results change are similar. Here implementation results of designs over $GF(2^{571})$ are discussed.

4.4.1 Time analysis

Consider now the calculation time requirement shown in Fig. 4.9. As the calculation time varies when different multiplier schedules are applied, the slowest design requires over 100 times the calculation time of the fastest one to complete the calculation, the figures showing calculation time use a logarithmic scale. Fig. 4.9 shows the calculation time required for designs using different numbers of multipliers ($\#Mult$) and using different digit sizes d. In this figure, each line shows a multiplier schedule used in the designs, i.e. using $\#Mult \in \{1, 2, 3, 4, 7\}$ DSMs, or using one Karatsuba

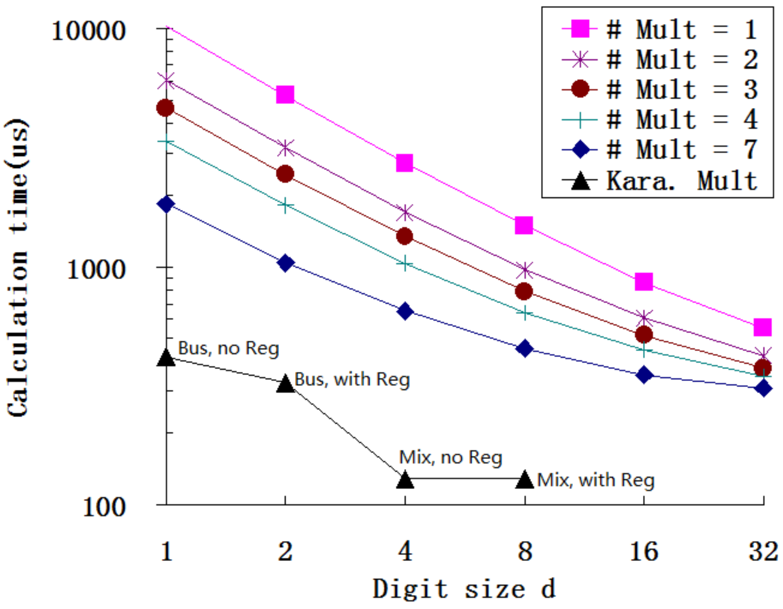


Fig. 4.9: Calculation time of η_T algorithm implementations, m=571

multiplier. In designs using DSMs, when using the same number of multipliers,an increase in digit size d leads to a decrease in the number n of clock cycles required for each digit serial multiplication and, thus, decreases the calculation time required. It can be clearly seen from Fig. 4.9 that for the same $\#Mult$, when the digit size d increases, the calculation time decreases. The decrease rate, i.e. the slope, of $\#Mult = 7$ is smaller than those of the others. This is because the calculation times of the multiplications in the Tate pairing are already improved by increasing the number of DSMs. Increasing the digit size d is more effective in designs with fewer multipliers. For field $GF(2^{571})$, the calculation times of designs using the Karatsuba multiplier are also illustrated in Fig. 4.9, shown as the triangle line at the bottom, each with their specifications by their side. The bus type Karatsuba multiplier designs run as fast as the fastest DSM design. The mixed type designs calculate the Tate pairing in even a shorter time.

Fig. 4.10 shows the calculation time comparison between designs of different field size m. In this figure the minimum DSM design ($\#Mult$ = 1, d=1) and the maximum DSM design ($\#Mult = 7$, d=16) are presented along with the Karatsuba designs. It can be clearly seen from Fig 4.10 that for all m, the minimum DSM design is much slower than other designs, while the maximum DSM design has almost the same speed level as the bus type Karatsuba designs. The mixed type Karatsuba designs show the best performance and a significant improvement when compared to

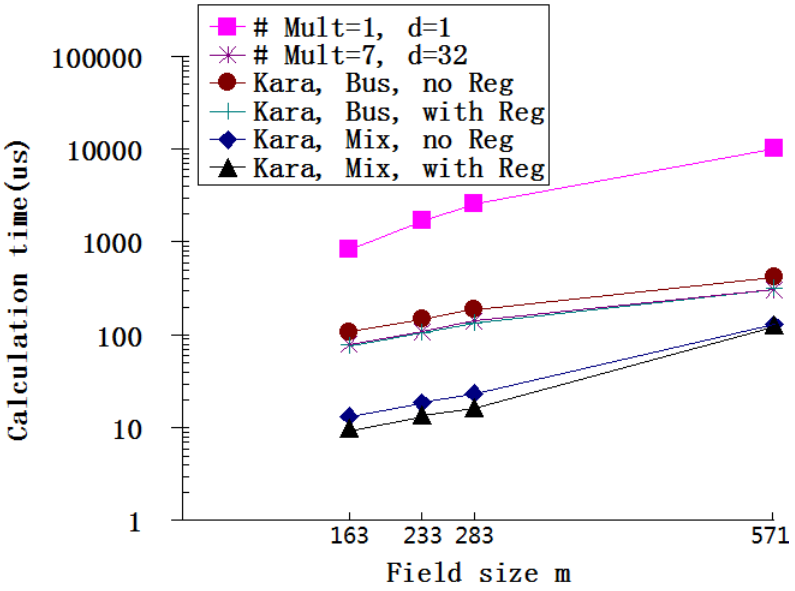


Fig. 4.10: Calculation time of η_T algorithm implementations for different field size m

that of using bus-type architecture. Since the multiplications are calculated in fixed time, rather than linear to field size m as the DSMs, the calculation times of the Tate pairings over different field size m are now decided by the number of iterations in the 'for' loop. i.e. the calculation times of the mixed type architecture designs grow at a linear rate with the field size m. Among the mixed type designs, the designs using Karatsuba multipliers with registers inserted give the best calculation speed for field size m=163, 233 and 283. For field size m=571, the design uses a 2^{nd} level Karatsuba multiplier, i.e. calculating 3 286-bit Karatsuba multiplications to form a 571-bit multiplication. The two additional clock cycles from the inserted registers in each 286-bit sub multiplication lead to 6 clock cycles in the 571-bit multiplication. Since the mixed type top-level architecture minimizes the calculation time of the 'small' operations in the 'for' loop, these 6 clock cycles result in a significant speed penalty, which cannot be compensated for by the increased operation frequency. Thus, the calculation time of the design using Karatsuba multipliers with registers inserted for field size m=571 is longer than that without registers inserted.

4.4.2 Area analysis

The areas required for designs over $GF(2^{571})$ are shown in Fig. 4.11. As the area required does not vary as much as the calculation time does, Fig. 4.11 uses linear scale. As can be clearly seen in Fig. 4.11, in designs using DSMs, the more multipliers

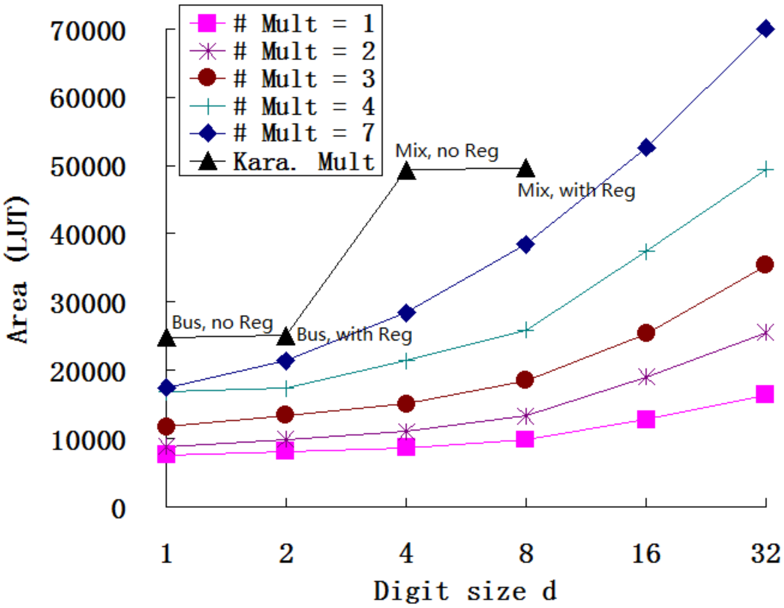


Fig. 4.11: Area of η_T algorithm implementations, m=571

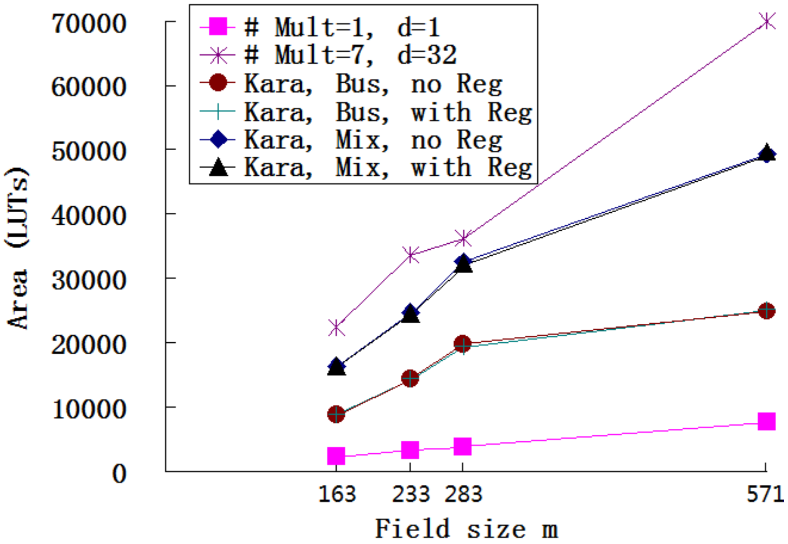


Fig. 4.12: Area of η_T algorithm implementations across different field size m

there are in the design, the more hardware resources are required. When using same $\#Mult$, using a larger digit size d leads to a larger area. Similar to the timing results, the area of the Karatsuba designs are presented in Fig. 4.11. Using Karatsuba in the designs requires more area than most of the DSM designs do. Among the Karatsuba designs, the mixed type designs are the biggest, almost as large as the maximum DSM design and twice as larger as the bus type Karatsuba designs.

Fig. 4.12 shows the comparison between designs of the same parameters over different fields. Larger field size always requires more hardware resources to implement the operations. As in the calculation time discussion, the area requirements of the minimum and maximum DSM designs, together with all the Karatsuba designs, are illustrated here. As shown in Fig. 4.12, the area requirements of implementing DSM designs are almost linear with the field size. This is because the sizes of all the components in the DSM designs, including the adder, squarer, DSM and divider are linearly proportional to the field size. The area requirement of the Karatsuba designs show an exponential increase with increasing field size m. This is because the area required for a Karatsuba multiplier is proportional to $n^2 \times 3^{\log_2 \frac{m}{n}}$, where n represents the digit size of the sub-multiplier block in the Karatsuba multiplier, as introduced in chapter 3. In the m=571 case, rather than the original Karatsuba multiplier, a 2^{nd} level Karatsuba multiplier is applied. This makes the bus type Karatsuba designs over field $GF(2^{571})$ only only slightly larger than those over the field $GF(2^{283})$. This additional area requirement comes from the 571-bit adder, squarer and divider. Fig. 4.12 also shows that the Karatsuba designs, with and without registers inserted, require almost the same number of LUTs.

*4.4.3 A*T product analysis*

The A*T trade-off performance of designs over $GF(2^{571})$ is shown in Fig. 4.13. As the product of two types of resources used in the implementations, a lower A*T product represents a higher hardware efficiency and a better trade-off between area and time.

For DSM designs, when the digit size d increases, the A*T products of the designs do not show a linear trend. The trends of A*T products for different $\#Mult$ values are not the same. For $\#Mult$=1 and 2 when d increases, the A*T product decreases. However, the slope decreases along with the increase of $\#Mult$. In the design using $\#Mult$=3, the A*T products for digit sizes d=16 and 32 are almost the same. For designs with $\#Mult$=4 and 7, the A*T products decrease to a minimum at d=8 and increase once again as d increases. i.e. in DSM designs, when $\#Mult$ increases, the

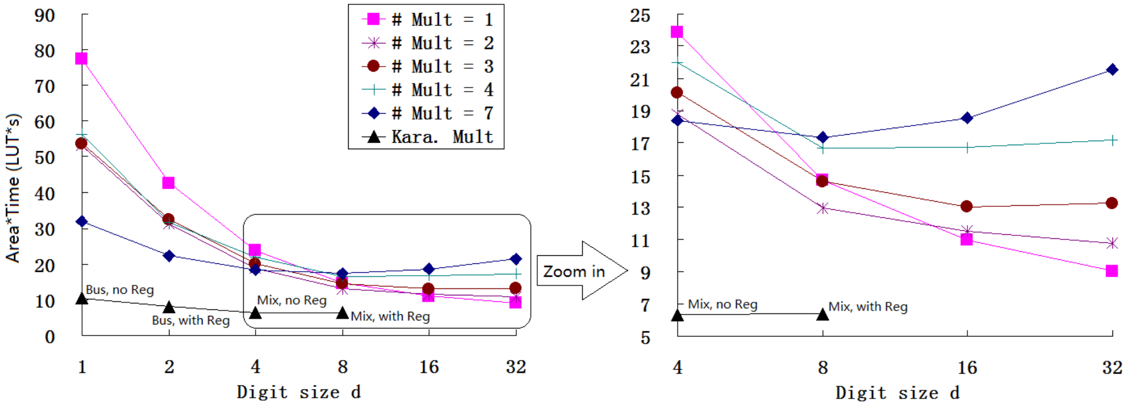


Fig. 4.13: Area*time (A*T) product of η_T algorithm implementations, m=571

effectiveness of improving the A*T product by increasing the digit size d decreases.

Fig. 4.13 also shows that all of the 4 designs using Karatsuba multipliers provide a good A*T product, being the same or better than the best A*T performance of DSM designs. Among all the Tate pairing implementations over $GF(2^{571})$, the 'Kara, mix, no reg' design, i.e. the mixed type design using the Karatsuba multiplier without registers inserted, shows the lowest A*T product, i.e. the best trade-off between area and time.

4.4.4 Comparison with earlier work

Table 4.5 compares the implementation results of the designs described in this chapter with earlier hardware based pairing accelerators. Note that in the table, the areas of Beuchat's work [11] are approximate values because the original area requirements were calculated in slice unit. The differences between Virtex 2, 4, 5 and 6 [82, 83, 84, 85] can be found Table C.1 in Appendix C. All works in Table 4.5 were implemented over $GF(2^m)$. The minimum DSM designs, maximum DSM designs and the mixed type Karatsuba designs, for each similar field size, are shown in the table.

The implementations designed by Shu et al [7] and Li et al [8] used the same algorithm as this work, i.e. the η_T algorithm. Both designs used a combinational logic top-level architecture and DSMs as the multiplier. Both works have their multi-DSMs pipelined, aiming to obtain a high calculation speed. Among them Shu acquired a better speed, faster than the maximum DSM designs in this work, but slower than the mixed type design using Karatsuba multiplier.

The work done by Keller [143] is an implementation of the BKLS/GHS algorithm.

Ref.	Field m	Mult type	VIRTEX Plaform	Area (LUTs)	Time (μs)	A*T (LUT*s)
Shu [7]	239	10×DSM, d=32	4	59971	43	2.58
Beuchat [11]	239	Kara. with reg	4	32406	3.46	0.112
This work	233	1×DSM, d=1	5	3200	1717.0	5.49
This work	233	7×DSM, d=32	5	33654	107.9	3.63
This work	233	Kara. with reg	5	24587	13.7	0.34
Keller [143]	271	1×DSM, d=1	2	3716	13500	50.2
Keller [143]	271	11×DSM, d=16	2	57032	839.7	47.9
Shu [7]	283	10×DSM, d=32	4	72961	61	4.45
Li [8]	283	12×DSM, d=32	4	104860	590	61.9
This work	283	1×DSM, d=1	5	3860	2658.5	10.26
This work	283	7×DSM, d=32	5	36312	141.0	5.12
This work	283	Kara. with reg	5	32345	16.5	0.53
Keller [143]	457	1×DSM, d=16	2	12406	4769	59.2
Keller [143]	457	11×DSM, d=6	2	47749	3389	161.8
Shu [7]	557	10×DSM, d=8	4	75862	675.5	51.2
Beuchat [11]	557	Kara. with reg	4	110312	13.2	1.46
This work	571	1×DSM, d=1	5	7598	10182.6	77.37
This work	571	1×DSM, d=16	5	12748	859.2	10.95
This work	571	7×DSM, d=32	5	69990	308.1	21.56
This work	571	Kara. no reg	5	49321	128.7	6.35
Ghosh [129]	1223	Kara. no reg	6	54681	190	10.4

Tab. 4.5: Comparison with other implementations of Tate Pairing

The calculation time of Keller's work is much longer than the maximum DSM design in this work. This is because the implementation uses an early technology (Virtex-2) which operate at a lower speed. In addition, the BKLS/GHS algorithm is not as efficient as the η_T algorithm in calculating the Miller's loop. However, the bus type top-level architecture of Keller's work is reconfigurable. By applying this architecture, the minimum DSM designs are implemented for all fields. For designs with strictly restricted hardware resource, such designs can accomplish the Tate pairing calculation using minimum area, at the cost of a long calculation time.

Designs by Beuchat et al in [11] and by Ghosh et al in [129] use the reduced η_T algorithm, which halves the 'for' loop in the original η_T algorithm, but introduces some more operations in the final exponentiation step. The reduced η_T algorithm is about 75% of the calculation time of the original η_T algorithm. Both designs in [11] and [129] also use combinational logic top-level architectures. In contrast with Shu's and Li's designs, the multiplier used in the works of Beuchat and Ghosh is the Karatsuba multiplier. This fully parallel structure provides a very short calculation time for the $GF(2^m)$ multiplication. Together with the fully combinational architecture, these two designs achieved a very fast calculation speed. Although the hardware resources required for the combinational logic top-level architectures and the Karatsuba multiplier are very large, the fast speed of these designs still makes the A*T products of such designs outperform the other designs. Note that the design by Ghosh et al in [129] is over the field $GF(2^{1223})$. This design is based on a very large finite field and provides a very high security level (128-bit security). In his design, because the original Karatsuba multiplier is much too big, a 3^{rd}-level Karatsuba multiplier is applied. Similar to the 2^{nd}-level Karatsuba structure, Ghosh used a 306-bit Karatsuba. By calculating 9 306-bit multiplications, plus the pre- and post- multiplication calculations, each 1223-bit multiplication requires 11 clock cycles to calculate.

4.5 Conclusions

This chapter showed the relevant operations over extension field $GF(2^{4m})$, including squaring, multiplication and inversion. How these extension operations can be performed using basic operations over field $GF(2^m)$ was presented. The $GF(2^m)$ operations required are (4**A**+2**S**), (34**M**+16**A**+4**S**+1**I**) and (9**M**+22**A**), respectively.

The η_T algorithm is implemented using different top-level architectures: the bus type and the mixed type architectures. In bus type architecture designs, both DSMs and Karatsuba multipliers are applied, while in mixed type architecture designs, only the Karatsuba multiplier is used for the pursuit of a fast calculation speed. In bus type designs, different numbers of DSMs were used along with designs of different digit size d = 1, 2, 4, 8, 16 and 32. As the Karatsuba multiplier is a fully parallel architecture and consumes large area, in both bus type and mixed type architecture designs, only one such multiplier is used. The implementation results are listed and the comparisons detween different parameters are discussed. The bus type architecture designs are of high flexibility and reduce the area of the design significantly. The reconfigurable designs using the bus type architecture provide the designer with different choices in resources constrained environments. In the mixed type designs, the operations in the 'for' loop are optimized for a faster speed. Although the area required for the mixed type designs is much larger than that of bus type designs, the calculation time of mixed type designs is much faster, even outweighs the area cost and gives the mixed type designs of the lowest A*T product, i.e. the best hardware efficiency.

The comparisons between different designs and with earlier works were presented. Among designs of different architectures, the combinational logic architecture provides the best hardware efficiency. The bus type architecture provides good flexibility in trade-offs between area and time rather than a fast calculation speed or a high hardware efficiency. In the choice of multipliers, DSMs can be reconfigurable and are suitable for design of different size. However, at the cost of a large area, Karatsuba multipliers are preferred when pursuing high speed designs.

5. SIDE CHANNEL ATTACKS AGAINST IMPLEMENTATIONS OF TATE PAIRING ALGORITHMS

5.1 Introduction

Attacks on hardware implementations of cryptosystems mainly focus on the mathematical analysis of leaked side channel information. Side Channel Analysis (SCA) attackers exploit leaked information in order to derive secret information. Whelan et al. [144] and Kim et al. [145] investigated the possibility of SCA attacks, including simple, differential and correlation power analysis, respectively SPA, DPA and CPA, against practical pairing algorithms. Former attacks making use of side channel information are introduced in Chapter 2. This work mainly focuses on practical CPA attacks against FPGA implementations.

Section 5.2 introduces an overview of SCA attacks. The theory of CPA attacks is specified and is used as the practical attack applied in this work. The power consumption model applied in this work is also analyzed here. Section 5.3 analyzes the weakness of the proposed η_T pairing algorithm. Making use of the ciphertext exposed in the insecure channel, both in the public key protocol and in the algorithmic calculations, is discussed. In section 5.4, practical CPA attacks are applied on the suspectable operation blocks, including the $GF(2^m)$ Adder block and the two multiplier blocks. The power traces collected from the attacks performed are shown.

5.2 Side Channel Analysis Attacks

Side Channel Analysis (SCA) attacks are based on the observation that the side channel information leaked from the cryptosystem is related to the instruction being executed. Among side channel attacks, power analysis attacks utilize the relationship between the data being manipulated and the power consumption leaked from the actual cryptographic devices, such as FPGAs and ASICs [13, 89]. Power analysis attacks typically consist of two steps. The first step is the collection phase, where the power consumption traces of the device are recorded. The second step is the analysis phase, where the power consumption traces acquired from the devices are analyzed to reveal secret information. The analysis method varies for different types of attacks. Successful analysis methods include simple power analysis (SPA), differential power analysis (DPA) [13] and correlation power analysis (CPA) [96]. SPA focuses mainly on deducing the operation being executed in the devices from the shape of the power consumption traces. In contrast, DPA focuses on the data being manipulated, using statistical methods to form a correlation between a number of power traces and the secret information. CPA is a more accurate variant of DPA that estimates a hypothetical power consumption for each possible value of the secret information and compares them to the actual power consumption traces using a correlation test, such as Pearson's correlation coefficient [146]. An actual cryptosystem always includes many components and the inherent noise of the components may hide the information the attacker is looking for. By taking more traces and averaging out the noise, an attacker can eliminate or reduce such effects. In SPA, an average of 1-50 traces may reveal the secret information because it considers only the profile of the traces [13]. While in a DPA and CPA attacks, the number of measured power traces typically varies from 500 to 10000 [13, 147, 148]. When choosing the number of power traces, using more power traces averages out the effect of noise and helps identify the highest correlation peak which indicates the correct hypothetical value.

5.2.1 Correlation Power Analysis Attacks

Based on the Hamming-Distance model, correlation power analysis (CPA) is a form of SCA attack that exploits the correlation between hardware power consumption and the intermediate values of the cryptographic algorithm in order to recover the secret information. With recorded power traces of the Tate pairing operation, such a power analysis attack can be successfully applied against the multiplier and adder units.

The Hamming-Distance model is based on the Hamming-Weight model [103]. In a hardware implementation of a cryptosystem, an m-bit register state at time t is represented as an m-bit binary data word $D_t = \Sigma_{j=0}^{m-1} d_j 2^j$, where $d_j \in \{0, 1\}$. The Hamming-Weight of the register is the number of elements that are equal to 1, i.e. $H(D_t) = \Sigma_{j=0}^{m-1} d_j$. The Hamming-Weight model is used as the basis for many power analysis attacks on software implementations [149]. Since H is an integer satisfying $0 \leq H \leq m$, if the data words D_t are independent, D_t has an average Hamming-Weight $\mu_H = m/2$ and a variance $\sigma_H^2 = m/4$.

The Hamming-Distance model [96] assumes that the side channel information leaked from a system depends on the number of bits switching from one state to the other and is more appropriate for hardware implementations. The basic Hamming-Distance model is:

$$W = aH(D_t \oplus D_{t+1}) + noise, \tag{5.1}$$

where *noise* encapsulates switching and electrical noise. D_t is the current state of a register and D_{t+1} is next state The scalar a is the gain between W the power consumption measured from actual cryptographic devices and H the Hamming-Weight of $(D_t \oplus D_{t+1})$ which represents the number of bits switched between register states D_t and D_{t+1}. This is called the Hamming-distance between D_t and D_{t+1}. In this model, D_t is usually a state in the register targeted by the attacker.

5.2.2 Relationship between intermediate variables and power consumption

The basic principle of CPA [96] is that there exists a relationship between the Hamming Distance of two register states and the measurable power consumption [105, 150]. The correlation factor between the Hamming Distance and the consumed power, named $\rho_{(W,H)}$, is used to tell whether the Hamming Distance model fits the real power consumption or not [148]. It is the covariance between the two variables H and W, normalized by the product of their standard deviations. With the Hamming Distance model, there exists:

$$\rho_{(W,H)} = \frac{cov(W,H)}{\sigma_W \sigma_H} = \frac{a\sigma_H}{\sigma_W} = \frac{a\sigma_H}{\sqrt{a^2\sigma_H^2 + \sigma_{noise}^2}} = \frac{a\sqrt{m}}{\sqrt{ma^2 + 4\sigma_{noise}^2}} \quad [96] \tag{5.2}$$

In equation 5.2, $cov(W,H)$ represents the covariance between H and W, σ_H and σ_W represent the standard deviation of H and W, σ_{noise}^2 represents the variance of the noise. Let E be the mean of an array, the standard deviation of array H can be calculated as per equation 5.3:

$$\sigma_H = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(H_i - E(H))^2} \tag{5.3}$$

And the covariance between H and W can be calculates as per equations 5.4.

$$cov(H,W) = (\sum_{i=1}^{N}\sum_{j=1}^{N}(E[(H_i - E[H])(W_j - E[W])])) \tag{5.4}$$

Assuming the noise has a Gaussian distribution, the variance of the noise σ_{noise}^2 tends to a finite constant. In practical power consumption, this finite constant is very small and tends to zero [96]. By applying a low pass filter or simply averaging a number of traces, the spikes of noise can be reduced and thus, the affects of the variance σ^2_{noise} can be minimized. In this work, each of the power traces used in attacking the multipiler modules are averaged from 20 raw power traces, while the power traces used in attacking the adder are filtered by a low pass filter with a cutoff frequency of 200MHz.

The relationship in equation 5.2 shows that $\rho_{(W,H)}$ can be used to determine the next state D_{t+1}. If the hypothetical value of D_{t+1} is correct, the value $\rho_{(W,H)}$ tends to ± 1 at the correlated point. In this work, the coefficient a is positive. Thus, $\rho_{(W,H)}$ in equation 5.2 tends to 1 in this case. In experiments, if an attacker knows the current state D_t and predicts the correct next state D_{t+1} of the target register at time t, there will be a high correlation value at the related point, otherwise the correlation values tend to 0, or other values, rather than 1.

If the model applies only to l independent bits amongst L, a partial correlation still exists:

$$\rho_{(W,H)_{l/L}} = \frac{a\sqrt{l}}{ma^2 + 4\sigma^2_{noise}} = \rho_{(W,H)}\sqrt{\frac{l}{L}} \tag{5.5}$$

This equation indicates that in the CPA attacks, the number of register bits being targeted affects the value of the correlation factor ρ. The more bits being targeted at a time, the larger the value of ρ the attacker gets. Theoretically, even if the ρ values are small, according to equation 5.2, the correct one also can be recognized from the incorrect ones, as long as the linear relationship between the power consumption and the hamming distance exists. However, since the ρ value is small and near the noise level, the attacker has to take more power traces and to average out the affect of the noise. Thus, in applying CPA attacks on the hardware implemented Tate pairing algorithm, the fewer bits being target at a time, the more power traces need to be taken.

5.2.3 CPA attack setup

The practical setup for the CPA attack experiment is shown in Fig. 5.1. The structure of the CPA attack setup in this work was illustrated in Fig. 2.10. In the experiment, the Tate pairing algorithm described in Algorithm 2.2 is implemented on a Xilinx Virtex-V xc5vlx50 technology [84] FPGA on the Sasebo-GII evaluation board [151]. There is a shunt resistor of 1Ω inserted on the core V_{dd} line of the cryptographic FPGA for measuring power traces. An oscilloscope is used to collect

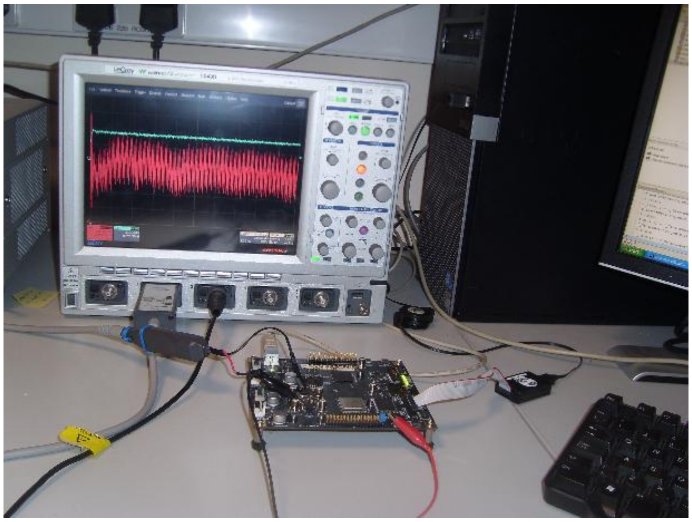


Fig. 5.1: CPA attack setup

and store traces of the voltage across this resistor. The voltage across this resistor linearly correlates with the current through the resistor, thus, it is also linearly correlated with the power consumption of the FPGA.

To minimise noise during thee capture of the power traces it is suggested that the following are used:

- differential probe
- stable power supply
- room with stable temperature
- control FPGA to clock the inputs to the crypto FPGA.

During the data acquisition stage when the voltage traces are recorded, Matlab [152] is used to communicate with the FPGA and the oscilloscope. Input points to the pairing algorithm are sent from the PC, through the control FPGA and then to the crypto FPGA. Following the transmission of the input information, the crypto FPGA calculates the Pairing and the oscilloscope measures and records the corresponding voltage traces which will be used in the CPA attack. Through Matlab, the power traces measured by the oscilloscope can be read back to and stored in the PC for analysis.

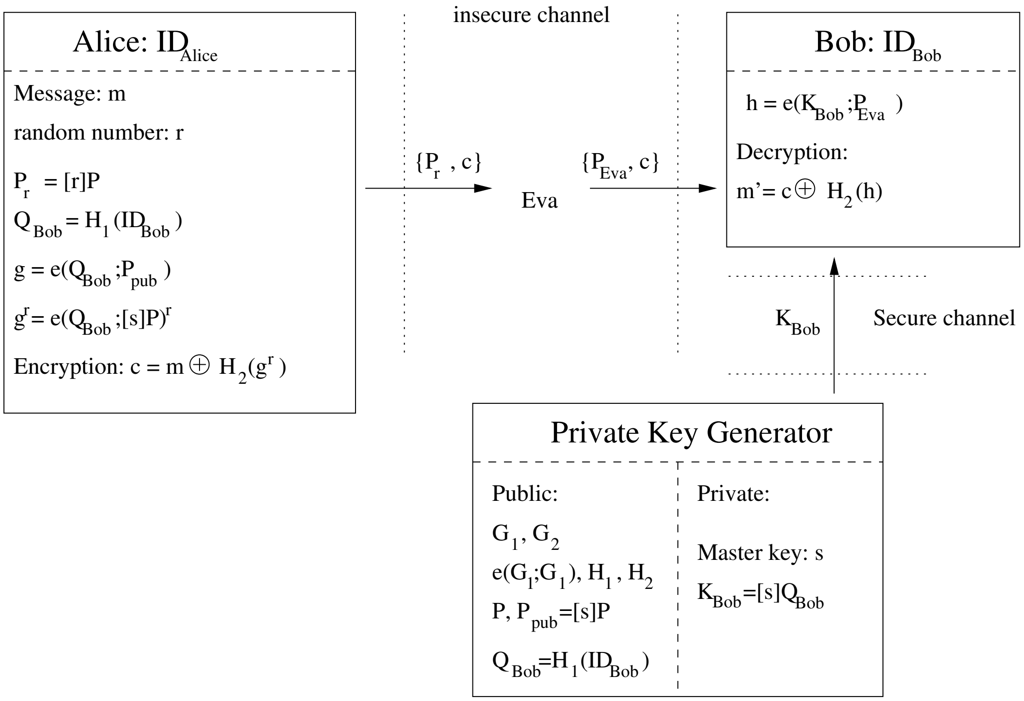


Fig. 5.2: Pairing based IBE scheme encounters eavesdropper

5.3 Side-channel security analysis of the η_T Pairings

5.3.1 Weakness of η_T pairing based IBE scheme

In the IBE system described in Fig. 2.7 and the ABE system described in Fig. 2.8, assume there is an eavesdropper and attacker Eva. Eva can record and tamper with the information in the insecure channel and can get the power consumption information leaked from the decryption operation of Bob. As was introduced in Chapter 2, when the information pair $\{U, V\}$ sent from Alice to Bob is exposed to the insecure channel, there can be unexpected weaknesses in the security of the system. In this section the information pair $\{U, V\}$ and the weaknesses are addressed for IBE and ABE schemes.

In an IBE scheme, the information pair $\{U, V\}$ consists of Alice's symbol P_r and cypher text c, i.e. $\{U, V\} = \{P_r, c\}$. Eva would keep cypher text c and tamper with element P_r to produce $P'_r = P_{Eva}$ for the attack, as shown in Fig. 5.2. On receiving the cypher text $\{P_{Eva}, c\}$, Bob calculates $e(P_{Eva}; K_{Bob})$ as the first step in the decryption calculation. Eva monitors the power consumption information leaked from Bob's decryption and tries to get information about Bob's private key K_{Bob}.

Recall the ABE system described in Fig. 2.8, there are multiple recievers. Alice publishes cypher text E' and public key set $\{E_1, E_3, E_4\}$. Except for the cypher

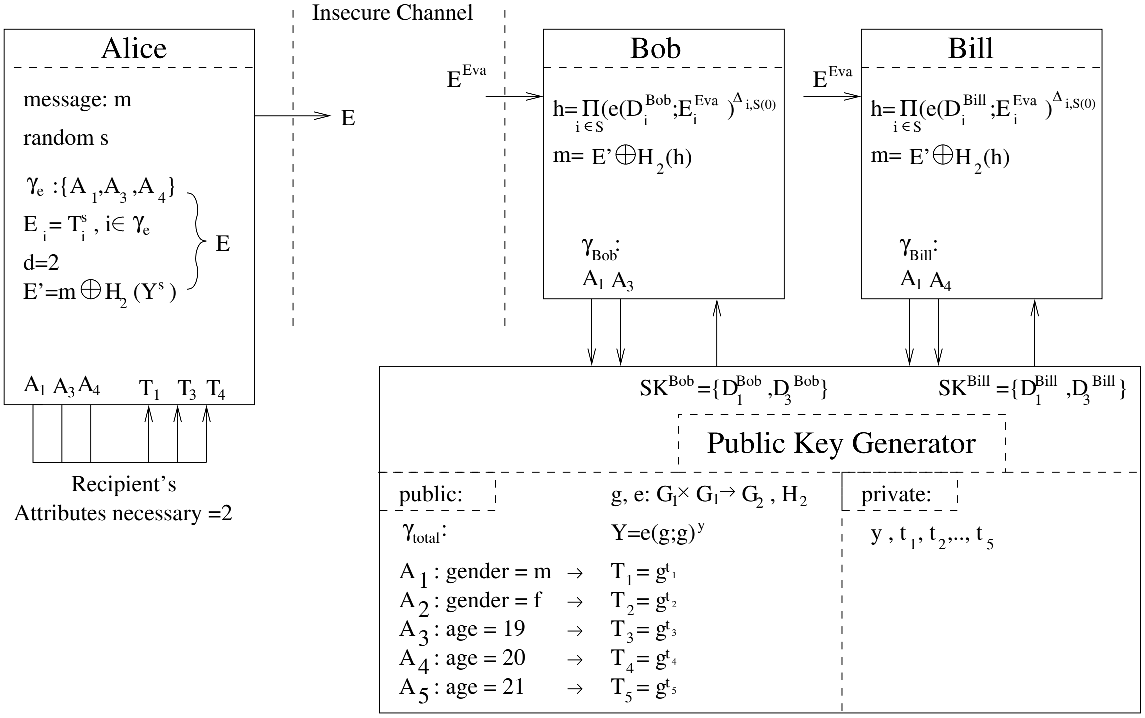


Fig. 5.3: Pairing based ABE scheme

text E', receiver Bob with attributes $\{A_1, A_3\}$ requires $\{E_1, E_3\}$ for decryption and Bill with attributes $\{A_1, A_4\}$ requires $\{E_1, E_4\}$ for decryption. In this work, Bob is taken as an example and Alice sends the information pair $\{(E_1, E_3), E'\}$ to Bob, i.e. $\{U, V\} = \{(E_1, E_3), E'\}$. Eva would keep cypher text $E_i = \{E_1, E_3\}$ and tamper with it to produce $E_i^{Eva} = \{E_1^{Eva}, E_3^{Eva}\}$ for the attack, as shown in Fig. 5.3. On receiving the information pair $\{(E_1^{Eva}, E_1^{Eva}), E'\}$, Bob calculates $e(E_1^{Eva}; D_1^{Bob})$ and $e(E_3^{Eva}; D_3^{Bob})$ as the first step in the decryption calculation. Eva monitors the power consumption information leaked from Bob's decryption and tries to get information about Bob's private key D_1^{Bob} and D_3^{Bob}.

Based on pairing algorithms, both IBE and ABE schemes can be insecure. For simplicity, the elements associated with the η_T pairing calculation and the attack are extracted from the IBE and ABE systems and listed as below:

1. An attacker can acquire and tamper with the information of P.
2. The attacker sends P to the hardware computing $e(P; Q)$.
3. The attacker observes the power consumption of the calculation $e(P; Q)$.
4. The attacker is trying to discover point Q.

P and Q are two points over the Elliptic Curve $E(GF(2^m))$, fully represented as $P(\alpha, \beta)$ and $Q(x, y)$. Since the elliptic curve $E(GF(2^m))$ restricts the two coordinates of a point $Q(x, y)$, knowing one coordinate leads to the other. Thus, when attacking the η_T pairing, an attacker only needs to focus on the x coordinate of the secret input $Q(x, y)$. In the η_T pairing described in Algorithm 2.2, there are several weak points vulnerable to a power analysis attack.

5.3.2 Weakness in addition

Recall the η_T algorithm for calculating the Tate pairing described in Algorithm 2.2 of Chapter 2, step 4 of Algorithm 2.2 is rewriten here as equation 5.6:

$$\text{step 4}: \quad A(t) \leftarrow \gamma + u + \beta + (\alpha + v)t + (\alpha + v + 1)t^2 \tag{5.6}$$

To indicate the difference, in this section α_{orig} is used to represent the original coordinate of input point P and α is used to represent the updated intermediate variable α in the 'for' loop. In the step shown in equation 5.6, the addition operation $(\alpha + v)$ contains the intermediate variable α from point P and v which is closely related with point Q. Assume the attacker is trying to attack operation $(\alpha + v)$ of the first iteration of the 'for' loop. It is assumed in this attack that one of these two elements can be controlled and known by the attacker. Two conditions may occur.

1. When point P is sent through the insecure channel as a public input, coordinate α of point P is known by the attacker. In this case, the attacker wants to reveal the information contained in coordinate x of point $Q(x, y)$, i.e. reveal the value of intermediate variable v because $x = \sqrt{v - 1}$.

2. When point Q is taken as the insecure input to the algorithm, value v is known by the attacker by calculating $v = x^2 + 1$. In this case, discovering α is the attacker's goal. Once the attacker acquires α, she can compute the information of input P via $\alpha_{orig} = \sqrt[4]{\alpha}$.

Note that square root operations [153] are required to reveal the secret information. If the attacker can break the $GF(2^m)$ Adder shown in Fig. 3.1 which calculates $(\alpha + v)$, she can reveal information about the private point with the intercepted public input information. The detail of how the private point information is revealed through the attack on the addition operation is introduced in Section 5.4.1.

In the bus type top-level architecture designs [143] illustrated in Fig. 4.5, inputs α and v to this addition operation are stored in the block RAM. When this addition is performed, the data bus presents these two values to the $GF(2^m)$ Adder. After one clock cycle, the result array $(\alpha + v)$ is read from the $GF(2^m)$ Adder block and written into the block RAM. In the combinational logic top-level architecture designs [7, 11, 8, 129] illustrated in Fig. 4.3 and 4.4, inputs α and v and output $(\alpha + v)$, are stored in three register arrays, each of m bits. In each iteration, the values of α and v are updated. Following the update of the inputs, the $GF(2^m)$ addition is calculated in the next clock cycle. i.e. after one clock cycle, the output $(\alpha + v)$ is updated. Thus, the weakness in the addition operation does not depend on the hardware architecture the designer chooses.

5.3.3 Weakness in multiplication

Recall step 2 and step 6 of Algorithm 2.2, rewriten here as equation 5.7 and equation 5.8 respectively:

$$\text{step 2}: \quad u \leftarrow x^2 + y^2 + g + \frac{m-1}{2}, v \leftarrow x^2 + 1, \alpha \leftarrow \alpha^4, \beta \leftarrow \beta^4, \gamma \leftarrow \alpha v \tag{5.7}$$

$$\text{step 6}: \quad u \leftarrow u + v, v \leftarrow v + 1, \alpha \leftarrow \alpha^4, \beta \leftarrow \beta^4, \gamma \leftarrow \alpha v \tag{5.8}$$

In these steps, the multiplication $\gamma \leftarrow \alpha v$ contains the coordinate elements v and α. Since the multiplications in the ‘for’ loop can be distributed in a multi-multiplier design, as described in Fig. 4.7 in Chapter 4, $\gamma \leftarrow \alpha v$ in step 6 is a more complicated condition and is chosen for attack in this work. The attacker can always generate the value of α in any iteration using the information of input P or alternatively generate the value of v in any iteration using the information of input Q. Thus, in this work, it is assumed that the attacker tries to reveal the secret information through CPA attacks on multiplication $\gamma \leftarrow \alpha v$ in step 6 of the first iteration of the ‘for’ loop. Similar to the addition calculation, two conditions occur here.

1. An attacker who controls point $P(\alpha, \beta)$ wants to reveal the value of v in this multiplication. Having acquired v in step 6, the attacker reveals the information of the secret input Q by calculating $x = \sqrt{v}$.

2. An attacker who controls point $Q(x, y)$ would calculate v first and wants to acquire information about α in this multiplication. Having obtained α in step 6 of the first iteration, the attacker reveals the information of the secret input P by calculating $\alpha_{orig} = \sqrt[8]{\alpha}$.

If the attacker can break the $GF(2^m)$ multiplication which calculates $\gamma \leftarrow \alpha v$, he can get the secret information of the private input. The detail of how the coordinate x of $Q(x, y)$ is revealed through the attack on the multiplication is introduced in Section 5.4.3.

Similar to the weak $GF(2^m)$ addition, the multiplication $\gamma \leftarrow \alpha v$ exists independently of the hardware architecture of the η_T pairing design. In bus type top-level architecture designs [143], inputs α and v are stored in the block RAM and are ready for reading by the data bus when the multiplication is underway. In combinational logic top-level architecture designs [7, 11, 8, 129], inputs α and v are stored in the register arrays and updated during each iteration. The register arrays are connected to the multiplier together with some other inputs. Multiplexors select which inputs are active for the current multiplication. When multiplication $\gamma \leftarrow \alpha v$ is performed, values α and v will be presented to the inputs of the $GF(2^m)$ Multiplier.

5.4 CPA attack against the $GF(2^m)$ operations

Because of the weaknesses of the algorithm, some attacks are proposed against the operation blocks applied in the FPGA implementation of the η_T pairing design. In the proposed attacks, since each element of the algorithms are of m bits, there are $\#E : \#E(GF(2^m)) = 2^m + 1 - T_r$ possible values for each of the elements. It is not possible for a hardware attacker to generate the correlation factor of all the $\#E$ possible values. A practical way is that the attacker tries to deal with a section of the secret information and deal with the rest in the same way if the attack works. In this condition, an attacker does not care about how big the field size is. He is always choosing a few bits, or even only one bit at a time, if the target element is of 163-bit size or 571-bit size.

5.4.1 CPA against addition: Condition 1) $P(\alpha, \beta)$ public, attacking $Q(x, y)$

As mentioned in Section 5.3.3, in step 4 of Algorithm 2.2, the operation $\gamma \leftarrow \alpha + v$ can be attacked to reveal information about the coordinate of the private point. Firstly, assume the input point $P(\alpha, \beta)$ is taken as the public input, known and controlled by the attacker. In this case, coordinate x and, thus, v is the secret information the attacker wants to reveal. The architecture of the adder is shown in Fig. 3.1 and the recorded power trace sample of an addition in the implementation is shown in Fig. 5.4. It takes 1 clock cycle to operate the addition. In this design, the Block-RAM structure is used. Thus, it will take another clock cycle to write the result back to

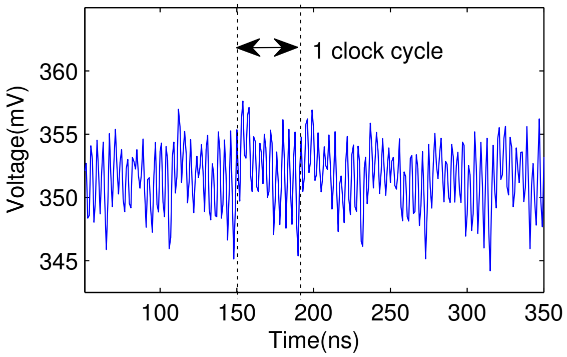


Fig. 5.4: Measured power trace sample of an addition operation in the Tate pairing

the memory. The chosen field for this attack is $GF(2^{571})$, with 7 DSMs each of digit size d=4. This is the largest design that can be mounted on the Virtex-V xc5vlx50 technology used in this work. To apply larger digit size over $GF(2^{571})$, the number of DSMs in the design must be reduced so that there are enough hardware resources for the implementation. Theoretically, according to equation 5.5, attacks on designs of larger area will be more difficult. Quadrupling the area of the whole design halves the expected correlation factor's value. However, according to equation 5.2, this effect can be compensated for by taking more power traces which reduces the effect of the noise.

In the η_T pairing implementation over $GF(2^m)$, the adder contains an m-bit XOR gate array and an m-bit register which stores the result after 1 clock cycle. Two values, α and v, are input to the XOR gates and the value in the output register is updated on the next clock cycle. Since an addition operation takes only 1 clock cycle, all information must be extracted during the single clock execution. With knowledge of α, an attacker can easily generate the hypothetical value of the secret input v. Thus, the attacker can choose how many bits of v to attack at a time. Attacking j bits at a time implies 2^j hypothetical values. In the attacks studied here, only 1 bit is processed at a time.

Let the i^{th} bit of the target variable v be $v(i)$ and let $\alpha(i)$ represent the i^{th} bit of variable α. The hypothetical values of $v(i)$ are '0' and '1'. To reveal $v(i)$, for each public input α, the following calculation are performed:

1. For each addition operation, the current state D_t of the register is '0'.

2. Always assume the target $v(i)$ is '1'.

3. For each α and v, calculate the hypothetical value of the next state of the

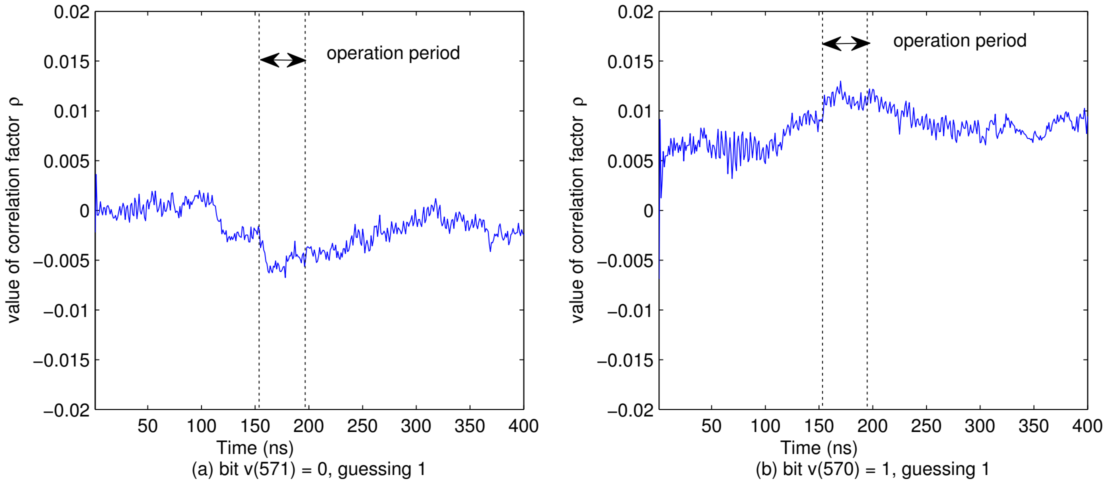


Fig. 5.5: Correlation with the $v(571)$ and $v(570)$ of the secret v

target bit of register $D_{t+1} = \alpha(i) \oplus v(i) = \alpha(i) \oplus 1$.

4. Calculate the hypothetical Hamming Distance values H between the current state ('0') and next state, namely $H = H(D_t \oplus D_{t+1}) = D_{t+1}$.

5. Calculate the correlations between the hypothetical Hamming Distance values H and the measured power traces W, namely $\rho_{(W,H)} = \frac{cov(W,H)}{\sigma_W \sigma_H}$.

Since the correlation results by guessing '1' and guessing '0' for the same target bit are exactly opposite to each other, it is only necessary to generate hypothetical values by guessing '1' or '0'.

When attacking the adder, the attacker is generating the hypothetical value of only 1 bit of the target m-bit '$(\alpha + v)$ Register'. According to equation 5.5, the correct correlation value of attacking an adder tends to a very small value, which is quite near the noise level. Thus, a large amount of power traces are taken for the correlation calculation. In attacking the adder in this design, N=50000 public inputs are sent to the FPGA as point $P(\alpha, \beta)$ to get a good correlation value which helps the attacker identify the correct hypothesis.

The correlation result of $v(571)$ and $v(570)$ are shown in Fig. 5.5. In the figure, the XOR operation happens between 160 ns and 200 ns. Since there are only 2 possible values for the target bit, one correct and the other incorrect, the correct correlation value is positive while the incorrect correlation value is negative. In Fig. 5.5(a), $v(571)$ is targeted. There is a drop in correlation at the operation period, which shows that the hypothetical value '1' of this point is incorrect, i.e. this bit

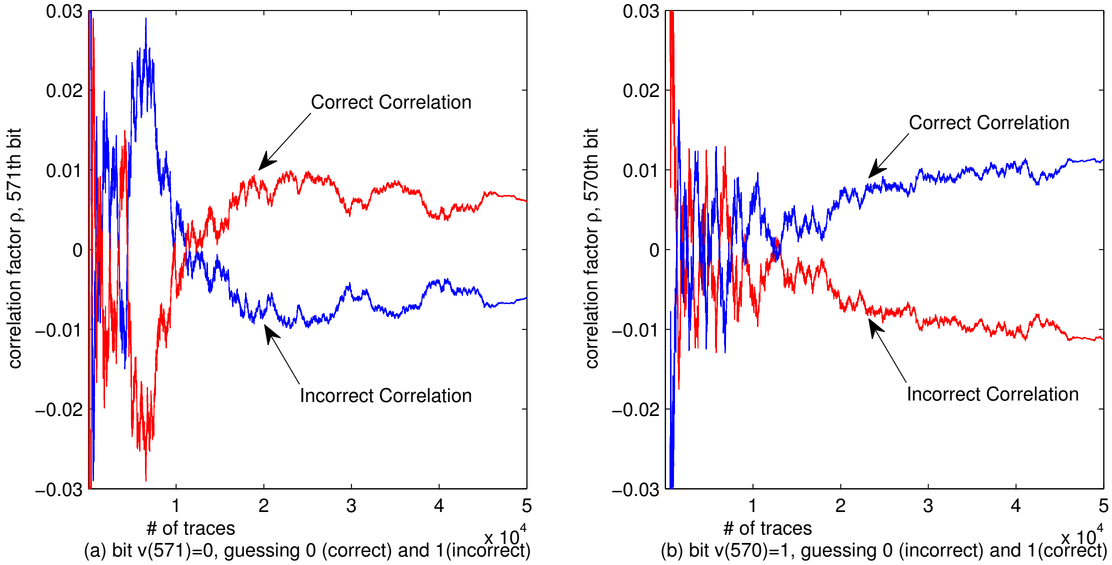


Fig. 5.6: Performance of Correlation factor ρ when more power traces are taken into consideration. Take $v(571)$ and $v(570)$ of the secret v as an example.

should be a '0'. Correspondingly, the correlation trace in Fig. 5.5(b) rises at the operation period, which shows the hypothetical value '1' of bit $v(570)$ is correct.

According to the CPA attack theory of equations 5.1 and 5.2 in section 5.3, by taking more power traces, the calculated correlation factor ρ tends to a constant value. This means that more power traces help distinguish the correct correlation value from the incorrect values. Fig. 5.6 shows the practical trends of the correlation factor ρ when the number of power traces taken into consideration increases.

In Fig. 5.6, the x-axis is the number of power traces taken into consideration whilst the y-axis represents the calculated value of Correlation factor ρ. As can be seen, in this attack, 50,000 power traces are used in the calculation of the correlation factor ρ. When the number of power traces taken is small, the correlation factor is very unstable. One cannot clearly recognize the correct correlation factor from the incorrect one. However, by taking more and more power traces into the calculation (more than 15,000 power traces in this attack), the correct correlation factor ρ (red) tends to a steady value which is obviously different from the incorrect one (blue).

The targeted hardware design of this attack contains 17,865 registers. According to equation 5.5, applying CPA attacks on only a 1-bit register against the noise of the whole FPGA design leads to a small correlation factor ρ: ρ tends to a value $\sqrt{\frac{l}{L}} = \sqrt{\frac{1}{17{,}865}} = 0.0075$. This value is not very accurate because I/O pads and other devices on the board might affect the power traces measured, but it shows the trend. This indicates that the correlation factor ρ tends to a very small value. Thus, the large quantity of power traces is necessary to help average out the affect of the noise. The correlation result shown in Fig. 5.6 shows that at least 15,000 power traces are necessary to attack a 1-bit register. In this attack, 50,000 power traces are taken to make sure that the correct correlation value can be clearly recognized from the incorrect one.

In this attack, the mathematical analysis is calculated in Matlab. The 50,000 power traces used to attack $v(571)$ and $v(570)$ can also be used to attack the remaining bits $v(569 \sim 1)$. By repeating the same analysis steps mentioned 571 times using the same set of power traces, the attacker is able to get value of v in step 4, described as in equation 5.6 and then calculate the secret information of point $Q(x, y)$, as described in section 5.3.2.

5.4.2 CPA against addition: Condition 2) $Q(x, y)$ public, attacking $P(\alpha, \beta)$

Because of the bilinearity property of the pairing algorithms (as introduced in Chapter 2), the result of the calculation $e(P; Q)$ is equal to that of $e(Q; P)$. This means that no matter which of the input points P and Q is being transferred in the insecure channel, the designer can switch the coordinates (x, y) and (α, β). In this case, the inputs of the operation blocks switch too.

Under this assumption, the point P is the private input and $Q(x, y)$ is the public input controlled by the attacker. The coordinate α of the $(\alpha + v)$ operation is the target to be revealed and v can be calculated using the public input x ($v \leftarrow x^2 + 1$). In this case the above attack still works. The only difference is that the attacker now holds the information of coordinate $v(i)$, generates the hypothetical value and calculates the correlation factor ρ of α on a bit-by-bit basis. Having acquired α, the attacker can calculate the original coordinate α_{orig} of point P, as described in section 5.3.2.

5.4.3 CPA against digit-serial multiplier(DSM): Condition 1) $P(\alpha, \beta)$ public, attacking $Q(x, y)$

The multiplier used to implement the multiplication operation in $GF(2^m)$ is also susceptible to power analysis attacks. Consider the first condition in which point $P(\alpha, \beta)$ is taken as a public input and the attacker wants to reveal the value of v. To attack the multiplication operation in the Tate pairing, pick the multiplication $\gamma \leftarrow \alpha v$ in step 6 of Algorithm 2.2, as analyzed in section 5.3.3.

A DSM is used in the Tate pairing architecture, as illustrated in Fig. 4.6. The structure of the multiplier is shown in Fig. 3.2. With two inputs a and b, the multiplier calculates the product of a and d bits of b during every iteration. Take $d = 4$ as an example, in this work the $d = 4$ bits of input b are referred to as a 4-bit word of b. The target register, 'z Register' in Fig. 3.2, stores the product of a and a 4-bit word of b and is updated during every iteration. The multiplication $c \leftarrow a \times b$ finishes in $n = \lceil \frac{m}{d} \rceil$ clock cycles. In this section it is always assumed that the input α is bound to input a of the DSM and v to input b. In Condition 1), $P(\alpha, \beta)$ is public, i.e. input a of the DSM is public. The alternative condition, in which $Q(x, y)$ is public, i.e. a private and b public, will be discussed in Condition 2).

As can be seen from Fig. 3.2, in the DSM calculation, the input signal goes through the multiplication block (the $\otimes$ block), 'reduction block a', 'xor' block, 'shifted z' block and 'reduction block b' and finally arrives at the 'z Register'. To generate the content in the 'z Register' using information about input a and b, the attacker has to generate the values of signals in each step of the DSM calculation. i.e. the attacker must generate the $m+d-1$ bit array after the multiplication blocks and then the m bit array generated by 'reduction block a' and so on until the m bit content in the 'z Register' is determined. To make sure the generation of the hypothetical values works correctly for each of the intermediate signals, this work applies CPA attack on a small DSM as the first experiment. If the attack works well for the small DSM, it will be applied on Tate pairing implementations with different DSM parameters and for different field sizes m.

An experiment on a 32-bit DSM of digit size $d = 4$

To apply a CPA attack on a DSM, initially consider a 32-bit DSM of digit size $d = 4$ (32/4 DSM). The 32/4 DSM is implemented on the Sasebo-GII board. A sample of the collected power traces of the DSM in operation is shown in Fig. 5.7(a). As can be seen, between the input and output operation which generate substantial noise, there are 8 peaks corresponding to the 8 clock cycles of the 32/4 DSM operation. For the structure of a DSM introduced in Fig. 3.2, during each iteration of the 32/4 DSM calculation, the 32 bit input a is known by the attacker. The other input, the 4-bit word of b, is unknown. As was introduced in section 5.3, if the Hamming Distance of the 'z Register' is correctly generated by the attacker, it will match the collected power traces and, thus, results in a high correlation factor ρ value that can be recognized from the noise level during the targeted clock cycle.

Thus, when attacking the 32/4 DSM, on keeping the same input b, N=1000 public inputs are sent to the a side of the DSM and, consequently, 1000 power traces are recorded. This number is picked based on former CPA experiments [13]. In the multiplication the most significant bits (MSBs) are first dealt with.

When attacking the first clock cycle of the multiplication, the following steps are taken:

1. For each of the $N = 1000$ public inputs, generate 2^4 hypothetical values of the 4 bits of input b by enumerating all possible values from '0000' to '1111'.

2. Current state D_0 of the 'z Register' is initialized as all '0'.

3. For each of the 2^4 hypothetical values of b, generate hypothetical values of the 35 bit array after the 'multiplication block' and then the 32 bits array after the 'reduction block a' and so on until the 32 bit array in the targeted 'z Register', named as D_1.

4. Calculate the hypothetical Hamming Distance of the value in 'z Register' between current state D_0 and next states D_1 for each hypothetical value of the 'z Register' state, namely $H = H(D_0 \oplus D_1)$.

5. Calculate the correlations between the hypothetical Hamming Distance values H and the measured power traces W, namely $\rho_{(W,H)} = \frac{cov(W,H)}{\sigma_W \sigma_H}$.

As all the bits in the 'z Register' are set to '0' before the multiplication starts, there is no need to generate the values of the 'z Register'. The hamming distance between current state D_t and next state D_{t+1} is equal to the hamming weight of the next state D_{t+1}.

The correct correlation result of the 1^{st} clock cycle is shown in Fig. 5.7(c). A peak can be clearly seen at the time point where the 1^{st} clock cycle of multiplication happens. For the incorrect hypothetical values, for example an incorrect correlation of the 1^{st} clock cycle shown in Fig. 5.7(b), the correlation value is constant. This makes it easy to distinguish the correct guess from the incorrect ones.

By correctly guessing the first 4-bit word of input b, the attacker now can make sure the status of the 'z Register' at the 1^{st} clock cycle of multiplication, i.e. the attacker now knows D_1. With this knowledge, the attacks on the $2^{nd} \sim 8^{th}$ 4-bit words of input b can be performed. The attacks on the remaining 4-bit words of input b are basically the same as attacking the first 4-bit word. The only difference is the generation of D_t. When attacking the first 4-bit word, the current status D_t of the 'z Register' is all '0'. However, when attacking the remaining 4-bit words,

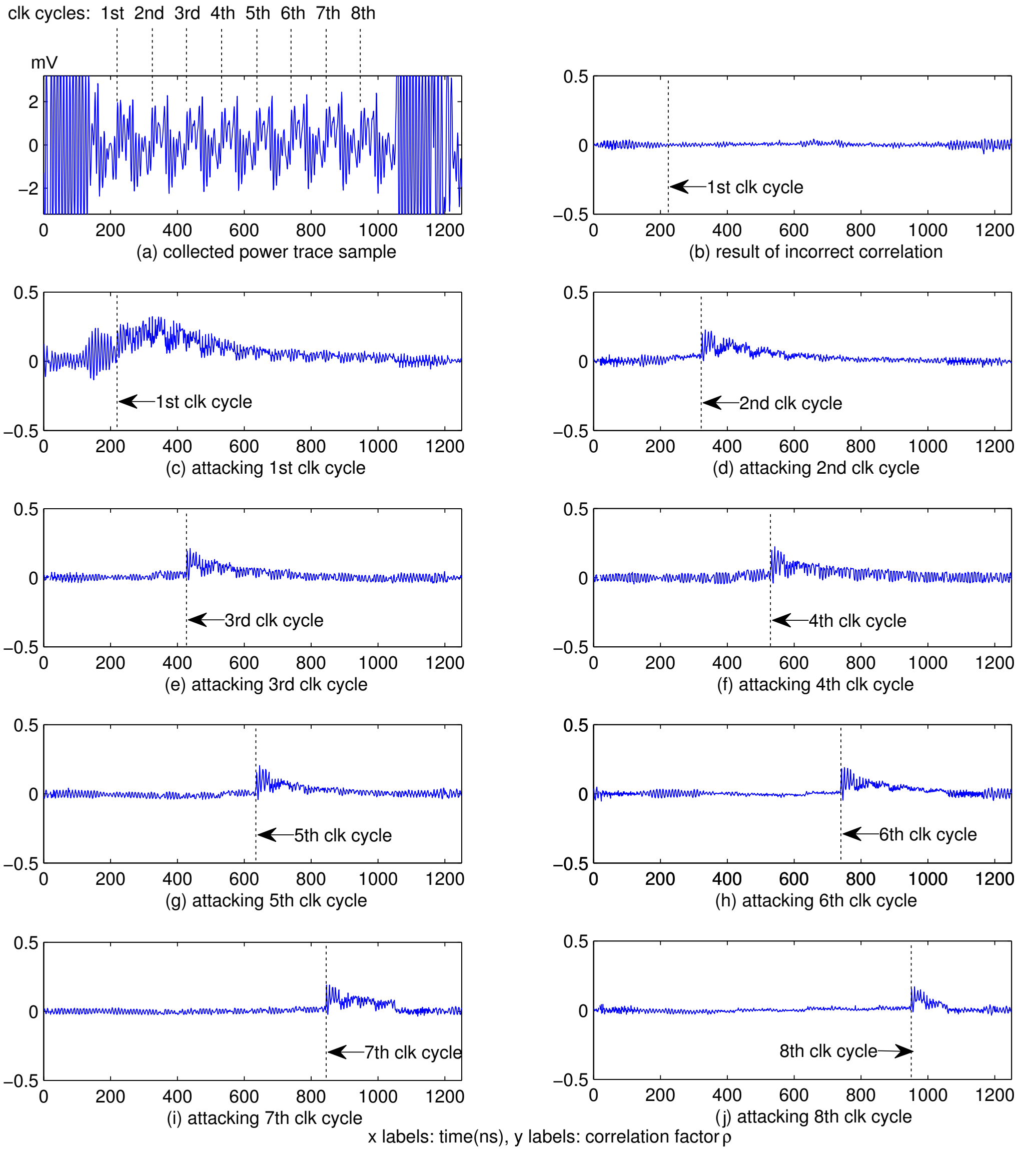


Fig. 5.7: CPA attack: Correct and incorrect correlation factors Vs. measured power traces of a simple DSM, $m = 32$, $d = 4$, a public, attacking b

the attacker must generate the correct current status D_t according to the successful attacks on the previous 4-bit words. Fig. 5.7 shows the correct correlation results using 1000 power traces. As can be seen from Fig. 5.7, when attacking each clock cycle of the DSM, the correct correlation trace shows a peak at the related timing point. By repeating the attack described above, the value of b of the 32/4 DSM can be revealed by the attacker.

The attack above is applied on a small DSM, which has the same structure as the normal DSMs used. The successful attack showed that attacking a DSM can

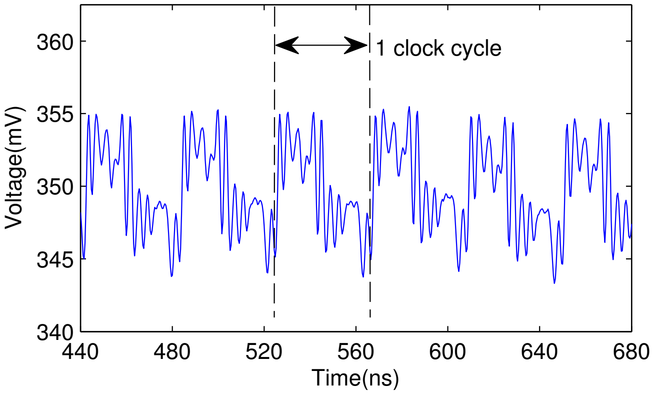


Fig. 5.8: Measured power trace sample of the Tate pairing when 7 digit serial multiplierare operating in parallel

be divided into attacking the 'z Register' $n = \lceil \frac{m}{d} \rceil$ times. When attacking each clock cycle of the digit-serial multiplication, knowledge of the previous d-bit word is necessary. It should be noted that, the power traces used to attack the 1^{st} d-bit word can be reused when attacking the 2^{nd} d-bit word, i.e. the power traces can be reused in attacking the whole DSM.

An experiment on an η_T pairing over $GF(2^{163})$, with 7×DSM of digit size $d = 4$

In this experiment, pick a DSM which is used inside an η_T pairing system over $GF(2^{163})$. This η_T pairing design uses 7 multipliers with all 7 multiplications performed in parallel, as shown in Fig. 4.7. This makes the design a large area implementation with a large amount of routing and complex control. Using 7 multipliers reduces the calculation time but increases the difficulty of a successful attack. In this case, in equation 5.1, scale a is comparatively small and $noise$ becomes very large. This is a much harder configuration to attack than a simple 32/4 DSM. A power trace sample of the multiplication operation is shown in Fig. 5.8.

In the design of the Tate pairing using 7 multipliers, there are 5,187 registers in total. When attacking the multiplier, the attacker generates the hypothetical value of all 163 bits of the target 163-bit 'z Register' with knowledge of input α and all possible values of the 4-bit word v. Thus, 2^4 hypothetical values of the 163-bit 'z Register' are generated while attacking each 4-bit word of v. Among all the 7 multipliers in this design, one is processing the multiplications $\gamma \leftarrow \alpha v$ in step 6 which is under attack, while the remaining 6 are processing the multiplications in $C(t) \leftarrow C(t)^2 \times A(t)$ in step 5 of Algorithm 2.2. All 7 multipliers and other logic cells, which consist of the lookup tables, registers and wiring between slices, consume power when operating. The power consumption of these components can be treated as *noise*, as described in equation 5.1. According to equation 5.5, the correct correlation value of attacking a multiplier tends to $\sqrt{\frac{l}{L}} = \sqrt{\frac{163}{5{,}187}} = 0.18$.

In the attack, N=1000 public inputs are sent to the FPGA as point $P(\alpha, \beta)$ and, thus, 1000 power traces are recorded for the correlation calculation. As the CPA attack using N=1000 power traces reveals the secret information, taking more power traces is not necessary. The multiplier operates on 4 bits of the input v in all iterations, except for the first. In the first iteration, since the targeted field size is m = 163, the multiplier deals with one bit '0' as the most significant bit and then the first 3 bits of v.

For each clock cycle of the attack, the following steps are taken:

1. For each of the $N = 1000$ public inputs, generate 2^4 hypothetical values of the 4 bits of input v by enumerating all possible values from '0000' to '1111'.

2. Generate hypothetical values of all 163 bits in the target register 'z Register' for the current state D_t and the next state D_{t+1}.

3. Calculate the hypothetical Hamming Distance of the value in 'z Register' between current state D_t and next states D_{t+1} for each hypothetical value of the 'z Register' state, namely $H = H(D_t \oplus D_{t+1})$.

4. Calculate the correlations between the hypothetical Hamming Distance values H and the measured power traces W, namely $\rho_{(W,H)} = \frac{cov(W,H)}{\sigma_W \sigma_H}$.

Fig. 5.9 shows the CPA attack against one of the 7 multipliers of the Tate pairing algorithm design over field $GF(2^{163})$. It can be clearly seen from the figure that the correlation factor of the correct hypothetical value of the 4-bit word tends to 0.18 as calculated above, while all the incorrect ones are much smaller and tend to zero.

For better analysis of the the attack, take the real values in the CPA attack as an exmaple here. In this attack, the four most significant 4-bit words are '(0)101 0111 1000 1001'. Assuming the three most significant 4-bit words of v '(0)101 0111 1000' are correctly guessed, the hypothetical value of the next 4-bit word is one of '0000', '0001', ... , '1111'. The attacker generates the correlation values of the next multiplication state according to the hypothetical values. Fig. 5.9 shows the correlation of all 2^4 predictions of the next state. The correct prediction of '1001' of the 4-bit word results in a correlation value much higher than other incorrect guesses.

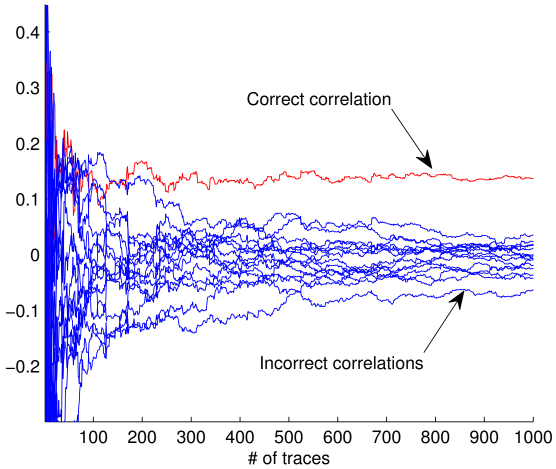


Fig. 5.9: CPA attack: Correct and incorrect correlation factors vs. # of power traces of Tate pairing, m=163, 7×DSM design, d=4, α public, attacking v

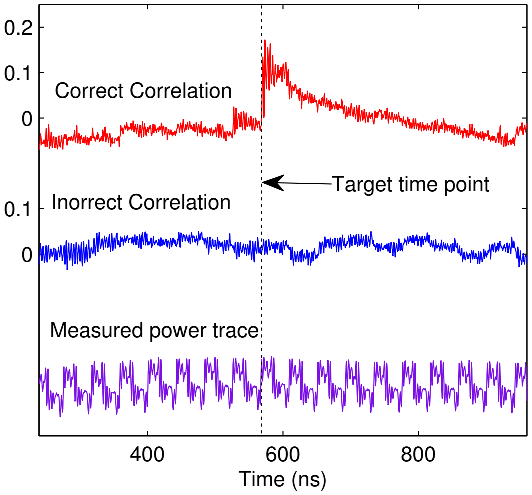


Fig. 5.10: CPA attack: Correct and incorrect correlation factors Vs. measured power traces of Tate pairing, m=163, 7×DSM design, d=4, α public, attacking v

For better analysis, an attacker may want to see how the correlation analysis works with other time points in the power traces. Here the traces of all time points rather than the target state only are presented. Fig. 5.10 shows the correct correlation and an incorrect correlation. In the correct correlation trace, there is a peak at the target state, while the incorrect correlation trace corresponds to noise. The correct correlation peak doesn't drop instantly after the clock's rising edge but decays in the next one to two clock cycles because of the discharge of the transistors in the FPGA takes time. A successful CPA attack shown in Fig. 5.9 determines 4 bits of the secret text each time. By doing the same attack n times, $n = \lceil \frac{m}{d} \rceil = \lceil \frac{163}{4} \rceil = 41$,

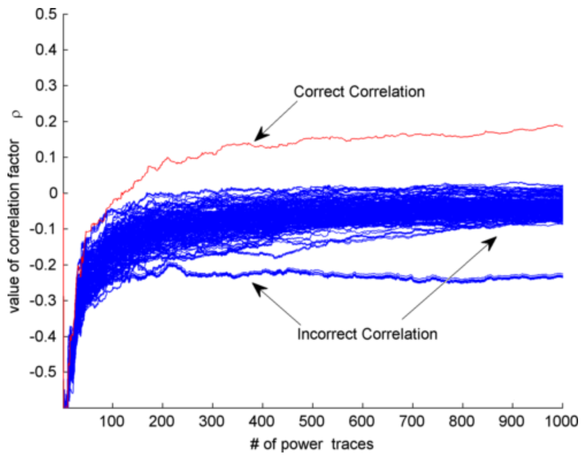


Fig. 5.11: CPA attack: Correct and incorrect correlation factors Vs. # of power traces of Tate pairing, m=571, 4×DSM design, d=8, α public, attacking v

using the same set of collected power traces, the secret text v can be recovered and, thus, the secret input $Q(x, y)$ can be revealed.

An experiment on a η_T pairing over $GF(2^{571})$, with 4×DSM of digit size $d = 8$

A more complicated pairing system is proposed here for further discussion: a pairing over field $GF(2^{571})$, with a multiplier of digit size $d = 8$. In this case, the input becomes 8 bits and, consequently the number of possible values of the input digits is $2^8 = 256$. The same steps as the previous attack were taken and the correlation results are shown in Fig. 5.11. As can be expected from the above attacks, increasing the digit size increases the number of hypothetical values of the input digits. The computational complexity of attacking the digit-serial multiplier is therefore increased. It can be predicted that when digit size d increases to a very large value, it can be infeasible to solve the DSM using the attack proposed in this section. However, calculating more hypothetical values of the input digits does not require more power traces. As long as $d < m$, this digit-serial structure substantially decreases the security level of the pairing computations.

5.4.4 CPA against digit-serial multiplier(DSM): Condition 2) $Q(x, y)$ public, attacking $P(\alpha, \beta)$

As mentioned in section 5.4.2, in the alternative condition, the inputs alter.

Consider the multiplication $\gamma \leftarrow \alpha v$ in step 2 of Algorithm 2.2 over the field $GF(2^m)$ once again. The attacker holds the information of v and tries to reveal α. As introduced in the DSM structure, input v is stored in a shift-register. In each

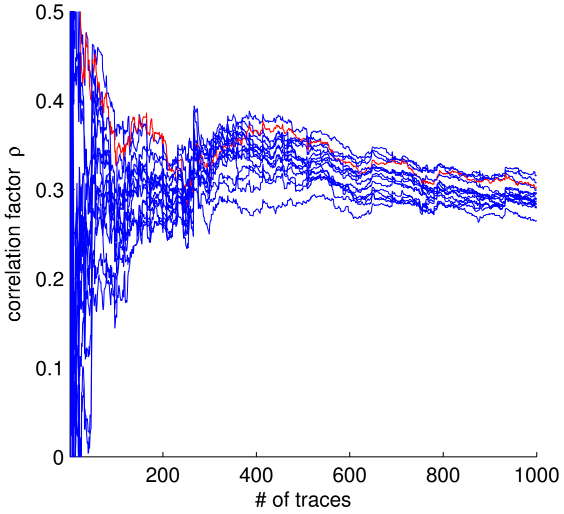


Fig. 5.12: CPA attack: Correct and incorrect correlation factors Vs. # of power traces of Tate pairing, m=163, 7×DSM design, d=4, v public, attacking α

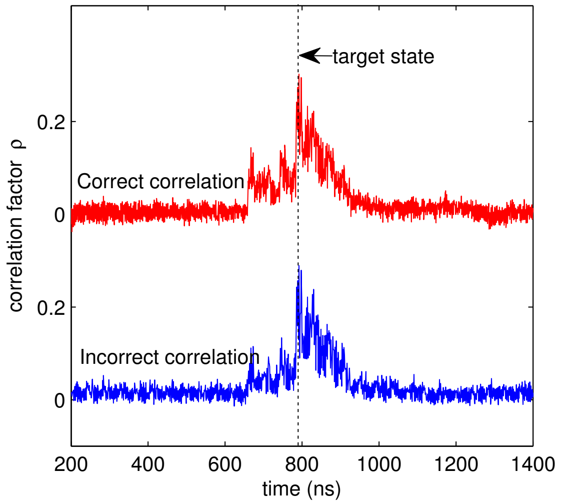


Fig. 5.13: CPA attack: Correct and incorrect correlation factors vs. real time plot of Tate pairing, m=163, 7×DSM design, d=4, v public, attacking α

clock cycle the most significant 4 bits of v are sent to the multiplier and are multiplied with all of the 163 bits of α. Note that here the attacker controls the information of all 163 bits of element v. However, in the first condition, the attacker can generate the hypothetical values of the 'z Register' for all possible secret inputs (2^4) in each clock cycle. In the second condition the secret input is sent to the multiplier once as m bits. It is not possible to generate all the hypothetical values of the 'z Register'.

In a CPA attack under this condition, the least significant 4 bits of input α are targeted. The $2^4 = 16$ distinct values of these 4 bits are generated. All the other bits of α are assumed to be '0' because of the lack of information about the other bits of the private input α. Then, 16 hypothetical states for the next state of the 'z Register' are generated. In this condition, the CPA attacker can only treat the affect of all other bits of α as noise. If this assumption is true, the effect of the other bits average out and the previous CPA attack still works under this condition.

When attacking the least significant 4 bits of input α, the following steps are taken:

1. For each of the $N = 1000$ public inputs, generate 2^4 hypothetical values of the least significant 4 bits of input α by enumerating all possible values from '0000' to '1111'. Assume all other bits of α are '0'.

2. Make use of the first and second 4-bit word of the public input v, i.e. $v(163 \sim 160)$ and $v(159 \sim 156)$, generate hypothetical values of all 163 bits in the target register 'z Register' for the state of the first clock cycle D_1 and the second clock cycle D_2.

3. Calculate the hypothetical Hamming Distance of the value in 'z Register' between first state D_1 and second states D_2 for each hypothetical value of the 'z Register' state, namely $H = H(D_1 \oplus D_2)$.

4. Calculate the correlations between the hypothetical Hamming Distance values H and the measured power traces W, namely $\rho_{(W,H)} = \frac{cov(W,H)}{\sigma_W \sigma_H}$.

The 16 resulting correlation traces are shown in Fig. 5.12 and Fig. 5.13.

Fig. 5.12 shows the correlation values at the target time point. In contrast with the previous attack shown in Figs. 5.7, 5.9 and 5.11, all correlation values at the time point tend to a high value rather than only the correct correlation value being high and incorrect correlation values being low. Also, Fig. 5.13 shows that not only the correct, but also the incorrect correlation traces, show a peak at the target state. This is because the hypothetical values the attacker generates in this case not only correlate with the lowest 4 bits of private input α, but also correlate with other bits of α. For example, if the least significant 8 bits of α are '0110 1011', the power traces measured will result in a high correlation value with the hypothetical value generated by 4 bits of '1011' as well as with '0110' and '1101' etc. Therefore an attacker cannot distinguish between the correlation results of the correct and incorrect predicted values of any 4 bits of input α. The unsuccessful attack shows that by carefully choosing the input to the DSM, the secret information in the multiplication of the η_T pairing algorithm can be protected from CPA attack.

5.4.5 CPA against Karatsuba multiplier

A Karatsuba multiplier can also be used to implement the computations over $GF(2^m)$ in the Tate pairing algorithms. The original Karatsuba multiplier is difficult to attack successfully. However, a variant of the Karatsuba multiplier, mentioned in [9] and discussed in section 3.5.3, which improves the critical path by inserting registers makes the Karatsuba multiplier insecure. Here a correlation attack against the Karatsuba multiplier with registers inserted is proposed.

As discussed in section 3.5.3, the Tate pairing designs over different field sizes uses different schemes of sub-multiplier blocks, as listed in Table 3.4. In this attack, an implementation of the η_T pairing over $GF(2^{283})$ is targeted. It uses a 283-bit short-critical-path Karatsuba multiplier with $n = 18$ bits sub-multiplier blocks, as described in section 3.5.3. A total of 81 sub-multiplier blocks are needed, as shown in Fig. 3.4. Among these 81 blocks, 16 blocks are directly related to the two 283-bit input arrays. Knowing the inputs of these 16 blocks leads to knowing the information of the input arrays to the Karatsuba multiplier. As in the previous attacks, here the multiplication $\gamma \leftarrow \alpha v$ in step 6 of Algorithm 2.2 is once again chosen as a target. In these sub-multiplier blocks, 'Sub-m1' calculates the least significant bits of the two inputs, i.e. $\alpha(17 \sim 0)$ and $v(17 \sim 0)$. This multiplication results in a 35-bit array $\gamma^*(34 \sim 0)$ which is stored in register 'post1' and will take part in the addition and reduction operations in the next clock cycle.

As was the case when attacking the DSMs, here the attacker is assumed to know one of the inputs into the Karatsuba multiplier, α or v and is trying to reveal the other. The two conditions ($P(\alpha, \beta)$ public or $Q(x, y)$ public) are discussed as follows.

5.4.6 CPA against register inserted Karatsuba multiplier: Condition 1) $P(\alpha, \beta)$ public, attacking $Q(x, y)$

In this condition, the attacker is assumed to be controlling the input α from public input $P(\alpha, \beta)$ and is trying to reveal the information in v. When attacking the Karatsuba multiplier, N=1000 public inputs are sent to the FPGA as point $P(\alpha, \beta)$ and 1000 power traces are recorded. In the Karatsuba multiplication, all sub-multiplications operates synchronously. When attacking the current sub-multiplication, e.g. when attacking $v(17 \sim 0)$, the operations in other sub-multiplier

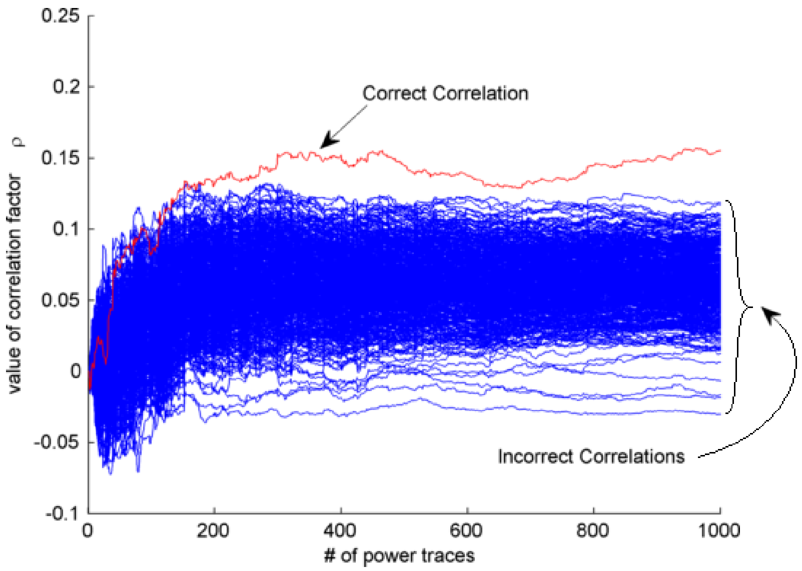


Fig. 5.14: CPA against Karatsuba multiplier in η_T pairing over $GF(2^m)$

blocks are treated as noise.

To attack the Karatsuba multiplication, the attacker does the following steps:

1. For each of the $N = 1000$ public inputs, generate $2^{18} = 262144$ hypothetical values of the 18 bits of input $v(17 \sim 0)$ by enumerating all possible values.

2. Generate hypothetical values for each of the 35 bits in the target register 'post1' for the current state D_t.

3. Load the value of the last state D_{t-1} in register 'post1', which was stored in the previous calculation.

4. Calculate the hypothetical Hamming Distance of the value in register 'post1' between current state D_t and previous states D_{t-1} for each hypothetical value of the 'post1' state, namely $H = H(D_t \oplus D_{t-1})$.

5. Calculate the correlations between the hypothetical Hamming Distance values H and the measured power traces W, namely $\rho_{(W,H)} = \frac{cov(W,H)}{\sigma_W \sigma_H}$.

6. If the hypothetical value proves to be correct, store the state D_t for the calculation of the next attack.

It can be clearly seen that the correlation of the correct hypothetical value is higher than most of the other correlation values. But it must be noted that, in this attack, because the result $\gamma^*(34 \sim 0)$ did not go through a reduction block, if the first bit of $v(17 \sim 0)$ is '0', i.e. if $v(17) = 0$ and, at the same time, the current state D_t is all '0', the correlation result of $v(17 \sim 0)$ will be the same as that of $v(17 \sim 0) << 1$, where $<< 1$ indicates a one bit left shift. The same case happens when $v(0) = 0$, the correlation values of $v(17 \sim 0) >> 1$ results in the same as that of $v(17 \sim 0)$, where $>> 1$ indicates a one bit right shift. This means that when guessing each 18-bit part of input v, the correlation attack locks the REAL value between several possible values. Let x represent the possible number of bits the REAL value can shift, where $1 \leq x \leq 17$.

For example, in the case where the possible values are:

011111111111111111 and 11111111111111110,

these two values can result in the same Hamming-Distance in the sub-multiplication and these two values can be seen as at most one bits shift of each other i.e. $x = 1$. Similarly in the case where the possible values are:

001111111111111111, 011111111111111110 and 111111111111111100,

these three values can result in the same Hamming-Distance in the sub-multiplication and $x = 2$ in this case.

There is a 25% possibility that $x = 1$ happens, a 12.5% possibility that $x = 2$ happens and etc. Thus, the mean value of x can be calculated as per equation 5.9.

$$\bar{x} = \sum_{x=1}^{n-1} \frac{1}{2^{x+1}} \times x \approx 1 \tag{5.9}$$

Equation 5.9 is considered as a good approximation for $n > 8$. As discussed in section 3.5.3, the sub-multiplier blocks are suggested to have bitwidth n of around 20, which fit equation 5.9. This indicates that in the Karatsuba multiplier, on average, the Real values shift within one bit. Thus, in attacking each sub-multiplier block, the REAL value of the secret information can be locked between two possible values.

This means that when attacking an η_T pairing system of field size m, sub-multiplier block size n, the correlation power analysis attack will lock the REAL information of the secret input between $2^{\lceil \frac{m}{n} \rceil}$ values. For example, an η_T pairing system with $m = 283$ and $n = 18$, the CPA attack locks the secret information between 2^{16} values. This greatly decreases the security value of the Karatsuba multiplier.

It can be seen from the above attack that the CPA attacks on the Karatsuba multiplier are basically attacks on the sub-multiplier block units in the Karatsuba multiplier architecture. For different field size m, the bitwidth n and the number of sub-multiplier blocks can be different. In the first condition when m increases, if the number of sub-multiplier stays the same, the bitwidth n must increase. The larger the bitwidth n of the sub-multiplier block, the more hypothetical values (2^n values) the attacker has to generate and, thus, the harder it is to perform CPA attacks. In the second condition when m increases, if the number of sub-multiplier increases, the bitwidth n can stay the same or decrease. No matter how n changes, more sub-multiplier blocks will always lead to a wider lock range of the REAL information an attacker can acquire. Thus, the same attack can be performed on designs of the same architecture over different fields. The difficulty of performing such an attack increases when the field size increases.

5.4.7 CPA against register inserted Karatsuba multiplier: Condition 2) $Q(x, y)$ public, attacking $P(\alpha, \beta)$

As in previous cases, the designer can choose to alter the two input points P and Q of the Tate pairing design. Here the attacker controls input point $Q(x, y)$, i.e. v and tries to reveal information about point $P(\alpha, \beta)$, i.e. α. Coordinate α of the $\gamma \leftarrow \alpha v$ operation is the target to reveal and v can be calculated using the public input x (calculate $v \leftarrow x^2$). In this case, the above attack still works. Take the 'Sub-m1' which calculates the least significant 18 bits of v and α as an example. The only difference from Condition 1) is that the attacker now holds the information of coordinate $v(17 \sim 0)$, generates the hypothetical values and calculates the correlation factor ρ for all possible values of $\alpha(17 \sim 0)$. By applying CPA attacks, the attacker locks the real value of α in a small range. On average, there are 2^{16} possible values of α. The attacker can calculate the original coordinate α_{orig} of point P through calculation $\alpha_{orig} = \sqrt[8]{\alpha}$, as discussed in section 5.3.3. Unlike the case of DSMs, altering the inputs does not provide actual protection to the Karatsuba multipliers.

5.5 Conclusions

In the above sections CPA attacks were successfully applied on the insecure steps of the η_T pairing algorithm. Both the operations of addition, $\alpha + v$ in step 4 and multiplication $\gamma \leftarrow \alpha v$ in step 6 of Algorithm 2.2, are shown to be insecure.

When attacking the adder, the attacker targets only one bit of the output register at a time. This results in a small value of ρ. To distinguish this small value from the *noise* level, the attacker has to take 50,000 power traces to perform a successful CPA attack against the adder. From Fig. 5.6 it can be clearly seen that, by taking 15,000 power traces, the correct correlation factor can be recognized. Also by taking 50,000 power traces, the attacker can be sufficiently certain which hypothetical value is correctly generated. In the successful attack against the DSMs, the attacker guesses 8 bits of input v, but generates all 571 bits of the 'z Register' for each clock cycle. This gives the attack a good correlation result, a much higher ρ value than in the attack on the adder. In this attack, 1000 power traces are taken. The correlation results in Fig. 5.9 and Fig. 5.11 show that, by taking only 300 power traces, the correct correlation factor can be recognized. This indicates that the number of bits being targeted in the CPA attack affects the number of power traces required. For the same design, the more bits being targeted, the less power traces are required.

Finally, consider the attacks on DSMs of different field size m. The design over $GF(2^{571})$ requires much more hardware resources and generates more noise in the operations than over $GF(2^{163})$. However, when attacking both designs over $GF(2^{571})$ and $GF(2^{163})$, the number of power traces required is almost the same. This is because the ratio of target registers to total registers, i.e. in equation 5.5 the value of $\sqrt{\frac{l}{L}}$, did not change much. This indicates that when applying CPA attacks on hardware implementations, when the total number of registers in the design increases, the number register targeted also should increase.

6. COUNTERMEASURES AGAINST CPA ATTACKS

6.1 Introduction

Successful CPA attacks have been demonstrated in Chapter 5. The results of several attacks against different components in the Tate pairing system showed that it is necessary to propose useful countermeasures against them. In this chapter, three countermeasures are considered and their implementation results are given. From both mathematical and hardware requirement perspectives, the efficiency of these countermeasures will be analyzsd and the security aspect of these countermeasures will be discussed.

Section 6.2 summarizes the weakness of the Tate pairing algorithm and gives several suggestions to improve the security of its implementation. Section 6.3 introduces several countermeasures proposed by Page and Vercauteren [16], Scott [17], Coron [18] and Kim [145] et al, including the bilinearity method, projective coordinate method and randomizing miller variable method. Section 6.4 discusses the security aspects of the proposed countermeasures. In sections 6.5 and 6.6 these countermeasures are implemented on FPGAs and their costs are listed. Section 6.7 analyzes the performance of these implementation results.

6.2 Weakness in the η_T pairing

As shown in Sections 5.4.1 and 5.4.3, by taking a number of power traces ($N = 1000 - 50000$) and applying a correlation power analysis (CPA) attack against the hardware implementation, the η_T pairing algorithm is shown to be vulnerable to power analysis attacks. The DSM is weak because when $P(\alpha, \beta)$ is public and $Q(x, y)$ is secret, by applying the CPA attack described above, an attacker can always reveal the intermediate value v which is tightly related to coordinate x of point $Q(x, y)$ [144]. The security of a Karatsuba multiplier is reduced if pipelining is applied to improve the multiplier's performance. The adder is even weaker since if any of the two operands is related to the inputs known by the attacker then, by applying a

CPA attack, an attacker can always reveal the other operand input into the adder. Thus, countermeasures must be applied on the η_T pairing to mask the vulnerable multiplications and additions.

6.3 Countermeasures to protect the η_T pairing

Some weakness of the operation blocks can be overcome by carefully implementing the algorithm following some rules. These include the weaknesses in the DSM, the Karatsuba Multiplier and the adder:

1. **RULE1:** A designer must make the input a of the DSM private and not accessible to the attacker.

   If DSMs are used in the structure, the input arrangement of the DSMs must be considered. Recall the DSM structure illustrated in Fig. 3.2, as proved in Chapter 5, the a input is safe while b might leak information. For example, in an IBE system, a designer should carefully choose the input to the multiplier and make sure that the coordinates of the private input point K_{Bob} are always input from the a side of the DSM. This can be achieved by altering the input points of the pairing system or by altering the input from the DSM's end.

2. **RULE2:** In the designs where a Karatsuba multiplier is used, the designer must not improve the critical path by pipelining.

   As discussed in the attack in section 5.4.5, if the designer uses inserted registers to improve the performance of the Karatsuba multiplier, the cryptosystem can be vulnerable. Although increasing the bitwidth n of the sub-multipliers in the Karatsuba multiplier structure or increasing the number of such sub-multipliers can make the CPA attack more difficult, the existence of inserted registers always decreases the security of the Karatsuba multiplier.

By following the rules above, a designer can protect the operations in the multiplication modules from side channel attacks. However, no matter how the designer chooses to implement the Tate pairing, the $GF(2^m)$ Adder structure is not secure. Both in combinational logic architecture designs and bus type architecture designs, secret information can be revealed through the attack presented in section 5.4.1. From the attacker's point of view, for the $GF(2^m)$ addition operation, both inputs α and v of the adder shown in Fig. 3.1 should not be accessible to the attacker. Neither carefully arranging the inputs of the η_T pairing nor by altering the inputs of the $GF(2^m)$ Adder can prevent the attacker, or the eavesdropper in the public key system, from revealing one of the inputs. Thus, $GF(2^m)$ Adder must be protected.

RULE3: Additional operations must be introduced into the original η_T algorithms to protect the two inputs of the $GF(2^m)$ Adder from being discovered by the attacker.

In this chapter, three algorithms that follow $RULE3$ and protect the weak $GF(2^m)$ addition operation in the η_T algorithm are proposed. Note that when implementing these countermeasures, $RULE1$ and $RULE2$ must be considered too, i.e. the multiplications still need to be carefully arranged due to the weaknesses in the DSM and the Karatsuba Multiplier.

6.3.1 Exploiting bilinearity to protect the $GF(2^m)$ Adder

Some of the natural properties of pairings can be used in order to protect the pairing against the attacks in Chapter 5 and the addition attack, in particular. One of the most important properties is the bilinearity property:

$$e(Q; P_1 + P_2) = e(P_1 + P_2; Q) = e(P_1; Q) \times e(P_2; Q), \tag{6.1}$$

where '×' represents a multiplication operation over $GF(2^{4m})$ [77]. Several countermeasures against SCA attacks have been proposed based on this property [16, 17]. Equation 6.1 provides a relationship $e(P; Q) = e(P; Q + R) \div e(P; R)$, where '$\div$' represents a division operation over $GF(2^{4m})$ [154]. This means that an η_T pairing calculation can be derived by introducing a redundant random point R and applying another two η_T pairing calculations.

Taking advantage of bilinearity, Page and Vercauteren [16] and Scott [17] presented a countermeasure which blinds an input point. By using the relationship $e(P; Q) = e(P; Q + R) \div e(P; R)$, an algorithm based on the original pairing algorithm is applied and shown in algorithm 6.1.

This modified algorithm has two inputs: P and $Q \in E(GF(2^m))$. A random point $R \in E(GF(2^m))$ is applied in this algorithm. Step 1 adds the two points Q and R through the point addition operation $S = Q + R$ over $E(GF(2^m))$ and two independent η_T pairing operations are subsequently applied. Moreover, a division '$\div$' over $GF(2^{4m})$ is required in step 4.

This countermeasure creates a new point S over the elliptic curve through the point addition operation over $E(GF(2^m))$. As shown in equation 6.2, a random number $\sigma \in GF(2^m)$ is introduced to this equation to protect the point addition. A

Algorithm 6.1 Countermeasure using bilinearity

```
Define: $e(P;Q)$ is an original $\eta_T$ pairing algorithm in Algorithm 2.2.
Input : $P, Q$.
Random Point : $R$
Output : $c = e(P;Q+R) \div e(P;R)$
1: $S = Q + R$   9M+9A+1S+2I
2: $c_1 = e(P;S)$  (7m+53)M+(24m+65)A+(8m+10)S+1I
3: $c_2 = e(P;R)$  (7m+53)M+(24m+65)A+(4m+10)S+1I
4: $c = c_1 \div c_2$   43M+38A+4S+1I
5: return $c$
        total cost:(14m+158)M+(48m+177)A+(12m+25)S+5I
```

cost of 9 **M**, 9 **A**, 1 **S** and 2 **I** is incurred to perform such a secure point addition.

$$\begin{array}{l} S = Q + R, \textbf{random number } \sigma \in GF(2^m) \\ \lambda = \frac{\sigma * y_Q + \sigma * y_R}{\sigma * x_Q + \sigma * x_R} \\ x'_S = \sigma * \lambda^2 + \sigma * \lambda + \sigma + \sigma * x_Q + \sigma * x_R \\ y'_S = (\sigma * x_Q + x'_S)\lambda + x'_S + \sigma * y_Q \\ y_S = \frac{y'_S}{\sigma}, x_S = \frac{x'_S}{\sigma} \\ Cost = 9\mathbf{M} + 9\mathbf{A} + 1\mathbf{S} + 2\mathbf{I} \end{array} \quad (6.2)$$

Because point R and the random number σ are generated by a random number generator and are not dependent on the inputs P or Q, point S is consequently private. The two independent η_T pairings in step 2 and 3 take public point P and private points S and R as inputs.

In this countermeasure, the two η_T pairings $c_1 = e(P, S)$ and $c_2 = e(P, R)$ can be executed in parallel. The updates of intermediate variables α and β and some of the RAM update operations are the same and can be combined. Thus, in each iteration of the 'for' loop in steps 3-7 of Algorithm 2.2, which is executed m times and dominates the calculation time, there are additional operations of 24**A**, 4**S** and 7**M**. The operation $c = c_1 \div c_2$ in step 4 of Algorithm 6.1 can be calculated through a $GF(2^{4m})$ inversion (as introduced in section 4.2.3, costing 34**M**+16**A**+4**S**+1**I**) followed by a $GF(2^{4m})$ multiplication (as introduced in section 4.2.1, costing 9**M**+22**A**), together costing 43**M**+38**A**+4**S**+1**I**. The total cost of this countermeasure is $(14m + 158)$**M**, $(48m + 177)$**A**, $(12m + 25)$**S** and 5**I**.

6.3.2 Randomized Miller variable to protect the $GF(2^m)$ Adder

The final exponentiation in step 8 of Algorithm 2.2 is calculated by

$$C(t)^{2^{2m}-1} = (C(t)^{2^m-1})^{2^m+1} = (\frac{C(t)^{2^m}}{C(t)})^{2^m+1}. \tag{6.3}$$

In this equation, $C(t) = \{C_0 + C_1t + C_2t^2 + C_3t^3\}$ and $C_i \in GF(2^m)$, $i = 0, 1, 2, 3$. For the Galois Field $GF(2^{4m})$ of $t^4 = t + 1$ in this work, for $m = 163$, 283 and 571, there is: $C(t)^{2^m} = (C_0^{2^m} + C_1^{2^m}) + (C_2^{2^m} + C_3^{2^m})t + C_1^{2^m}t^2 + C_3^{2^m}t^3$. Since there exists the basic Galois Field property:

$$a^{2^m} = a, \tag{6.4}$$

where $a \in GF(2^m)$. For equation 6.3, and with $m \mod 4 = 3$, $C(t)^{2^{2m}-1}$ can be expressed as

$$C(t)^{2^{2m}-1} = (C(t)^{2^m-1})^{2^m+1} = (\tfrac{(C_0+C_1)+(C_2+C_3)t+C_1t^2+C_3t^3}{C_0+C_1t+C_2t^2+C_3t^3})^{2^m+1}. \tag{6.5}$$

If all elements in $C(t)$ are multiplied by some random number $r \in GF(2^m)$, the results of the final exponentiation of $r * C(t) = r * C_0 + r * C_1t, r * C_2t^2, r * C_3t^3$ and of $C(t) = C_0 + C_1t + C_2t^2 + C_3t^3$ are the same because:

$$\begin{aligned} &(r * C(t))^{2^{2m}-1} \\ &= (\tfrac{(r*C_0+r*C_1)+(r*C_2+r*C_3)t+r*C_1t^2+r*C_3t^3}{r*C_0+r*C_1t+r*C_2t^2+r*C_3t^3})^{2^m+1} \\ &= (\tfrac{r*\{(C_0+C_1)+(C_2+C_3)t+C_1t^2+C_3t^3\}}{r*\{C_0+C_1t+C_2t^2+C_3t^3\}})^{2^m+1} \\ &= C(t)^{2^{2m}-1} \end{aligned} \tag{6.6}$$

For $m = 233$, the Frobenius map is different (see section 4.2.4), however, the same result can be derived. This means that in step 4 of Algorithm 2.2, if the intermediate Miller variable $A(t)$ is simply multiplied by a random constant $r \in GF(2^m)$, $r * A(t)$, the resulting value of $C(t)$ in step 9 of the η_T pairing will not change. This is a useful property in the η_T pairing.

Utilizing this property, Scott [17] introduced another countermeasure which randomizes the intermediate variable in the 'for' loop in steps 3-7 of Algorithm 2.2 by introducing a random number $r \in GF(2^m)$. To resist side channel analysis (SCA) attacks, all intermediate variables in steps 2 and 4 of Algorithm 2.2 are multiplied by this random number r. The pairing algorithm is modified accordingly and shown in Algorithm 6.2.

As can be seen in step 3, all the intermediate variables are multiplied by the

Algorithm 6.2 Randomizing Miller variable

```
Input : $P(\alpha, \beta), Q(x, y)$
Random Number : $r$
Output : $c = e(P; Q)$
C(t) ← 1
$u \leftarrow x^2 + y^2 + g + \frac{m-1}{2}, v \leftarrow x^2, \alpha \leftarrow \alpha^4, \beta \leftarrow \beta^4$
$u \leftarrow ru, \delta \leftarrow rv, \alpha_1 \leftarrow r\alpha, \beta_1 \leftarrow r\beta, \gamma \leftarrow \delta\alpha$
for $i = 0 : m - 1$
  $A(t) \leftarrow \gamma + u + \beta_1 + (\alpha_1 + \delta)t + (\alpha_1 + \delta + r)t^2$
  $C(t) \leftarrow C^2(t) * A(t)$
  $u \leftarrow u + \delta + r, \delta \leftarrow \delta + r, \alpha \leftarrow \alpha^4, \beta \leftarrow \beta^4, \gamma \leftarrow \delta\alpha$
  $\alpha_1 \leftarrow r\alpha, \beta_1 \leftarrow r\beta,$
end for
return $C(t) \leftarrow C(t)^{2^{2m}-1}$
    total cost:(11m+58)M+(26m+65)A+(8m+10)S+1I
```

random number $r \in GF(2^m)$. This masks all the input coordinates and protects the inputs of multiplications and additions from being controlled by the attacker. i.e. the Miller variable $A(t)$ can be expressed in equation 6.7.

$$\begin{aligned} A(t) &= \gamma + u + \beta_1 + (\alpha_1 + \delta)t + (\alpha_1 + \delta + r)t^2 \\ &= r * [\gamma + u + \beta + (\alpha + v)t + (\alpha + v + 1)t^2] \end{aligned} \tag{6.7}$$

This multiplies intermediate variable $C(t)$ by random number r in each iteration. According to equation 6.6, the affect of introducing r can be eliminated in the final exponentiation in step 10 of Algorithm 6.2. The calculation result is the same as the original pairing calculation.

This countermeasure takes 5 more multiplications in the pre-loop operation, 4 multiplications and 2 additions more in each iteration of the 'for' loop, $(4m + 5)$**M** and $(2m)$**A** more in total as the extra cost to protect the η_T pairing.

6.3.3 Using projective coordinates to protect the $GF(2^m)$ Adder

A countermeasure using projective coordinates proposed by Coron [18] was expanded on by Kim in [145]. This countermeasure takes a random value $\lambda \neq 0 \in GF(2^m)$ and projects an affine point $Q(x, y)$ to standard or homogeneous projective [31] coordinates $(X, Y, Z) \leftarrow (\lambda x, \lambda y, \lambda)$, where $x = \frac{X}{Z} = \frac{X}{\lambda}$, $y = \frac{Y}{Z} = \frac{Y}{\lambda}$. Algorithm 6.3 presents how this method is applied to the original η_T algorithm.

The coordinate projection operation happens in step 3 of Algorithm 6.3, following the first squaring operations of the input coordinates. The initiation of the

Algorithm 6.3 Using homogeneous projective coordinates

```
Input : $P(\alpha,\beta), Q(x,y)$
Random Number : $\lambda \neq 0 \in GF(2^m)$
Output : $c = e(P;Q)$
 1: $C(t) \leftarrow 1$
 2: $x \leftarrow x^2, y \leftarrow y^2, \alpha \leftarrow \alpha^4, \beta \leftarrow \beta^4$
 3: $(X,Y,Z) \leftarrow (\lambda x, \lambda y, \lambda)$
 4: $V \leftarrow X, U \leftarrow X + Y + \frac{m-1}{2}Z$
 5: $J \leftarrow \alpha(V+Z), K \leftarrow \beta Z, L \leftarrow \alpha Z$
 6: for $i = 0 : m-1$
 7:   $A(t) \leftarrow J + K + U + (L+V)t + (L+V+Z)t^2$
 8:   $C(t) \leftarrow C(t)^2 * A(t)$
 9:   $U \leftarrow U + V + Z, V \leftarrow V + Z, \alpha \leftarrow \alpha^4, \beta \leftarrow \beta^4$
10:   $J \leftarrow \alpha V, K \leftarrow \beta Z, L \leftarrow \alpha Z$
11: end for
12: return $C(t) \leftarrow C(t)^{2^{2m}-1}$
        total cost:(11m+58)M+(26m+65)A+(8m+10)S+1I
```

intermediate variables in step 2 of Algorithm 2.2 is changed into equation 6.8:

$$\begin{aligned} &v \leftarrow \tfrac{X}{Z}, u \leftarrow \tfrac{X}{Z} + \tfrac{Y}{Z} + \tfrac{m-1}{2} \\ &\gamma \leftarrow \alpha(\tfrac{X}{Z}+1), \beta \leftarrow \beta, \alpha \leftarrow \alpha \end{aligned} \tag{6.8}$$

However, performing $GF(2^m)$ division is time consuming. Consider the property of the η_T pairing algorithm, as presented in equation 6.6, where multiplying an element over $GF(2^m)$ with every element of the Miller variable does not affect the result of the η_T pairing's calculation. Here every element in equation 6.8 is multiplied by Z, where Z is initiated to the random number λ in step 3 and, thus, forms steps 4 and 5 in Algorithm 6.3. Note that the names of the intermediate variables are changed accordingly.

Step 4 of Algorithm 2.2 is changed into step 7 of Algorithm 6.3 by multiplying every element of $A(t)$ by Z:

$$\begin{aligned} A(t) &\leftarrow \alpha(\tfrac{X}{Z}+1) + \tfrac{X}{Z} + \tfrac{Y}{Z} + \tfrac{m-1}{2} + \beta + (\alpha + \tfrac{X}{Z})t + (\alpha + \tfrac{X}{Z} + 1)t^2 \\ &\leftarrow \tfrac{1}{Z}[\underbrace{\alpha(X+Z)}_{J} + \underbrace{X + Y + \frac{m-1}{2}Z}_{U} + \underbrace{\beta Z}_{K} + (\underbrace{\alpha Z}_{L} + \underbrace{X}_{V})t + (\underbrace{\alpha Z}_{L} + \underbrace{X}_{V} + Z)t^2 \\ &\leftarrow J + K + U + (L+V)t + (L+V+Z)t^2 \end{aligned} \tag{6.9}$$

Algorithm	**M**	**A**	**S**	**I**
Original η_T	7m+53	24m+65	8m+10	1
Bilinearity	14m+140	48m+123	12m+25	5
Random Miller	11m+58	26m+65	8m+10	1
Projective	11m+58	26m+65	8m+10	1

Tab. 6.1: Cost of η_T pairing and countermeasures

Removing $\frac{1}{Z}$ can be seen as multiplying by Z and the effect can be eliminated at the end of the algorithm by the final exponentiation, as introduced in equation 6.6.

Similar to the randomized Miller variable method, this countermeasure modifies the intermediate Miller variable, by introducing a random number $\lambda \in GF(2^m)$. Every intermediate variable that relates to the point $Q(x, y)$ is masked by this random number in step 3 of Algorithm 6.3. In step 5, both input coordinates α and β of input point $P(\alpha, \beta)$ are multiplied by Z, which is equal to λ. Therefore, every intermediate variable in Algorithm 6.3 is protected by the introduced random number λ.

This countermeasure incurs the same cost, $(4m+5)$**M** and $(2m)$**A**, as Algorithm 6.2 for protecting the η_T pairing.

Note that other than homogeneous projective coordinates [31], Jacobian projective coordinates recommended in [32] uses map $(X, Y, Z) \leftrightarrow (Z^2x, Z^3y, Z)$ and Lopez-Dahab projective coordinates proposed in [33] uses map $(X, Y, Z) \leftrightarrow (Zx, Z^2y, Z)$. These two types of projective coordinates also can be applied in this countermeasure. The countermeasure algorithms and the costs of using such projective coordinates are listed in Algorithms D.1 and D.2. Except for some differences in operations and some additional variables that must be stored, the countermeasures using these three projective coordinates are achieved at almost the same cost. Thus, in the rest of this work, the algorithm using homogeneous projective coordinate will be picked as a representative of all projective coordinate countermeasures.

6.3.4 Cost comparison between countermeasures

The costs of the original η_T pairing and the countermeasures are shown in Table 6.1. As can be seen from the table, the countermeasure using the bilinearity property almost doubles the calculation cost by performing two original η_T pairings. The countermeasures randomizing the Miller variables and using projective coordinates cost the same number of operations. The 11 multiplications in the 'for' loop dominate the calculation time.

In all of the proposed countermeasures, random variables are introduced . In the bilinearity countermeasure, there is a random number $\sigma \in GF(2^m)$ and a random point $R(x_R, y_R)$, where x_R and $y_R \in GF(2^m)$. Countermeasures randomizing Miller variables and using projective coordinates introduced random numbers r and $\lambda \in GF(2^m)$, respectively. In this work, these random numbers are pre-stored in the block RAM. However, it is suggested that designers of cryptosystems use hardware Random Number Generators (RNG) with strong seeds to generate cryptographically secure random numbers to protect the implementations [155]. Different types of embedded RNGs are designed for this purpose [156, 157]. Using dedicated hardware for random number generation ensures that the random number used in the same design changes for every calculation. In choosing RNGs, the designer must make sure that the RNG is secure against existing the attacks [158].

6.4 Security discussion

As discussed in section 6.3, to securely implement the η_T pairing algorithm the three $RULE$s must be obeyed. The following sections discuss the security aspects of the proposed countermeasures, with the successful attacks in mind.

6.4.1 Security of exploiting bilinearity method

The countermeasure using the bilinearity property introduced in Algorithm 6.1 does not change the pairing operation itself, but masks the initial point and performs an additional pairing calculation.

In an IBE (or ABE) system, decryptor Bob recieves point K_{Bob} (or D_i^{Bob}) from the PKG through a secure channel. By carefully arranging the inputs, the designer can make sure that only the public point P_r (or E_i) can be tampered with by the attacker. To apply an SCA attack, the attacker can only manipulate point P_r (or E_i). Consider Algorithm 6.1 where the multiplications in step 1 mask the input point $Q(x, y)$ by multiplying the coordinates x and y by the coordinates of the random point $R(x_R, y_R)$. These coordinates of point R are not known by the attacker and their effect cannot be removed. Even if the attacker knows and can tamper with the information in point Q, the masked point $S = (Q + R)$ is secure because of the use of the random number $\sigma \in GF(2^m)$.

As in previous discussions, firstly consider the case in which point P is secret, i.e. K_{Bob} or D_i^{Bob} and point Q is public, i.e. P_r or E_i. In this case, by introducing the random number σ and carefully arranging the inputs of the multipliers, the attacker cannot know any of the information contained in R and S in step 1 of Algorithm 6.1. Since the inputs of calculations $e(P;S)$ and $e(P;R)$ are secure, this countermeasure protects the η_T algorithm from CPA attacks. Consider the other case in which point P, i.e. P_r or E_i, is public and point Q, i.e. K_{Bob} or D_i^{Bob}, is secret. In this case, the attacker still controls the knowledge, in point P, of the calculations $e(P;S)$ and $e(P;R)$. The CPA attacks proposed in chapter 5 still work. By applying a CPA attack, the attacker can reveal the information in points S and R. A point substraction operation $Q = S - R$ can help the attacker reveal the secret information in point Q.

This means that the designer must ensure that point P in Algorithm 6.1 is secure against the attacker. For security reasons, the designer must apply K_{Bob} or D_i^{Bob} to input P of the pairing calculation $e(P;Q)$ and let public point P_r or E_i enter from input Q.

6.4.2 Security of randomized Miller variable method

Randomizing the Miller variable and using projective coordinates results in a similar increase in point operations for both countermeasures. In both instances, the Miller variable is affected by randomisation and the increase in the algorithm complexity is the same. Consider the security of randomizing the Miller variable. Firstly, assume the attacker knows the public point $P(\alpha, \beta)$. By inputting different values for P and running the algorithm several times, the attacker controls the values of α and β of Algorithm 6.2. In step 3 of Algorithm 6.2, the random value r is introduced into every intermediate variable which will be used in the 'for' loop. To ensure that u, δ, α_1, β_1 and γ are securely masked, random value r must be applied to the a input of the DSM at this step. This will prevent the CPA attacks on the DSMs and, therefore, prevent an attacker from determining the value of any of the variables in step 5.

Now assume that the point $Q(x, y)$ is the public point that can be manipulated by the attacker and, therefore, the attacker can manipulate u and v in step 2. The random value r masks the values u and δ in step 3. In DSM designs, as long as r is applied to the secure input a of the multiplier, the attacker will never reveal u and v. As a result, no matter which input is set to be public, this countermeasure is secure against CPA attacks. The coordinates of the public input will always be masked by the random value r, which is beyond the control of the attacker.

6.4.3 Security of using projective coordinates method

The countermeasure described in Algorithm 6.3 uses a similar approach to protect against CPA attacks. In this countermeasure the pre-stored random value λ is introduced. Assume the attacker knows the public point $P(\alpha, \beta)$ and tries to reveal information in point $Q(x, y)$. Intermediate values J, K and L are masked in step 5, while value V and Z are unknown to the attacker. By carefully arranging the inputs to the DSMs, an attacker will not be able to reveal any of the intermediate values J, K and L. Consider K as an example. In the multiplication $K \leftarrow \beta Z$ in step 5 of Algorithm 6.3, Z is applied to the a input of the DSM while β is applied to the b input of the DSM. In this way, the countermeasure protects information relating to the secret point $Q(x, y)$ from the attackers. Alternatively, assume the attacker knows the public point $Q(x, y)$ and, thus, $P(\alpha, \beta)$ is private. Values x and y are masked in step 3 of Algorithm 6.3 by the pre-stored random value λ. The attacker cannot get variables X, Y and Z. Thus, J, K and L are secured in step 5. The attacker has no chance of revealing information related to the secret point $P(\alpha, \beta)$ in this case. In this way, the countermeasure using projective coordinates protects the η_T algorithm against CPA attacks.

6.4.4 Security operations in common

In both countermeasures using the randomized Miller variable and projective coordinates, no matter how the inputs are assigned, $RULE1$ is satisfied. However, in the bilinearity countermeasure in Algorithm 6.1, the designer must be careful. Only when the public input (K_{Bob} in IBE or D_i^{Bob} in ABE) is set to Q and the private information (P_r in IBE or E_i in ABE) is set to P can this countermeasure follow $RULE1$ and thus, protect the η_T algorithm against CPA attacks.

$RULE2$ can be satisfied by not inserting registers into the Karatsuba multipliers.

All the three countermeasures proposed above provide different approaches to mask the weak addition operation in Algorithm 2.2, i.e. $RULE3$. The bilinearity method in Algorithm 6.1 adds a point addition operation before the pairing calculations and masks the public input point Q to an unknown point S and, thus, masks the input of the weak addition operation in the original η_T algorithm. Both countermeasures using randomized Miller variable and projective coordinates in Algorithms 6.2 and 6.3 introduced a random variable (r and λ) into the original η_T algorithm. By multiplying both inputs of the weak addition operation with the random variable, these two countermeasures ensure that the addition operation no longer leaks information.

6.5 Bus type top-level architecture implementation result of the countermeasures

Since no more operations other than addition, squaring, multiplication and division are needed in the proposed countermeasures, the bus type top-level architecture, as introduced in chapter 4, can be used to implement the countermeasures. In a bus type top-level architecture, as shown in Fig. 4.6, for the same field size m, the same number of multipliers $\#Mult$ and the same digit size d, the same design structure can be reused for the new algorithms. Thus, the area required for the countermeasures is the same as those of the naively implemented Tate pairing, the area results of the designs is not listed in this chapter. However, the newly added operations in the countermeasures require additional calculation time. The time increment of the countermeasure in different schedules of the Tate pairing designs is shown in Table E.1, E.2, E.3 and 6.2. For better comparison, the calculation time of the original design is also given as Algorithm 2.2. Note that, because the method of inserting registers to improve the Karatsuba multiplier is proven to be insecure, such method is not applied in the implementations of this chapter.

In Tables E.1, E.2, E.3 and 6.2, Algorithm 6.1 represents the operation time of the Tate pairing protected by the bilinearity countermeasure. The operation times for the randomised Miller variable (section 6.3.2) and projective coordinate (section 6.3.3) countermeasures are represented by Algorithm 6.2 & 6.3 respectively. Countermeasures in Algorithm 6.2 and Algorithm 6.3 have different motivations and different names for their intermediate variables, but result in the same cost and calculation flow. The schedules of the operations are the same and, thus, incur the same time penalty for each configuration. The column labelled 'Ratio' in Tables E.1, E.2, E.3 and 6.2 shows the ratio of calculation time for the countermeasures to the original algorithm under the same design parameters.

Since there are 14 multiplications in the 'for' loop of Algorithm 6.1 and 11 multiplications in the 'for' loop of Algorithms 6.2 and 6.3, for fastest designs, 14 and 11 multipliers are used to implement the countermeasures, respectively. As in previous chapters, area, time and A*T product are taken as the parameters to show the performance of the designs. The implementation results are shown in Tables 6.3 and 6.4.

It should be noted, the bus type implementations of Algorithm 6.1 using 11

Parameters			Alg.2.2	Alg. 6.1		Alg. 6.2 & 6.3	
#Mult	d	n	Time (μs)	Time (μs)	Ratio	Time (μs)	Ratio
1× DSM	1	571	10182.6	20451.0	200.8 %	15831.2	155.5 %
	2	286	5215.9	10476.0	200.8 %	8057.4	154.5 %
	4	143	2723.9	5471.0	200.9 %	4156.9	152.6 %
	8	72	1486.6	2986.0	200.9 %	2220.3	149.4 %
	16	36	859.2	1726.0	200.9 %	1238.4	144.1 %
	32	18	545.5	1096.0	200.9 %	747.4	137.0 %
2× DSM	1	571	6063.2	10832.8	178.7 %	8974.9	148.0 %
	2	286	3155.4	5664.2	179.5 %	4626.4	146.6 %
	4	143	1696.4	3070.7	181.0 %	2444.5	144.1 %
	8	72	972.0	1783.1	183.4 %	1361.2	140.0 %
	16	36	604.7	1130.2	186.9 %	811.9	134.3 %
	32	18	421.1	803.8	190.9 %	537.3	127.6 %
3× DSM	1	571	4609.2	7947.8	172.4 %	6086.5	132.1 %
	2	286	2427.4	4219.5	173.8 %	3179.9	131.0 %
	4	143	1332.6	2348.9	176.3 %	1721.5	129.2 %
	8	72	789.1	1420.1	180.0 %	997.4	126.4 %
	16	36	513.5	949.1	184.8 %	630.3	122.7 %
	32	18	375.7	713.6	190.0 %	446.7	118.9 %
4× DSM	1	571	3348.0	6879.9	205.5 %	4910.9	146.7 %
	2	286	1817.0	3728.1	205.2 %	2619.5	144.2 %
	4	143	1027.1	2102.2	204.7 %	1438.5	140.1 %
	8	72	642.1	1309.6	204.0 %	862.5	134.3 %
	16	36	446.9	907.7	203.1 %	570.4	127.6 %
	32	18	349.3	706.8	202.4 %	424.4	121.5 %
7× DSM	1	571	1829.0	3854.2	210.7 %	3404.4	186.1 %
	2	286	1037.8	2174.7	209.5 %	1836.6	177.0 %
	4	143	648.2	1347.4	207.9 %	1062.9	164.0 %
	8	72	452.3	931.6	206.0 %	674.5	149.1 %
	16	36	351.8	718.4	204.2 %	476.0	135.3 %
	32	18	302.3	613.3	202.9 %	377.8	125.0 %
Kara.	no Reg	5	419.7	843.6	201.0 %	516.9	123.2 %

Tab. 6.2: Calculation time (μs) of proposed countermeasures, $m = 571$

multipliers of digit size d=32 and Algorithm 6.2 and 6.3 using 14 multipliers of digit size d=16 and 32, are too complicated and cannot be implemented with the software tools. Several implementations in the table above exceed the area constraint of the Virtex-V xc5vlx50 technology [84]. The hardware resources required (in the form of registers and LUTs) can be calculated, but to implement such designs, a larger FPGA device must be used. In addition, the operation times of these implementations are

Parameters			Results				
m	d	n	Regs	LUTs	Freq. (MHz)	Time (μs)	Area*Time (LUT*s)
163	1	163	8721	10445	190.2	389.4	4.07
	2	82	8720	13053	180.6	184.8	2.41
	4	41	8705	13388	195.5	164.9	2.21
	8	21	8746	18103	195.5	140.9	2.55
	16	14	8843	29681	195.5	132.5	3.93
	32	14	9039	44746	185.3	139.8	6.25
233	1	233	12359	15330	183.2	557.5	8.55
	2	117	12360	18813	183.2	364.4	6.86
	4	59	12373	21378	185.4	264.7	5.66
	8	30	12414	28053	185.4	217.0	6.09
	16	15	12399	40848	184.5	193.2	7.89
	32	14	12595	61360	187.8	188.2	11.55
283	1	283	14977	14121	179.2	767.8	10.84
	2	142	14978	18334	179.9	486.6	8.92
	4	71	14962	22879	190.1	327.9	7.50
	8	36	15002	30969	188.0	265.5	8.22
	16	18	14990	50114	181.4	239.9	12.02
	32	14	14962	69814	183.9	228.9	15.98
571	1	571	29970	27978	178.1	2465.6	68.98
	2	286	29969	35943	178.1	1436.7	51.64
	4	143	29940	43420	193.4	847.6	36.80
	8	72	29983	51686	193.4	611.5	31.61
	16	36	Out of memory				
	32	18	Out of memory				

Tab. 6.3: Performance of the countermeasures applied on η_T algorithm, Algorithm 6.1, $\#Mult$=14, bus type top-level architecture

estimated values.

Having all multiplications in the 'for' loop of the Tate pairing operated in parallel speeds up the calculation. By using more multipliers, the calculation time of the 'for' loop can be reduced even further. However, considering the pipeline schedule of the DSMs, as illustrated in Fig. 2.11, the bus type architecture can arrange the input pair of only one DSM in one clock cycle. Thus, to pipeline all 14 (or 11) DSMs in the 'for' loop, at least 14 (or 11) clock cycles are required. In this case, even though it only takes 6 clock cycles to calculate a $GF(2^{163})$ digit-serial multiplication, or 8 and 9 clock cycles for fields $GF(2^{233})$ and $GF(2^{283})$, respectively, the DSM will still have to wait for 14 clock cycles until the data bus is free from arranging the inputs of the 13 other multiplications. This makes using digit size d=32 on these comparatively

Parameters			Results				
m	d	n	Regs	LUTs	Freq. (MHz)	Time (μs)	Area*Time (LUT*s)
163	1	163	7197	6719	211.1	221.1	1.49
	2	82	7196	8490	211.1	151.6	1.29
	4	41	7184	10368	211.1	116.4	1.21
	8	21	7217	12033	209.7	100.0	1.20
	16	11	7293	19800	211.1	90.7	1.80
	32	11	7458	33656	211.1	90.7	3.05
233	1	233	10206	9145	210.0	394.9	3.61
	2	117	10207	11834	216.4	248.7	2.94
	4	59	10217	14295	210.0	186.9	2.67
	8	30	10249	17065	210.0	152.2	2.60
	16	15	10237	27547	210.8	133.8	3.69
	32	11	10402	46281	210.8	129.0	5.97
283	1	283	12371	11868	217.8	527.2	6.26
	2	142	12369	15109	217.8	332.3	5.02
	4	71	12359	21249	210.4	242.5	5.15
	8	36	12392	24642	210.4	192.4	4.74
	16	18	12380	37236	210.4	166.6	6.20
	32	11	12369	55866	212.8	154.8	8.65
571	1	571	24769	23516	222.9	1776.2	41.77
	2	286	24767	29785	222.9	1023.1	30.47
	4	143	24756	25071	219.8	654.3	16.40
	8	72	24790	41569	219.8	464.1	19.29
	16	36	24779	66833	219.8	367.6	24.57
	32	18	Out of memory				

Tab. 6.4: Performance of the countermeasures applied on η_T algorithm, Algorithm 6.2 & 6.3, $\#Mult$=11, bus type top-level architecture

small fields inefficient.

6.6 Mixed Type Implementation results of the countermeasures

As illustrated in Fig. 4.8 of section 4.3.4, the mixed type top-level architecture can be applied to speed up the hardware accelerator of the Tate pairing algorithm. In this section the implementation results of the countermeasures proposed using the mixed type top-level architecture are presented. As in designing the original η_T algorithm, the mixed type design of Algorithms 6.2 and 6.3 expands all the addition and squaring operations in the '**for loop logic**' block. All these operations require only one clock cycle. Together with the 7 multiplications which require 8 clock cycles,

Parameters			Results					
m	Algorithm	n	Regs	LUTs	Freq. (MHz)	Time (us)	Ratio	Area*Time (LUT*s)
163	Alg. 6.1	1	14043	29257	149.1	34.1	254.8 %	1.00
	Alg. 6.2 & 6.3	1	11604	20735	148.9	18.2	135.9 %	0.38
233	Alg. 6.1	1	30747	42094	149.6	47.1	256.2 %	1.98
	Alg. 6.2 & 6.3	1	16424	30831	148.6	25.3	137.6 %	0.78
283	Alg. 6.1	1	37592	53758	145.5	58.1	251.4 %	3.12
	Alg. 6.2 & 6.3	1	20009	39727	140.6	32.1	138.9 %	1.27
571	Alg. 6.1	5	79648	93390	171.9	285.3	221.7 %	26.65
	Alg. 6.2 & 6.3	5	44203	64867	171.9	198.9	154.5 %	12.90

Tab. 6.5: Performance of the countermeasures applied on η_T algorithm, mixed type top-level architecture, no registers inserted

each iteration of the ‘for’ loop in Algorithms 6.2 and 6.3 requires 9 clock cycles. In the Karatsuba design of Algorithm 6.1, there is only one original ‘**for loop logic**’ block. The two Tate pairing operations share the same block. A 2:1 multiplexor array selects the input to the ‘**for loop logic**’ block. The multiplixor array requires additional hardware resources, but less than a ‘**for loop logic**’ block.

Area, time and A*T product are taken as parameters for analysis of the performance of the implementations. The experimental results over Xilinx Virtex-V xc5vlx50 technology [84] are shown in Table 6.5. Note that, except for the design of Algorithms 6.2 and 6.3 over $GF(2^{163})$, all the designs, as listed in Table 6.5, exceed the area constraint of Xilinx Virtex-V xc5vlx50 technology [84]. Larger FPGA technology must be chosen to implement these larger designs.

6.7 Analysis of implementation result of the proposed countermeasures

This section compares the performance of the designs according to the implementation results in the tables in the previous sections. As in the analysis in section 4.4, because for each m, when the parameters ($\#Mult$ and d) change, the trends of how the implementation results change are similar. Here, implementation results for proposed countermeasures over $GF(2^{571})$ are taken as an example.

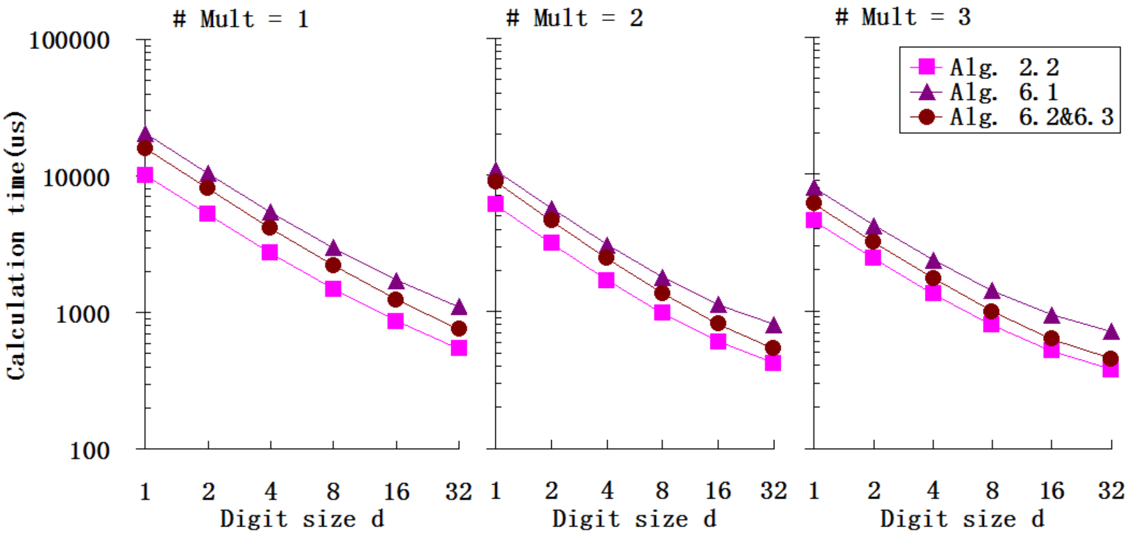


Fig. 6.1: Time requirement of the countermeasures, bus type, DSM designs, d=1, 2 and 3, m=571

6.7.1 Time analysis

Figs. 6.1 and 6.2 show the calculation time of the original η_T algorithm (Algorithm 2.2), the countermeasure in Algorithm 6.1 and the countermeasures in Algorithms 6.2 and 6.3, over $GF(2^{571})$. Each sub-figure shows the case for different $\#Mult$. Fig. 6.3 shows the calculation time of the maximum DSM designs of the countermeasures, i.e. 14×DSM for Algorithm 6.1 and 11×DSM for Algorithms 6.2 and 6.3. The Karatsuba design timing results are also illustrated in Fig. 6.3. As can be seen from these figures, when $\#Mult$ increases or when digit size d increases, the calculation time of the designs decreases. Optimizing the multiplications speeds up the countermeasures in the same way as it speeds up the original η_T algorithm. However, when using the same multiplier schedule, Algorithm 6.1 always requires more time for the calculation than Algorithms 6.2 and 6.3 due to the time penalty in the additional operations. In Figs. 6.1 and 6.2, when using the same number of multipliers, the calculation time required for Algorithms 6.2 and 6.3 is always more than that for the original η_T algorithm. However, the 'Max DSM Designs' sub-figure of Fig. 6.3 shows that by adding more DSMs into the design and having all the additional 4 multiplications pipelined with the original 7 multiplications, the calculation of Algorithms 6.2 and 6.3 can be speeded up to same level as that of the original η_T algorithm. Algorithm 6.1 doesn't show this feature because apart from additional multiplications, when compared to Algorithm 2.2, the additions and squarings in Algorithm 6.1 are doubled. Optimizing the multiplications in Algorithm 6.1 does not

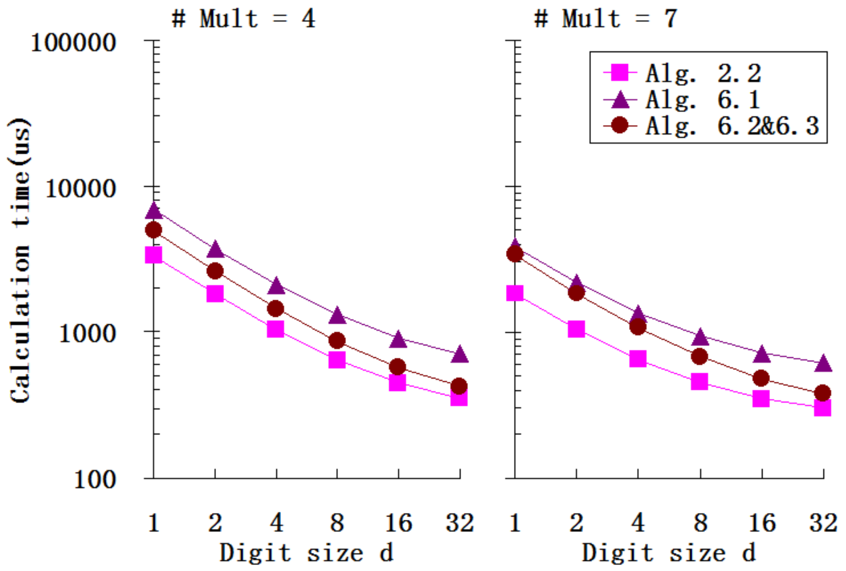


Fig. 6.2: Time requirement of the countermeasures, bus type, DSM designs, d=4 and 7, m=571

help reduce the calculation time of the other operations. The 'Karatsuba Designs' sub-figure shows the calculation time of designs using Karatsuba multipliers over field $GF(2^{571})$. 'BN' and 'MN' represent bus type design without registers inserted and mixed type design with registers inserted, respectively. Similar to the maximum DSM designs, the bus type Karatsuba design results of Algorithms 6.2 and 6.3 and Algorithm 2.2 are almost the same. However, in mixed type designs over $GF(2^{571})$, the 'for' loop takes 36 clock cycles for Algorithm 2.2, 72 clock cycles for Algorithm 6.1 and 56 clock cycles for Algorithms 6.2 and 6.3, respectively. This enlarges the difference between the calculation time for each algorithm.

6.7.2 Time ratio analysis

Fig. 6.4 and 6.5 show the time increment ratio of the proposed countermeasures compared to the original η_T of Algorithm 2.2. In Fig. 6.4 it can be clearly seen that the calculation time ratio for designs implementing Algorithm 6.1 is over 200% when using 1, 4 and 7 DSMs and Karatsuba multiplier of both types of top-level architecture. However, when using 2 and 3 DSMs, the designs of Algorithm 6.1 exhibit a time ratio of less time than 200%. This is because the unused DSMs in these two cases of the original η_T algorithm, as shown in the schedule in Fig. 4.7, are utilized to perform the additional operations in Algorithm 6.1. When digit size d increases, the proportion of calculation time consumed by the multiplications

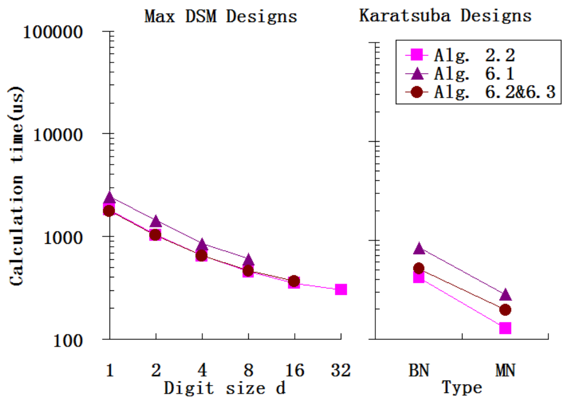


Fig. 6.3: Time requirement of the countermeasures, bus type, maximum DSM designs and Karatsuba designs, m=571

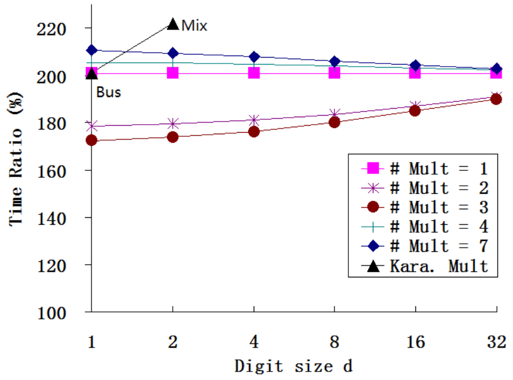


Fig. 6.4: Time Ratio of the countermeasure in Algorithm 6.1, m=571

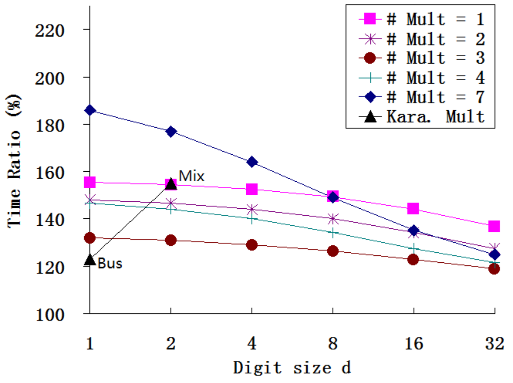


Fig. 6.5: Time Ratio of the countermeasures in Algorithms 6.2 & 6.3, m=571

decreases. The time ratio does not vary significantly, but shows a trend towards 200%. When the calculation time of the multiplications is reduced to the extreme case, i.e. in only 1 clock cycle, the ratio should be 200%, as in the case of bus type Karatsuba designs. As in Fig. 6.4, in the calculation time increment ratio shown in Fig. 6.5, different numbers of DSMs provide different utilizations of the unused DSMs in the original η_T algorithm. In bus type designs, increasing the digit size d of the DSMs makes the time ratio tend to the same level as the bus type Karatsuba design. In both Figs. 6.4 and 6.5, mixed type Karatsuba designs exhibit higher time ratios than bus type Karatsuba designs. This is because the '**for loop logic**' block optimized the calculation time of the additions and squarings to a very short time and consequently multiplications once again dominate the calculation time and, thus, dominate the time increment ratio: about $\frac{14}{7}$ = 200% for Algorithm 6.1 and

$\frac{11}{7} = 157\%$ for Algorithms 6.2 and 6.3.

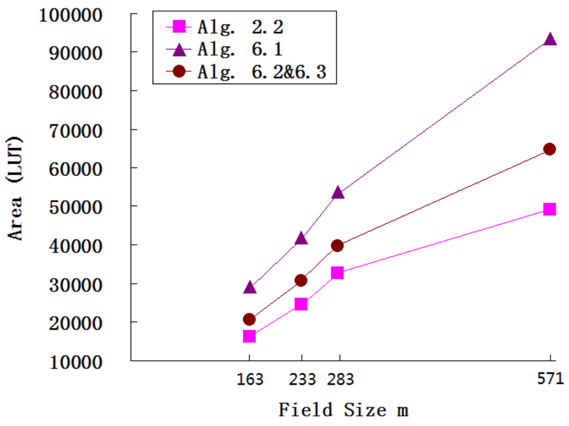


Fig. 6.6: Area requirement of countermeasure implementations across different field size m, mixed type top-level architecture

As the countermeasure designs using DSMs has the same structure as the original η_T algorithm, the bus type designs' area performances are not discussed further here. Fig. 6.6 shows the area performance of the mixed type designs of the countermeasures. As can be seen from the algorithms, the countermeasures require additional operations to protect the pairing algorithm. To implement the additional multiplications in the 'for' loop, more multiplexors are required. Also, to implement additional additions and squarings in the 'for' loop, more XOR chains are required. These all make the hardware resources requirement of the countermeasures larger than that of the original η_T algorithm. When field size m increases from 163 to 283, the Karatsuba multiplier in the mixed type designs increases proportional to $n^2 \times 3^{\log_2 \frac{m}{n}}$. When m increases from 283 to 571, although the Karatsuba multiplier does not change significantly, all the addition and squaring components in the designs grow linearly with m and, thus, the area of the designs still increases.

*6.7.3 A*T product analysis*

Fig. 6.7 shows the A*T product comparison of bus type and mixed type Karatsuba designs. Since the bus type designs of the same parameters are of the same area as the original η_T algorithm, the A*T product of such designs are of the same ratio as their timing performance as seen in Fig. 6.7. However, the A*T product performances of the mixed type Karatsuba designs shows a difference. The A*T product of Algorithm 2.2 is smaller than its bus type design. In the countermeasure designs, because of the additional operations, both the calculation time and the hardware resources

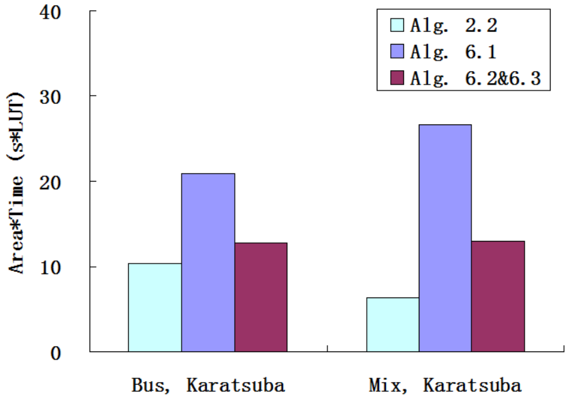


Fig. 6.7: Area*Time product of the countermeasures, Karatsuba designs, m=571

requirements increased. This makes the A*T product of mixed type Karatsuba designs larger than that of bus type Karatsuba designs.

6.8 Conclusions

This chapter investigated the FPGA implementation of three countermeasures applied to the η_T algorithm, to protect it against CPA attacks. All the three countermeasures proposed in this chapter ensure the security of η_T pairing in IBE and ABE systems. As discussed in section 6.4, the countermeasures Randomizing Miller variable in Algorithm 6.2 and using projective coordinates in Algorithm 6.3 provide the security of the η_T pairing algorithm, irrespective of which input is exposed to the insecure channel. However, the designers using the bilinearity method shown in Algorithm 6.1 still have to carefully arrange the inputs by connecting the information from an insecure channel to input Q of the pairing based cryptosystem.

The operations required for the countermeasures, when compared to original η_T algorithm, were analyzed. The implementation results of these three countermeasures for bus type and mixed type architectures were shown. Similar to the implementation results of the original η_T algorithm, the mixed type architecture shows the best performance in calculation time. However, the cost of fast calculation speed is a large hardware resources requirement. Most of the designs of the countermeasures using the mixed type architecture exceed the area constraint of the proposed FPGA device in this work (Xilinx Virtex-V xc5vlx50 technology [84]).

The time increments of the countermeasures under different multiplier schedules were listed. The results show that the bilinearity countermeasure takes 200% of the calculation time of the original η_T algorithm when using the same amount of hardware resources. The time ratio of countermeasures Randomizing Miller variable and using projective coordinates is about 157%, which is smaller.

This chapter showed the implementation results of the proposed three countermeasures. Unlike previous works [16, 17, 18, 145], this work presented the cost overheads of the countermeasures from the hardware designer's point of view.

7. CONCLUSIONS

7.1 Contributions to the Field

The main contributions of this thesis are summarized in this Chapter. In Chapter 3 the basic architecture for operations on elliptic curves over finite field $GF(2^m)$ was introduced. The adder and the bit-parallel squarer are commonly used in Elliptic curve based cryptosystems. Among the two multipliers introduced, the DSM calculates a multiplication in $n = \lceil \frac{m}{d} \rceil$ clock cycles. The original Karatsuba multiplier calculates a $GF(2^m)$ multiplication in only 1 clock cycle. With registers inserted to improve the critical path performance, the Karatsuba multiplier calculates a $GF(2^m)$ multiplication in 3 clock cycles. For the two structures of divider/inverter, the EEA divider calculates in a fixed time of $2m$+1 clock cycles, the IT inverter reuses the multiplier and the squarer and the calculation time depends on the multiplier structure used in the design. These different operation blocks provides the designer with various ways of configuring the hardware resources in implementing Elliptic curve based cryptosystems.

Chapter 4 studies the common used top-level structures of the pairing based cryptosystems. With all the operation blocks introduced in Chapter 3, the designer is able to build up his own cryptosystem. A bus type top-level architecture for the η_T algorithm of Tate pairing is implemented and the results are given. In this architecture a bus connect all the $GF(2^m)$ components and memory control units. This type of structure can be flexible. With more hardware resource connected to the bus, the system can calculate the algorithm faster. In this work, time, area, and A*T product (area × time) are used to analyse the efficiency of the hardware implementations. As was to be expected, for the same architecture, the more area used, the faster speed achieved. Fore designs over $GF(2^{571})$, among the designs of bus type architecture using DSMs, the design with one DSM of digit size d=32 shows the lowest A*T product, i.e. the best efficiency between area and time. However, the design using Karatsuba multiplier with registers inserted shows the lowest A*T product amongst all bus type architecture designs.

In Chapter 4, the reconfigurable designs of bus type top-level architecture are introduced. As there are 7 multiplications in the main loop of the proposed η_T algorithm, using different number of multipliers requires different scheduling of the operations. With different digit sizes of the DSMs, this work presents the implementation results of the reconfigurable structure. Both DSM and Karatsuba multiplier are used as the $GF(2^m)$ multiplier unit. The implementation results with both area and timing performance are shown. This work was presented in the 6th International Symposium on Applied Reconfigurable Computing 2010 (ARC 2010) [159].

The bus type architecture is reconfigurable, but adds in only multipliers to speed up the multiplications of the η_T pairing algorithms. It does not perform the addition and squaring operations in the "for" loop efficiently. Thus a mixed type top-level architecture which speeds up the other operations in the "for" loop is proposed. This method further compresses the calculation time of the "for" loop by calculating all addition and squaring operations in the loop in parallel. It results in an excellent calculation time. Among all the designs over the mixed type architecture design using Karatsuba multiplier without registers inserted shows the lowest A*T product. However, a large area is required to implement such method.

Chapter 5 considers the side channel analysis (SCA) attacks which make use of the side channel information leaked during the operation of target cryptosystem. A Correlation power analysis attack is applied to the FPGA implemented cryptosystems in this work. The commonly used power consumption models to fit the hardware implementations, the Hamming Weight model and the Hamming Distance model, are studied. This work chose the Hamming Distance model because this model fits the side channel information better.

For the η_T algorithm implementation in this work, the attacks applied are based upon the assumption that the η_T algorithm is to be used in an IBE or an ABE system which works allows two parties to communicate through an insecurity channel. In the insecure channel, it is assumed that an attacker can get access to and can alter the ciphertext sent between the sender and the intended receiver. This assumption is a common condition in public key cryptosystem attacks. This chapter fully analyzes the η_T algorithm in the IBE and ABE schemes and identifies several possible weaknesses in the addition operation and the multiplication operations of the η_T algorithm. The detailed steps for attacking each possible weakness are introduced. The power traces collected from the operating hardware and the calculated correlation results are illustrated. Successful attack results shows that the power consumption model selected in this work fits well the hardware power consumption.

As long as the attacker finds a suitable target and build the computable power consumption model, the correlation power analysis attack can work well with the side channel information collected.

The results of attacks on the pairing implementation with different number of multipliers are presented. The successful attack shows that increasing the amount of hardware resource taken does not help the security aspect of the cryptosystem. Whether with one multiplier or with 7 multipliers in parallel, the side channel attacks will reveal the private information. This CPA attack against the multipliers in the η_T algorithm implementation of bus type top level structure was presented in the 7th International Symposium on Applied Reconfigurable Computing in 2011 (ARC 2011) [160].

In Chapter 6, several important weaknesses when implementing the multiplications in a pairing based cryptosystem are listed from the attacker's point of view. The security analysis in this chapter pointed out that by carefully arranging the multipliers in the η_T pairing algorithm, weaknesses in the $GF(2^m)$ multiplications can be eliminated. However, additional operations must be applied on the original algorithm to protect the $GF(2^m)$ addition. For this reason, this chapter introduced three countermeasures which protect the $GF(2^m)$ operation in the η_T algorithm from the CPA attack through mathematically methods. Along with the same top level architecture introduced in implementing the original η_T algorithm, the implementation results of the three countermeasures are presented. The additional operations needed and the additional operation time required for calculating a secure η_T algorithm using the countermeasures are analyzed. Amongst the countermeasures the bilinearity method requires almost 100% additional calculation time. The other two countermeasures, randomizing the Miller variable and using projective coordinates, incur the same cost which is about 150% of the original η_T algorithm, less than that of the bilinearity method. The implementation results of the designs provide the designers with different choices.

7.2 Future work

From the design and implementation of the basic operation blocks, to the top level structure of the whole algorithm, and then to the flaws in the hardware implemented cryptosystem and how to solve such problems, this thesis presented a full design flow of how to securely implement an η_T pairing algorithm which is suitable for use in IBE and ABE cryptosystems. The functional algorithms in the IBE and ABE schemes are all introduced in this thesis including the point operations, the hash function, and the pairing algorithms. One of the directions of the future work is to complete the peripheral circuits, such as the hash functions, the random number generator and the wrapper which fits the cryptosystem into suitable number of input and output ports, and make the pairing based public key schemes an independent circuit.

In the implementation of $GF(2^m)$ multiplications, the DSM [126] and Karatsuba multiplier[77] are condisered. However, a designer may try to mix these two types of multipliers to take advantage of their properties. There are two ways of combining them: 1) the $d \times m$ multiplication block of the DSM can be composed of several parallel sub-multiplier blocks of the Karatsuba structure; 2) the $n \times n$ sub-multiplier blocks of the Karatsuba multiplier can use digit-serial or bit-serial structures. However, as mentioned by Rebeiro [130], the size of the sub-multiplier blocks of the Karatsuba multiplier should be around 20. Thus the combination of the DSM and the Karatsuba multiplier doesn't suit small digit sized multipliers.

The operations in the η_T algorithm are all based on Elliptic curves. In other Elliptic curve based cryptographies, these operations are widely used as well. The CPA attack provides a test that shows the potential risks that might exist in other cryptosystems. Further work of the CPA attackers should be focused on other protocols and algorithms that make use of the Elliptic curve operation blocks. As proved by this work, the hamming distance model well fits the power consumption of FPGA implementations. Other side channel analysis methods such as fault attacks [161, 162], timing attacks [14], electro-magnetic ratiation [15] attacks can be applied on Elliptic curve based cryptosystems. In power analysis attacks, there also are different approaches such as the template attack [163] and high-order DPA attacks [164]. All these methods provide effective ways of revealing the weaknesses that exist in these public key cryptosystems.

This work only investigated performance and the security aspect of FPGA implemented pairing algorithm designed for IBE and ABE schemes. Such cryptosystems can also be implemented over other hardware such as microprocessors or ASICs, which are of different structures and show different features in the side channel information leaked during operation. Different hardware platforms can lead to different power consumption models to fit the side channel information and may require different methods of generating the hypothetical value of the target data. This can be another interesting direction of research.

The performances investigated in this work mainly focus on the area and timings required for each design. Electrical power and energy are two other parameters that feature in the performance of an implementation. The power P dissipated in the implementation is calculated as per $P = Re[V \cdot I^*]$, where V is the supply voltage, I is the current flow through the circuit, * denotes complex conjugate and $Re[]$ represents the real part of the product. Energy E consumed by the implementation is the product of power and time and can be calculated by $E = P_{avg} \cdot T$, where T is the time the circuit is operating and P_{avg} represent the average power over the time. A fully charged battery contains only a certain amount of energy. Thus , in battery powered system, minimising energy usage helps extend the system lifetime. For this reason, power and energy consumption of the implementations can also be insteresting to designers.

7.3 Publications

The following publications arose as a result of the research presented in this thesis.

- W. Pan, and W. P. Marnane, "A Correlation Power Analysis Attack against Tate Pairing on FPGA". *ARC 2011*, LNCS Vol. 6578, pp. 340-349 (2011)

- W. Pan and W. Marnane, "A Reconfigurable Implementation of the Tate Pairing Computation over $GF(2^m)$". *ARC 2010* pp. 80-91 (2010)

- W. Pan, E. Popovici, S. Lidholm, and L. Marnane, "ASIC Implementation of a Finite Field Arithmetic Processor Core". *Signals and Systems Conference (ISSC 2009), IET Irish*, pp 1-6 (2009)

- Brian Baldwin, Andrew Byrne, Mark Hamilton, Neil Hanley, Robert P. McEvoy, Weibo Pan, William P. Marnane, "FPGA Implementations of SHA-3 Candidates: CubeHash, Grostl, LANE, Shabal and Spectral Hash.". *Digital Systems Design, Euromicro Symposium on -DSD 2009* , pages 783-790. (2009)

APPENDIX

A. SCHEDULE OF “FOR” LOOP OPERATIONS

In each iteration of the “for” loop in steps 3-7 of Algorithm 2.2, the operations can be divided into three parts: the pre-multiplication operations (“PRE” for short), the 7 multiplications, and the post-multipication operations(“POST” for short). Each part requires $11\mathbf{A}+8\mathbf{S}$, $7\mathbf{M}$, and $13\mathbf{A}$ respectively. The operation flow of the “PRE” and “POST” blocks are listed in Fig. A.1 and A.2. The arrangement of the $7\mathbf{M}$ using different number of multipliers are illustrated in Fig. A.3, A.4, A.5, A.6, and A.7.

In these figures, the block signs “A”, “S”, and “M” represent the operations of addition, squaring, and multiplication over $GF(2^m)$ respectively. The operands of each operation are listed on the right hand side of the block signs. The intermediate variables generated by each operation are denoted as “sum”, “sq”, and “p”. These variables will be part of the following calculations or the calculations in the next “for” loop iteration.

The → signs indicate that the results generated by the current operations has a specific name in Algorithm 2.2. There are two kinds of subscription in the figures: The numbers without brackets indicates that this variable is part of an element over $GF(2^{4m})$, i.e. $A(t) = a_0+a_1t+a_2t^2$ and $C(t) = c_0+c_1t+c_2t^2+c_3t^3$; The number with brackets indicates the iteration this variable is going to be calculated. For example, $c_{0(i)}$ was generated in iteration $(i-1)$, and takes part in the calculation of iteration (i) (i is the current iteration number); $c_{0(i+1)}$ is generated in the current iteration and will be an operand in iteration $(i+1)$’s calculation.

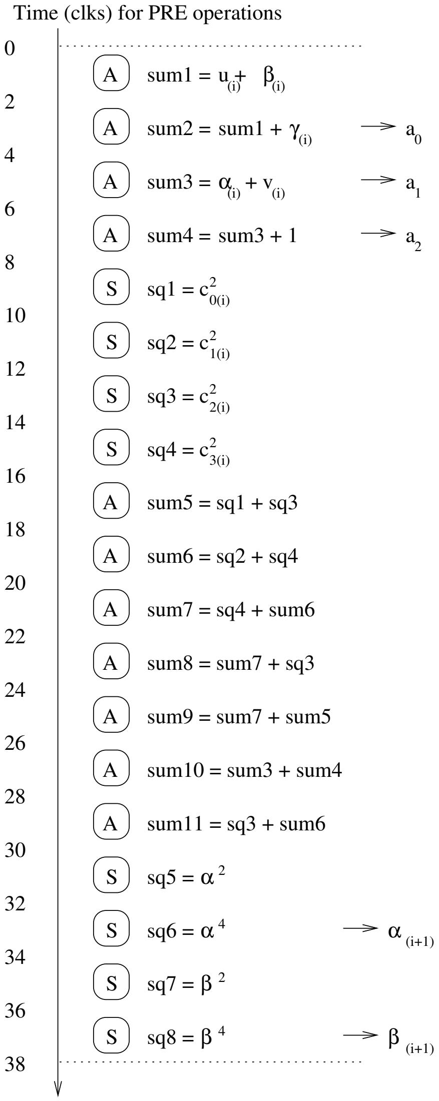


Fig. A.1: Schedule of "PRE" block operations

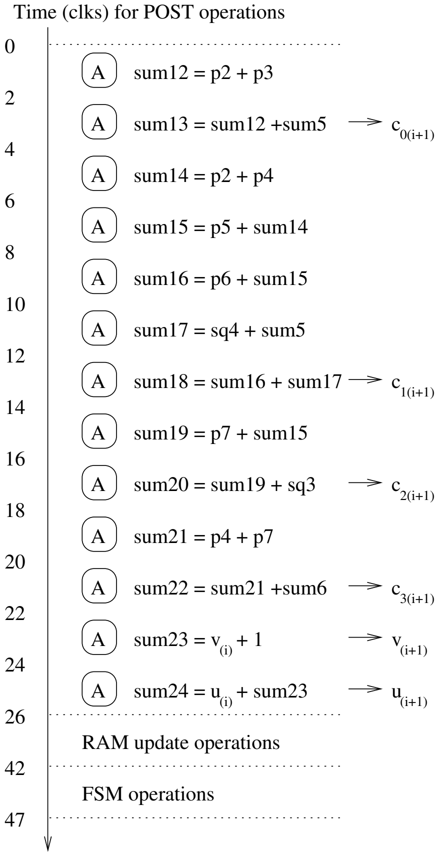

Time (clks) for POST operations
0
A sum12 = p2 + p3
2
A $sum13 = sum12 + sum5 \rightarrow c_{0(i+1)}$
4
A sum14 = p2 + p4
6
A sum15 = p5 + sum14
8
A sum16 = p6 + sum15
10
A sum17 = sq4 + sum5
12
A $sum18 = sum16 + sum17 \rightarrow c_{1(i+1)}$
14
A sum19 = p7 + sum15
16
A $sum20 = sum19 + sq3 \rightarrow c_{2(i+1)}$
18
A sum21 = p4 + p7
20
A $sum22 = sum21 + sum6 \rightarrow c_{3(i+1)}$
22
A $sum23 = v_{(i)} + 1 \rightarrow v_{(i+1)}$
24
A $sum24 = u_{(i)} + sum23 \rightarrow u_{(i+1)}$
26
RAM update operations
42
FSM operations
47

Fig. A.2: Schedule of "POST" block operations

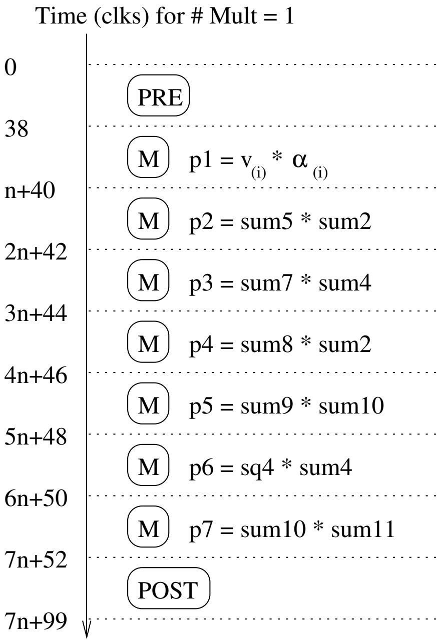


Fig. A.3: Schedule of the 7 multiplications, $\#Mult$=1

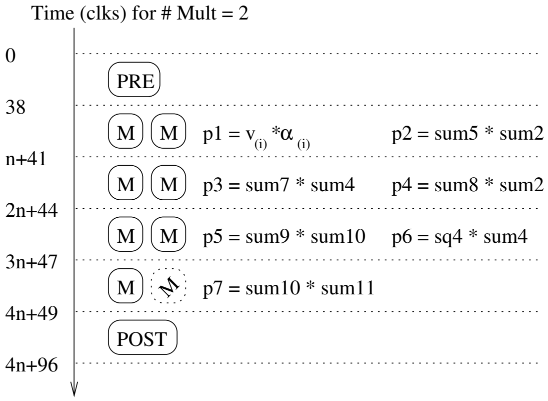


Fig. A.4: Schedule of the 7 multiplications, $\#Mult$=2

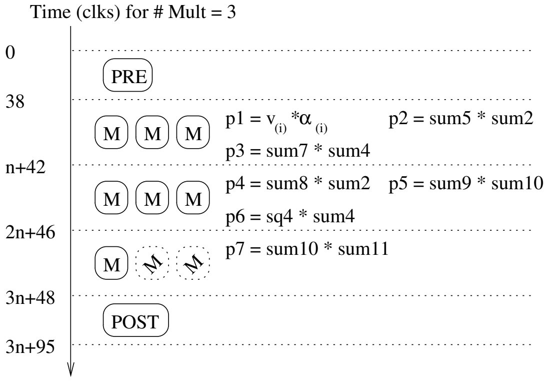


Fig. A.5: Schedule of the 7 multiplications, $\#Mult$=3

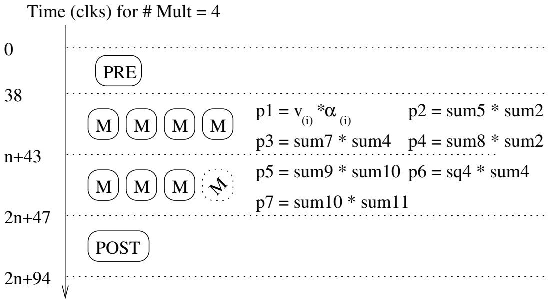


Fig. A.6: Schedule of the 7 multiplications, $\#Mult$=4

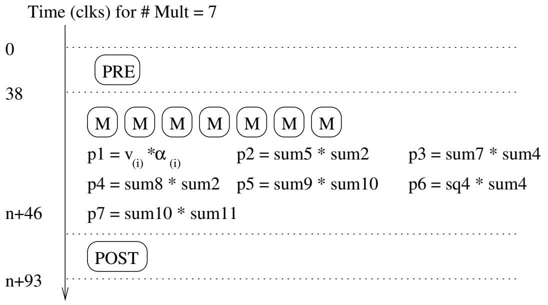


Fig. A.7: Schedule of the 7 multiplications, $\#Mult$=7

B. IMPLEMENTATION RESULTS OF TATE PAIRING

Parameters			Results				
#Mult	d	n	Regs	LUTs	Freq. (MHz)	Time (μs)	A*T (LUT*s)
1× DSM	1	163	2202	2296	248.6	845.3	1.94
	2	82	2202	2457	248.6	456.3	1.12
	4	41	2201	2625	248.6	259.4	0.68
	8	21	2204	2809	248.6	163.3	0.46
	16	11	2211	3478	248.6	115.3	0.40
	32	6	2231	4733	248.6	91.3	0.43
2× DSM	1	163	2701	2641	243.6	520.7	1.38
	2	82	2703	2982	243.6	293.6	0.88
	4	41	2700	3304	243.6	178.7	0.59
	8	21	2706	3676	243.6	122.6	0.45
	16	11	2719	5030	243.6	94.6	0.48
	32	6	2753	7516	243.6	80.6	0.61
3× DSM	1	163	3194	3471	225.1	438.3	1.52
	2	82	3194	3948	225.1	254.8	1.01
	4	41	3190	4453	225.1	161.9	0.72
	8	21	3199	5431	225.1	116.5	0.63
	16	11	3220	7848	225.1	93.9	0.74
	32	6	3273	10952	226.7	82.0	0.90
4× DSM	1	163	3694	4959	217.8	329.5	1.63
	2	82	3695	5596	217.8	201.2	1.13
	4	41	3689	5948	212.6	139.6	0.83
	8	21	3710	7573	217.8	104.5	0.79
	16	11	3729	10890	217.8	88.7	0.97
	32	6	3796	14077	220.1	79.9	1.13
7× DSM	1	163	5196	6587	214.8	207.4	1.37
	2	82	5196	7715	214.8	140.2	1.08
	4	41	5187	8907	214.8	106.3	0.95
	8	21	5208	11250	214.8	89.7	1.01
	16	11	5257	17163	214.8	81.4	1.40
	32	7	5366	22400	209.9	79.9	1.79
Kara.	no Reg	1	1865	8938	158.8	105.3	0.94
Kara.	with Reg	3	5088	8864	244.8	78.1	0.69

Tab. B.1: Implementation results of Tate pairing, bus type, Xilinx Virtex-V, m=163

Parameters			Results				
#Mult	d	n	Regs	LUTs	Freq. (MHz)	Time (μs)	A*T (LUT*s)
1× DSM	1	233	3102	3200	241.4	1717.0	5.49
	2	117	3102	3434	241.4	907.8	3.12
	4	59	3103	3668	241.4	503.2	1.85
	8	30	3106	3912	241.4	300.8	1.18
	16	15	3104	5305	241.4	196.2	1.04
	32	8	3129	6693	243.9	145.9	0.98
2× DSM	1	233	3347	3157	301.3	818.1	2.58
	2	117	3813	4141	233.0	578.4	2.40
	4	59	3815	4608	233.0	338.7	1.56
	8	30	3823	5543	233.0	218.9	1.21
	16	15	3818	7868	233.0	156.9	1.23
	32	8	3857	10436	233.0	127.9	1.34
3× DSM	1	233	4524	4090	222.7	852.6	3.49
	2	117	4524	4791	222.7	477.5	2.29
	4	59	4528	5494	222.7	290.0	1.59
	8	30	4537	6892	222.7	196.3	1.35
	16	15	4533	10378	222.7	147.8	1.53
	32	8	4586	14283	228.0	122.2	1.75
4× DSM	1	233	4535	6738	219.8	614.7	4.14
	2	117	5235	7903	219.8	358.7	2.84
	4	59	5240	8832	219.8	230.8	2.04
	8	30	5252	10681	219.8	166.8	1.78
	16	15	5247	15302	219.8	133.7	2.05
	32	8	5313	19783	216.2	120.2	2.38
7× DSM	1	233	7365	9757	217.8	366.8	3.58
	2	117	7367	11404	217.8	234.7	2.68
	4	59	7372	10013	207.2	177.3	1.78
	8	30	7393	16347	217.8	135.7	2.22
	16	15	7385	24491	217.8	118.6	2.90
	32	8	7494	33654	223.3	107.9	3.63
Kara.	no Reg	1	2634	14420	159.0	149.6	2.16
Kara.	with Reg	3	7301	14478	254.8	106.6	1.54

Tab. B.2: Implementation results of Tate pairing, bus type, Xilinx Virtex-V, m=233

Parameters			Results				
#Mult	d	n	Regs	LUTs	Freq. (MHz)	Time (μs)	A*T (LUT*s)
1× DSM	1	283	3756	3860	226.6	2658.5	10.26
	2	142	3755	4137	226.6	1392.9	5.76
	4	71	3755	4431	226.6	755.6	3.35
	8	36	3757	4993	241.4	414.4	2.07
	16	18	3756	6419	226.6	279.9	1.80
	32	9	3765	8145	229.6	196.5	1.60
2× DSM	1	283	4618	4446	221.8	1604.8	7.13
	2	142	4619	5276	221.8	865.4	4.57
	4	71	4615	5590	221.8	493.1	2.76
	8	36	4622	6988	221.8	309.6	2.16
	16	18	4620	9569	221.8	215.2	2.06
	32	9	4627	12523	227.3	164.0	2.05
3× DSM	1	283	5481	5893	225.4	1211.2	7.14
	2	142	5480	6724	225.4	667.0	4.49
	4	71	5477	7603	225.4	393.0	2.99
	8	36	5485	9374	225.4	257.9	2.42
	16	18	5482	12737	225.4	188.4	2.40
	32	9	5487	17584	229.2	151.1	2.66
4× DSM	1	283	6342	7902	210.9	912.1	7.21
	2	142	6341	9571	215.6	509.6	4.88
	4	71	6337	10743	215.6	317.0	3.41
	8	36	6350	12976	215.6	222.0	2.88
	16	18	6345	17587	215.6	173.2	3.05
	32	9	6348	21882	217.1	147.7	3.23
7× DSM	1	283	8925	11231	212.1	524.0	5.88
	2	142	8925	12764	217.6	317.7	4.05
	4	71	8919	15261	212.1	226.1	3.45
	8	36	8939	19218	212.1	177.0	3.40
	16	18	8932	29199	212.1	151.7	4.43
	32	9	8929	36312	209.2	141.0	5.12
Kara.	no Reg	1	3188	14420	153.7	187.6	2.71
Kara.	with Reg	3	8859	14478	240.9	136.6	1.98

Tab. B.3: Implementation results of Tate pairing, bus type, Xilinx Virtex-V, m=283

C. FEATURES OF VIRTEX FAMILIES

Family	Virtex-2	Virtex-4	Virtex-5	Virtex-6
LUTs per Slice	2	2	4	4
LUTs type	4-in,1-out	4-in,1-out	6-in,1-out	6-in,1-out or 5-in,2-out
Max Freq.	200 MHz	450 MHz	550 MHz	600 MHz

Tab. C.1: Area and Speed Features of Virtex Families 2, 4, 5 and 6 [84, 82, 83, 85]

D. COUNTERMEASURES ALGORITHMS USING PROJECTIVE COORDINATES

Algorithm D.1 Using Jacobian projective coordinates

```
Input : $P(\alpha, \beta), Q(x, y)$
Random Number : $\lambda \neq 0 \in GF(2^m)$
Output : $c = e(P; Q)$
$C(t) \leftarrow 1$
$x \leftarrow x^2, y \leftarrow y^2, \alpha \leftarrow \alpha^4, \beta \leftarrow \beta^4$
$(X, Y, Z) \leftarrow (\lambda^3 x, \lambda^2 y, \lambda)$
$Z \leftarrow Z^6$
$V \leftarrow X, U \leftarrow X + Y + \frac{m-1}{2} Z$
$J \leftarrow \alpha(V + Z), K \leftarrow \beta Z, L \leftarrow \alpha Z$
for $i = 0 : m - 1$
  $A(t) \leftarrow J + K + U + (L + V)t + (L + V + Z)t^2$
  $C(t) \leftarrow C(t)^2 * A(t)$
  $U \leftarrow U + V + Z, V \leftarrow V + Z, \alpha \leftarrow \alpha^4, \beta \leftarrow \beta^4$
  $J \leftarrow \alpha V, K \leftarrow \beta Z, L \leftarrow \alpha Z$
end for
return $C(t) \leftarrow C(t)^{2^{2m-1}}$
      total cost:(11m+59)M+(26m+65)A+(8m+12)S+1I
```

Algorithm D.2 Using Lopez-Dahab projective coordinates

Input : $P(\alpha, \beta), Q(x, y)$

Random Number : $\lambda \neq 0 \in GF(2^m)$

Output : $c = e(P; Q)$

```
$C(t) \leftarrow 1$
$x \leftarrow x^2, y \leftarrow y^2, \alpha \leftarrow \alpha^4, \beta \leftarrow \beta^4$
$(X, Y, Z) \leftarrow (\lambda^2 x, \lambda y, \lambda)$
$Z \leftarrow Z^2$
$V \leftarrow X, U \leftarrow X + Y + \frac{m-1}{2} Z$
$J \leftarrow \alpha(V + Z), K \leftarrow \beta Z, L \leftarrow \alpha Z$
for $i = 0 : m - 1$
  $A(t) \leftarrow J + K + U + (L + V)t + (L + V + Z)t^2$
  $C(t) \leftarrow C(t)^2 * A(t)$
  $U \leftarrow U + V + Z, V \leftarrow V + Z, \alpha \leftarrow \alpha^4, \beta \leftarrow \beta^4$
  $J \leftarrow \alpha V, K \leftarrow \beta Z, L \leftarrow \alpha Z$
end for
return $C(t) \leftarrow C(t)^{2^{2m}-1}$
      total cost:(11m+58)M+(26m+65)A+(8m+11)S+1I
```

E. CALCULATION TIME OF COUNTERMEASURES AGAINST CPA ATTACK

Parameters			Alg.2.2	Alg. 6.1		Alg. 6.2 & 6.3	
#Mult	d	n	Time (μs)	Time (μs)	Ratio	Time (μs)	Ratio
1× DSM	1	163	845.3	1713.5	202.7 %	1282.7	151.7 %
	2	82	456.3	924.4	202.6 %	679.6	148.9 %
	4	41	259.4	524.9	202.4 %	374.3	144.3 %
	8	21	163.3	330.1	202.1 %	225.4	138.0 %
	16	11	115.3	232.6	201.8 %	150.9	130.9 %
	32	6	91.3	183.9	201.5 %	113.7	124.6 %
2× DSM	1	163	520.7	970.5	186.4 %	748.9	143.8 %
	2	82	293.6	551.9	187.9 %	411.8	140.2 %
	4	41	178.7	340.0	190.3 %	241.1	134.9 %
	8	21	122.6	236.6	193.0 %	157.9	128.7 %
	16	11	94.6	184.9	195.5 %	116.2	122.9 %
	32	6	80.6	159.1	197.5 %	95.4	118.5 %
3× DSM	1	163	438.3	806.9	184.1 %	567.2	129.4 %
	2	82	254.8	474.8	186.4 %	323.2	126.9 %
	4	41	161.9	306.7	189.5 %	199.7	123.4 %
	8	21	116.5	224.7	192.8 %	139.5	119.7 %
	16	11	93.9	183.7	195.6 %	109.4	116.5 %
	32	6	82.6	163.2	197.6 %	94.3	114.2 %
4× DSM	1	163	329.5	710.5	215.6 %	462.7	140.4 %
	2	82	201.2	428.6	213.1 %	271.9	135.2 %
	4	41	139.6	292.9	209.9 %	179.7	128.7 %
	8	21	104.5	216.3	206.9 %	128.3	122.7 %
	16	11	88.7	181.5	204.6 %	104.7	118.0 %
	32	6	80.8	164.1	203.1 %	92.9	115.0 %
7× DSM	1	163	207.4	470.0	226.7 %	342.5	165.2 %
	2	82	140.2	308.6	220.1 %	212.0	151.2 %
	4	41	106.3	226.9	213.6 %	145.9	137.3 %
	8	21	89.7	187.1	208.6 %	113.7	126.8 %
	16	11	81.4	167.1	205.4 %	97.6	119.9 %
	32	7	78.1	159.2	203.9 %	91.2	116.8 %
Kara.	no Reg	1	105.3	211.7	201.1 %	105.3	113.7 %

Tab. E.1: Calculation time (μs) of proposed countermeasures, $m = 163$

Parameters			Alg.2.2	Alg. 6.1		Alg. 6.2 & 6.3	
#Mult	d	n	Time (μs)	Time (μs)	Ratio	Time (μs)	Ratio
1× DSM	1	233	1717.0	3467.6	202.0 %	2631.0	153.2 %
	2	117	907.8	1832.9	201.9 %	1371.6	151.1 %
	4	59	503.2	1015.5	201.8 %	741.8	147.4 %
	8	30	300.8	606.8	201.7 %	427.0	141.9 %
	16	15	196.2	395.4	201.5 %	264.1	134.6 %
	32	8	147.4	296.8	201.4 %	188.1	127.6 %
2× DSM	1	233	818.1	1500.0	183.4 %	1190.1	145.5 %
	2	117	578.4	1068.9	184.8 %	825.0	142.6 %
	4	59	338.7	633.5	187.0 %	468.0	138.2 %
	8	30	218.9	415.9	190.0 %	289.5	132.3 %
	16	15	156.9	303.3	193.3 %	197.2	125.7 %
	32	8	127.9	250.7	196.0 %	154.1	120.5 %
3× DSM	1	233	852.6	1531.4	179.6 %	1112.1	130.4 %
	2	117	477.5	868.3	181.8 %	613.1	128.4 %
	4	59	290.0	536.7	185.1 %	363.6	125.4 %
	8	30	196.3	371.0	189.0 %	238.8	121.7 %
	16	15	147.8	285.2	193.0 %	174.3	117.9 %
	32	8	125.1	245.2	196.0 %	144.2	115.2 %
4× DSM	1	233	614.7	1302.4	211.9 %	877.6	142.8 %
	2	117	358.7	754.6	210.4 %	496.1	138.3 %
	4	59	230.8	480.7	208.3 %	305.3	132.3 %
	8	30	166.8	343.8	206.1 %	209.9	125.9 %
	16	15	133.7	272.9	204.2 %	160.5	120.1 %
	32	8	118.2	239.9	202.9 %	137.5	116.3 %
7× DSM	1	233	366.8	811.6	221.3 %	632.1	172.3 %
	2	117	234.7	509.1	216.9 %	373.3	159.0 %
	4	59	177.3	376.1	212.1 %	256.3	144.6 %
	8	30	135.7	282.2	208.0 %	179.2	132.1 %
	16	15	118.6	243.1	205.0 %	145.7	122.9 %
	32	8	110.6	224.8	203.3 %	130.1	117.6 %
Kara.	no Reg	1	149.6	300.8	201.0 %	149.6	113.8 %

Tab. E.2: Calculation time (μs) of proposed countermeasures, $m = 233$

Parameters			Alg.2.2	Alg. 6.1		Alg. 6.2 & 6.3	
#Mult	d	n	Time (μs)	Time (μs)	Ratio	Time (μs)	Ratio
1× DSM	1	283	2658.5	5360.7	201.6 %	4091.1	153.9 %
	2	142	1392.9	2808.3	201.6 %	2117.9	152.1 %
	4	71	755.6	1523.0	201.6 %	1124.4	148.8 %
	8	36	414.4	834.9	201.5 %	595.7	143.8 %
	16	18	279.9	563.6	201.4 %	382.7	136.7 %
	32	9	199.1	400.7	201.3 %	256.8	129.0 %
2× DSM	1	283	1604.8	2921.1	182.0 %	2346.1	146.2 %
	2	142	865.4	1586.7	183.3 %	1243.8	143.7 %
	4	71	493.1	914.8	185.5 %	688.7	139.7 %
	8	36	309.6	583.6	188.5 %	415.1	134.1 %
	16	18	215.2	413.2	192.0 %	274.4	127.5 %
	32	9	168.0	328.1	195.2 %	204.0	121.4 %
3× DSM	1	283	1211.2	2151.2	177.6 %	1585.4	130.9 %
	2	142	667.0	1198.5	179.7 %	861.0	129.1 %
	4	71	393.0	718.8	182.9 %	496.3	126.3 %
	8	36	257.9	482.3	187.0 %	316.5	122.7 %
	16	18	188.4	360.6	191.4 %	224.0	118.9 %
	32	9	153.7	299.8	195.1 %	177.7	115.7 %
4× DSM	1	283	912.1	1916.7	210.1 %	1312.0	143.8 %
	2	142	509.6	1065.3	209.0 %	712.5	139.8 %
	4	71	317.0	657.6	207.5 %	425.0	134.1 %
	8	36	222.0	456.6	205.7 %	283.3	127.6 %
	16	18	173.2	353.2	204.0 %	210.4	121.5 %
	32	9	148.7	301.6	202.7 %	173.9	116.9 %
7× DSM	1	283	524.0	1145.3	218.6 %	921.6	175.9 %
	2	142	317.7	683.5	215.2 %	518.6	163.3 %
	4	71	226.1	477.6	211.2 %	335.9	148.5 %
	8	36	177.0	367.4	207.6 %	239.2	135.2 %
	16	18	151.7	310.7	204.8 %	189.5	124.9 %
	32	9	139.0	282.3	203.1 %	164.6	118.4 %
Kara.	no Reg	1	187.6	377.2	201.0 %	187.6	113.8 %

Tab. E.3: Calculation time (μs) of proposed countermeasures, $m = 283$

REFERENCES

[1] U.S. Code, "COORDINATION OF FEDERAL INFORMATION POLICY", *PUBLIC PRINTING AND DOCUMENTS*, pp:3501-3549, U.S. Code, (2010)

[2] A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone, "Handbook of Applied Cryptography", CRC Press, (1996)

[3] D. Hankerson, A. Menezes, and S. Vanstone, "Guide to Elliptic Curve Cryptography", Springer, (2004)

[4] D. Boneh and M. K. Franklin, "Identity-based encryption from the Weil pairing", *Advances in Cryptology - CRYPTO '01*, LNCS Vol. 2139, pp:213-229, Springer, (2001)

[5] A. Shamir, "Identity-based cryptosystems and signature schemes", *Advances in Cryptology - CRYPTO '84*, LNCS Vol. 196, pp:47-53, Springer, (1985)

[6] A. Sahai and B. Waters, "Fuzzy identity-based encryption", *Advances in Cryptology - EUROCRYPT '05*, LNCS Vol. 3494, pp:457-473, Springer, (2005)

[7] C. Shu, S. Kwon, and K. Gaj, "Reconfigurable Computing Approach for Tate Pairing Cryptosystems over Binary Fields", *IEEE Transaction on Computers*, Vol. 58(9), pp.1221-1237, IEEE, (2009)

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

[9] J. L. Beuchat, "Hardware Architectures for the Cryptographic Tate Pairing", *Pairing-Based Cryptography - Pairing 2012*, presentation, (2012)

[10] S. Ghosh, D. Roychowdhury, A. Das, "High speed cryptoprocessor for η_T pairingon 128-bit secure supersingular elliptic curves over characteristic two fields", *Cryptographic Hardware and Embedded Systems - CHES '11*, LNCS Vol. 6917, pp:442-458, Springer, (2011)

[11] J. L. Beuchat , J. Detrey , N. Estibals , E. Okamoto , F. Rodrí́guez-henrí́quez, “Fast architectures for the η_T pairing over small-characteristic supersingular elliptic curves”, *IEEE Transaction on Computers*, Vol. 60(2), pp:266-281, IEEE, (2011)

[12] M. Keller, R. Ronan, W. Marnane, C. Murphy, “Hardware architectures for the Tate pairing over $GF(2^m)$”, *Computers and Electrical Engineering*, Vol. 33(5-6), pp:392-406, Elsevier, (2007)

[13] P. C. Kocher, J. Jaffe, and B. Jun, “Diferential Power Analysis”, *Advances in Cryptology - CRYPTO ’99*, LNCS Vol. 1666, pp:388-397, Springer, (1999)

[14] P. C. Kocher, “Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and other Systems”, *Advances in Cryptology - CRYPTO ’96*, LNCS Vol. 1109, pp:104-113, Springer, (1996)

[15] K. Gandolfi, C. Mourtel, and F. Olivier, “Electromagnetic Analysis: Concrete Results”, *Cryptographic Hardware and Embedded Systems - CHES ’01*, LNCS Vol. 2162, pp:251-261, Springer, (2001)

[16] D. Page and F. Vercauteren, “Fault and Side-Channel Attacks on Pairing Based Cryptography”, *IEEE Transaction on Computers*, Vol. 55(9), pp:1075-1080, IEEE, (2006)

[17] M. Scott, “Computing the Tate Pairing”, *Topics in Cryptology - CT-RSA 2005*, LNCS Vol. 3376, pp:293-304, Springer, (2005)

[18] P. C. Kocher, J. Jaffe, and B. Jun, “Differential power analysis”, *Advances in Cryptology - CRYPTO ’99*, LNCS Vol. 1666, pp:388-397, Springer, (1999)

[19] R. M. Avanzi, H. Cohen, C. Doche, G. Frey, T. Lange, K. Nguyen and F. Vercauteren, “Handbook of Elliptic and Hyperelliptic Curve Cryptography”, *Discrete Mathematics and Its Applications*, Chapman & Hall/CRC, (2006)

[20] V. S. Miller, “Use of elliptic curves in cryptography”, *Advances in Cryptography - CRYPTO ’85*, LNCS Vol. 218, pp:417-426, Springer, (1986)

[21] N. Koblitz “Elliptic curve cryptosystems”, *Mathematics of Computation*, Vol. 48, pp:203-209, AMS, (1987)

[22] J. H. Graff, “The Arithmetic of Elliptic Curves”, *Graduate Texts in Mathematic*, Vol. 106, Springer, (1994)

[23] National Institute of Standards and Technology, “FIPS Publication 186: Digital Signature Standard”, NIST, (1994)

[24] J.L. Beuchat, N. Brisebarre, J. Detrey, E. Okamoto, and F. R. Henríquez, “A Comparison between Hardware Accelerators for the Modified Tate Pairing over F_{2^m} and F_{3^m}”, *Pairing-Based Cryptography - Pairing 2008*, LNCS Vol. 5209, pp:297-315, Springer, (2008)

[25] R. Granger, D. Page, and N. P. Smart, “High Security Pairing-Based Cryptography Revisited”, *Algorithmic Number Theory*, LNCS Vol. 4076, pp:480-494, Springer, (2006)

[26] I. Duursma and H. Lee, “Tate pairing implementation for hyperelliptic curves $y^2 = x^p - x + d$”, *Advances in Cryptology - ASIACRYPT ’03*, LNCS Vol. 2894, pp:111-123, Springer, (2003)

[27] A. Weimerskirch, C. Paar, and S. C. Shantz, “Elliptic Curve cryptography on a Palm OS device”, *6th Australiasian Conference on Information Security and Privacy - ACISP ’01*, LNCS Vol. 2119, pp:502-513, Springer, (2001)

[28] A. Dabholkar and K. Choong Yow, “Efficient implementation of elliptic curve cryptography (ECC) for personal digital assistants (PDAs)”, *Wireless Personal Communications*, Vol. 29(3-4), pp:233-246, Kluwer Academic Publishers, (2004)

[29] L. B. Oliveira, D. F. Aranha, E. Morais, F. Daguano, J. Lopez, and R. Dahab, “TinyTate: Computing the tate pairing in resource-constrained sensor nodes”, *Network Computing and Applications - NCA ’07*, pp:318-323, IEEE, (2007)

[30] L. B. Oliveira, M. Scott, J. Lopez, and R. Dahab, “TinyPBC: Pairings for authenticated identity-based non-interactive key distribution in sensor networks”, *Computer Communications*, Volume 34(3), pp:485-493, Elsevier, (2007)

[31] A. Menezes, “Elliptic Curve Public Key Cryptosystems”, Kluwer Academic Publishers, (1993)

[32] IEEE, “IEEE standard specifications for public-key cryptography, 2000”, *IEEE Standard P1363 project*, IEEE, (2000)

[33] J. Lopez and R. Dahab, "Improved algorithms for elliptic curve arithmetic in $GF(2^n)$", *Selected Areas in Cryptography - SAC '98*, LNCS Vol. 1556, pp: 201-212, Springer, (1999)

[34] A. A. Bruen and M. A. Forcinito, "Cryptography, Information Theory and Error-Correction", *A Handbook for the* 21^{st} *Century*, John Wiley & Sons, Inc., (2005)

[35] R. Anderson, "Security Engineering", John Wiley & Sons, Inc.,(2001)

[36] D. B. Parker, "Toward a New Framework for Information Security", *In Computer Security Handbook*, John Wiley & Sons, Inc., (2002)

[37] B. Schneier, "Applied Cryptography", John Wiley & Sons, Inc., (1996)

[38] M. Copeland, J. Grahn, D. A. Wheeler, "The GNU Privacy Handbook", The Free Software Foundation, (1999)

[39] J. H. Ellis, "The Story Of Non-Secret Encryption", http://cryptome.org/jya/ellisdoc.htm, Accessed 02/09/2013, (1997)

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

[41] J. R. Vacca, "Public Key Infrastructure: Building Trusted Applications and Web Services", *Taylor & Francis Group*, (2004)

[42] L. M. Kohnfelder, "Towards a Practical Public-Key Cryptosystem", *Bachelor Thesis*, Department of Electrical Engineering, Massachusetts Institute of Technology, (1978)

[43] V. Goyal,O. Pandey, A. Sahai ,B. Waters, "Attribute-based encryption for fine-grained access control of encrypted data", *Computer and Communications Security - CCS '06*, pp:89-98, ACM, (2006)

[44] J. Bethencourt, A. Sahai, B. Waters, "Ciphertext-policy attribute-based encryption", *IEEE Symposium on Security and Privacy*, pp:321-334, IEEE, (2007)

[45] L. Cheung, C. Newport, "Provably secure ciphertext policy ABE", *Computer and communications security - CCS '07*, pp:456-465, ACM, (2007)

[46] V. Goyal, A. Jain, O. Pandey, A. Sahai, “Bounded ciphertext policy attribute based encryption”, *Automata, Languages and Programming 2008*, LNCS Vol. 5126, pp:579-591. Springer, (2008)

[47] T. Nishide, K. Yoneyama, K. Ohta, “Attribute-based encryption with partially hidden encryptor-specified access structures”, *Applied Cryptography and Network Security*, LNCS Vol. 5037, pp:111-129, Springer, (2008)

[48] B. Waters, “Ciphertext-policy attribute-based encryption: An expressive, efficient, and provably secure realization”, *Public Key Cryptography - PKC 2011*, LNCS Vol. 6571, pp:53-70, Springer, (2011)

[49] J. J. Rotman, “An introduction to the Theory of Groups”, *Graduate Texts in Mathematics*, Vol. 148, Springer, (1995)

[50] N. L. Biggs, “Cods: An Introduction to Information Communication and Cryptography”, Springer, (2008)

[51] I. Blake, G. Seroussi, and N. Smart, “Elliptic Curves in Cryptography”, *London Mathematical Society*, Lecture Note Series 265, Cambridge University Press, (1999)

[52] D. Hankerson, A. Menezes, and S. Vanstone, “Guide to Elliptic Curve Cryptography”, Springer, (2004)

[53] E. Wenger, M. Hutter, “Exploring the Design Space of Prime Field vs. Binary Field ECC-Hardware Implementations”, *Secure IT systems Nordic conference*, LNCS Vol. 7161, pp:256-272, Springer, (2012)

[54] D. Page and N. Smart, “Hardware implementation of Finite Fields of characteristic three”, *Cryptographic Hardware and Embedded Systems - CHES ’02*, LNCS Vol. 2523 pp:529-539, Springer, (2003)

[55] O. A. B. Birkedal, “Counting Points on Elliptic Curves”, *Master Thesis*, Department of Mathematical Sciences, Norwegian University of Science and Technology, (2010)

[56] J. H. Silverman, “An Introduction to the Theory of Elliptic Curves”, *Summer School on Computational Number Theory and Applications to Cryptography*, University of Wyoming, (2006)

[57] T. ElGamal, "A public-key cryptosystem and a signature scheme based on discrete logarithms", *IEEE Transactions on Information Theory*, Vol. 31(4), pp:469-472, IEEE, (1985)

[58] National Institute of Standards and Technology, "RECOMMENDED ELLIPTIC CURVES FOR FEDERAL GOVERNMENT USE", *http://csrc.nist.gov/groups/ST/toolkit/documents/dss/NISTReCur.pdf*, Accessed 15/12/2012, NIST, (1999)

[59] J. H. Silverman, "The arithmetic of elliptic curves", Springer, (1995)

[60] U. Maurer and S. Wolf, "The diffie-hellman protocol", *Designs, Codes and Cryptography*, Vol. 19, pp:147-171, Springer, (2000).

[61] A. J. Menezes, T. Okamoto and S. A. Vanstone, "Reducing Elliptic Curve Logarithms to Logarithms in a Finite Field", *IEEE Transaction on Infomation Theory*, Vol. 39(5), pp:1639-1646, IEEE, (1993)

[62] G. Frey and H. G. R*ü*ck, "A remark concerning m-divisibility and the discrete logarithm problem in the divisor class group of curves", *Mathematics of Computation*, Vol. 62(206), pp:865-874, AMS, (1994)

[63] G. Frey, M. Muller and H. G. R*ü*ck, "The Tate pairing and the discrete logarithm applied to elliptic curve cryptosystems", *IEEE Transaction on Information Theory*, Vol. 45(5), pp:1717-1719, IEEE, (1999)

[64] V. Miller, "Short Programs for functions on curves", *unpublished manuscript*, Available from http://crypto.stanford.edu/miller/miller.pdf, (1986)

[65] P. Barreto, H. Kim, B. Lynn, and M. Scott, "Efficient algorithms for pairing based cryptosystems", *Advances in Cryptology - Crypto '02*, LNCS Vol. 2442, pp:354-368, Springer, (2002)

[66] S. D. Galbraith, "The Weil pairing on elliptic curves over C", IACR Cryptology ePrint Archive, (2005)

[67] G. Frey, "Applications of arithmetical geometry to cryptographic constructions", *Finite Fields and Applications*, pp:128-161, Springer, (2001)

[68] S. Kwon, "Efficient Tate Pairing Computation for Elliptic Curves Over Binary Fields", *Australasian Conference Information Security and Privacy - ACISP '05*, LNCS Vol. 3574, pp:134-145, Springer, (2005)

[69] E.R. Verheul, "Evidence that XTR is more secure than supersingular elliptic curve cryptosystems", *Journal of Cryptology*, Vol. 17(4), pp:277-296, Springer, (2004)

[70] A. J. Menezes, "Elliptic Curve Public Key Cryptosystems", Kluwer Academic Publisher, (1993)

[71] I. Blake, G. Seroussi, and N. Smart, "Advances in Elliptic Curves Cryptography", Cambridge University Press, (2005)

[72] R. Lidl and H. Niederreiter, "Finite Fields", *Encyclopedia of Mathematics and its Applications*, Vol. 20, Addison-Wesley, (1983)

[73] N. P. Smart, "How Secure are elliptic curves over composite extension fields?", *Advances in Cryptology - EUROCRYPT '01*, LNCS Vol. 2045, pp:30-39, Springer, (2001)

[74] K. McCusker, N. O'Connor, and D. Diamond, "Low-energy finite field arithmetic primitives for implementing security in wireless sensor networks", *International Conference on Communications, Circuits And Systems - ICCCAS '06*, pp:1537-1541, IEEE, (2006)

[75] Certicom, "The Certicom ECC challenge", *http://www.certicom.com/index.php/the-certicom-ecc-challenge* , Accessed 20/03/13.

[76] E. Thomé, "Computation of discrete logarithms in $\mathbb{F}_{2^{607}}$", *Advances in Cryptology - ACRYPT '01*, LNCS Vol. 2248, pp:107-124, Springer, (2001)

[77] S. C. Shantz, A. Karatsuba, Y. Ofman, "Multiplication of many digital numbers by automatic computers", *Translation in Physics-Doklady 7*, pp:595-596, Doklady Akad. Nauk SSSR, (1963)

[78] A. Joux, "A one round protocol for tripartite Diffie-Hellmans", *Journal of Cryptology*, Vol. 17(4), pp:263-276, Springer, (2004)

[79] R. Sakai, K. Ohgishi and M. Kasahara, "Cryptosystems based on pairing", *Symposium on Cryptography and Information Security - SCIS '00*, (2000)

[80] R. Dutta, R. Barua, and P. Sarkar, "Pairing-based cryptographic protocols: A survey", *Cryptology ePrint Archive*, Report 064/2004, 2004. http://eprint.iacr.org/2004/064.pdf, Accessed 12/12/2012, (1996)

[81] Certicom, “STANDARDS FOR EFFICIENT CRYPTOGRAPHY”, *Certicom Research*, (2000)

[82] Xilinx, “Virtex-2 Family Overview”, *Available from http://www-mtl.mit.edu/Courses/6.111/labkit/datasheets/virtex2datasheet.pdf*, accessed 07/03/2013

[83] Xilinx, “Virtex-4 Family Overview”, *Available from http://www.xilinx.com/support/documentation/data_sheets/ds112.pdf*, accessed 07/03/2013

[84] Xilinx, “Virtex-5 Family Overview”, *Available from http://www.xilinx.com/support/documentation/data_sheets/ds100.pdf*, accessed 04/03/2013

[85] Xilinx, “Virtex-6 Family Overview”, *Available from http://www.xilinx.com/support/documentation/data_sheets/ds150.pdf*, accessed 07/03/2013

[86] T. Iyama, S. Kiyomoto, K. Fukushima, T. Tanaka, T. Takagi, “Efficient Implementation of Pairing on BREW Mobile Phones”, *Advances in Information and Computer Security*, LNCS Vol. 6434, pp:326-336, Springer, (2010)

[87] D. F. Aranha, J.-L. Beuchat, J. Detrey, N. Estibals, “Optimal Eta Pairing on Supersingular Genus-2 Binary Hyperelliptic Curves”, *Topics in Cryptology - CT-RSA ’12*, LNCS Vol. 7178, pp 98-115, Springer, (2012)

[88] N. Koblitz and A. Menezes, “Pairing-based cryptography at high security levels”, *Cryptography and Coding*, LNCS Vol. 3796, pp:13-36, Springer, (2005)

[89] S. C. Pohlig, “Algebraic and Combinatoric Aspects of Cryptography”, *Thechnical report* , Vol. 6602(1), Information systems laboratory, Stanford University, (1977)

[90] S. C. Pohlig and M. E. Hellman, “An improved algorithm for computing logarithms over GF(p) and its cryptographic significance”, *IEEE Transaction on Information Theory*, Vol. 24(1), pp:106-110, IEEE, (1978)

[91] D. Shanks, “Class number, a theory of factorization, and genera”, *Proceedings of Symposia in Pure Mathematica 20*, pp:415-440, Number Theory Institute, (1971)

[92] J. M. Pollard, “Monte Carlo methods for index computation (mod p)”, *Mathematics of Computation*, Vol. 32(143), pp:918-924, AMS, (1978)

[93] National Institute of Standards and Technology, “Recommendation for Key Management”, *Special Publication 800-57*, part 1 Rev.3, NIST, (2011)

[94] F. X. Standaert, G. Rouvroy, J. J. Quisquater, “FPGA Implementations of the DES and Triple-DES Masked Against Power Analysis Attacks”, *Field Programmable Logic and Applications - FPL ’06*, pp:1-4, IEEE, (2006)

[95] T. S. Messerges, E. A. Dabbish, and R. H. Sloan, “Examining smart-card security under the threat of power analysis attacks”, *IEEE Transactions on Computers*, Vol. 51(5), pp:541-552, IEEE, (2002)

[96] E. Brier, C. Clavier, and F. Olivier, “Correlation Power Analysis with a Leakage Model”, *Cryptographic Hardware and Embedded Systems - CHES 2004*, LNCS Vol. 3156, pp:135-152, Springer, (2004)

[97] M. Joye, “Elliptic Curves and Side-Channel Analysis”, *ST Journal of System Research*, Vol. 4(1), pp:283-306, STMicroelectronics, (2003)

[98] M. Joye and S. Yen, “The Montgomery Powering Ladder”, *Cryptographic Hardware and Embedded Systems - CHES ’02*, LNCS Vol. 2523, pp:291-302, Springer, (2003)

[99] 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*, Vol. 2(10), pp:52-62, Elsevier, (2007)

[100] Texas Instruments, “CMOS Power Consumption and Cpd Calculation”, TI, (1997)

[101] F. X. Standaert, S. B. Örs, J.J. Quisquater, and B. Preneel , “Power Analysis Attacks against FPGA Implementations of the DES”, *Field-Programmable Logic and Applications - FPL ’04*, LNCS Vol. 3203, pp:84-94, Springer, (2004)

[102] T. S. Messerges, E. A. Dabbish, and R. H. Sloan, “Investigations of Power Analysis Attacks on Smartcards”, *Proceedings of USENIX Workshop on Smartcard Technology*, pp:151-162, The USENIX Association, (1999)

[103] M. L. Akkar, R. Bévan, P. Dischamp, and D. Moyart, "Power analysis, what is now possible...", *Advances in Cryptology - ASIACRYPT '00*, LNCS Vol. 1976, pp:489-502, Springer, (2000)

[104] P. Grabher, Johann Groβschädl, and D. Page, "Non-deterministic processors: FPGA-based analysis of area, performance and security", *Workshop on Embedded Systems Security Article No. 1 - WESS '09*, pp:1-10, ACM, (2009)

[105] C. Clavier, J.-S. Coron, and N. Dabbous, "Differential power analysis in the presence of hardware countermeasures", *Cryptographic Hardware and Embedded Systems - CHES '00*, LNCS Vol. 1965, pp:252-263, Springer, (2000)

[106] P. S. L. M. Barreto, S. Galbraith, C. O'hEigeartaigh, and M. Scott, "Efficient pairing computation on supersingular Abelian varieties", *Designs, Codes and Cryptography*, Vol. 42(3), pp:239-271, Springer, (2007)

[107] H. Fan and M. A. Hasan, "Fast bit parallel-shifted polynomial basis multipliers in $GF(2^n)$", *IEEE Transactions on Circuits and Systems I: Regular Papers*, Vol. 53(12), pp:2606-2615, IEEE, (2006)

[108] Xilinx, *http://www.xilinx.com/*, accessed 22/01/2013

[109] Altera, *http://www.altera.com/*, accessed 22/01/2013

[110] Research Center for Information Security, "National Institute of Advanced Industrial Science and Technology", *Side-channel Attack Standard Evaluation Board SASEBO-GII Specification* , Version 1.01, RCIS, (2009)

[111] Xilinx, "Spartan-3E FPGA Family: Data Sheet, 2010", *Available from http://www.xilinx.com/support/documentation/data_sheets/ds529.pdf*, accessed 22/01/2013

[112] S. Gao and D. Panario, "Tests and Constructions of Irreducible Polynomials over Finite Fields", *Foundations of Computational Mathematics*, pp:346-361, Springer, (1997)

[113] M. O'Rabin, "Probabilistic algorithms in finite fields", *Foundations of Computer Science - SFCS '81*, pp:394-398, IEEE, (1981)

[114] J. V. Z. Gathen and J. Gerhard, "Arithmetic and factorization of polynomials over $\mathbb{F}_2$", *International Symposium on Symbolic and Algebraic Computation - ISSAC '96*, pp:1-9, ACM, (1996)

[115] J. V. Z. Gathen and V. Shoup, "Computing Frobenius Maps and Factoring Polynomials", *Computational Complexity*, Vol. 2(3), pp:187-224, Birkhäuser, (1992)

[116] S. Ronan, "Field Theory", Springer, (1995)

[117] A. J. Menezes, I. F. Blake, X. Gao, R. C. Mullin, S. A. Vanstone, and T. Yaghoobian, "Applications of Finite Fields", Kluwer Academic Publishers, (1993)

[118] T. Beth, B. M. Cook, D. Gollmann, "Architectures for Exponentiation in $GF(2^n)$", *Advances in Cryptology - CRYPTO '86*, LNCS Vol. 263, pp:302-310, Springer, (1987)

[119] G. Meurice de Dormale and J.-J. Quisquater, "High-speed Hardware implementations of elliptic curve cryptography: A survey", *Journal of Systems Architecture*, Vol. 53(2-3), pp:72-84, Elsevier, (2007)

[120] E. D. Mastrovito, "VLSI Architectures for Computation in Galois Fields", *PhD Thesis*, Department of Electrical Engineering, Linkoping University, (1991)

[121] C. Paar, "Efficient VLSI Architectures for Bit-Parallel Computation in Galois Fields", *PhD Thesis*, Institute for Experimental Mathematics, Universigy of Essen, (1994)

[122] C. Negre, "Efficient parallel multiplier in shifted polynomial basis", *Journal of Systems Architecture*, Vol. 53(2-3), pp:109-116, Elsevier, (2007)

[123] J. L. Imana and J. M. Sánchez, "Efficient reconfigurable implementation of canonical and normal basis multipliers over Galois fields $GF(2^m)$ generated by AOPs", *Journal of VLSI Signal Processing Systems*, Vol. 42(3), pp:285-296, Kluwer Academic Publishers, (2006)

[124] S. E. Tavares, P. A. Scott, and L. E. Peppard, "A fast VLSI multiplier for $GF(2^m)$", *IEEE Journal on Selected Areas in Communications*, Vol. 4(1): pp:62-66, IEEE, (1986)

[125] E. Ferrer, D. Bollman, and O. Moreno, "A fast finite field multiplier", *Reconfigurable Computing: Architectures, Tools and Applications - ARC '07*, LNCS Vol. 4419, pp:238-246. Springer, (2007)

[126] L. Song and K. Parhi, "Low energy digit-serial/parallel finite field multipliers", *Journal of VLSI Signal Processing Systems*, Vol. 19(2), pp:149-166, Kluwer Academic Publishers, (2006)

[127] M. H*ü*tter, J. Groβsch*ä*dl, G. A. Kamendje, "A Versatile and Scalable Digit-Serial/Parallel Multiplier Architecture for Finite Fields $GF(2^m)$", *Information Technology: Coding and Computing - ITCC '03*, pp:692-700, IEEE, (2003)

[128] J.-L. Beuchat, T. Miyoshi, Y. Oyama, and E. Okamoto, "Multiplication over F_{p^m} on FPGA: A Survey", *Reconfigurable Computing: Architectures, Tools and Applications - ARC '07*, LNCS Vol. 4419, pp:214-225, Springer, (2007)

[129] S. Ghosh, D. Roychowdhury, and A. Das "High Speed Cryptoprocessor for η_T Pairing on 128-bit Secure Supersingular Elliptic Curves over Characteristic Two Fields", *Cryptographic Hardware and Embedded Systems - CHES '11*, LNCS Vol. 6917, pp:442-458, Springer, (2011)

[130] C. Rebeiro and D. Mukhopadhyay, "High Speed Compact Eliptic Curve Cryptoprocessor for FPGA Platforms", *Progress in Cryptology - INDOCRYPT '08*, LNCS Vol. 5365, pp:367-388, Springer, (2008)

[131] S. W. Golomb, "Combinatorial proof of Fermat's "Little" Theorem", *The American Mathematical Monthly*, Vol. 63(10), pp:718, Mathematical Association of America, (1956)

[132] T. Itoh and S. Tsujii, "A fast algorithm for computing multiplicative inverses in $GF(2^m)$ using normal bases", *Information and Computation*, Vol. 78(3), pp:171-177, Elsevier, (1956)

[133] J. H. Guo, "Systolic array implementation of Euclid's algorithm for inversion and division in $GF(2^m)$", *IEEE Transactions on Computers*, Vol. 47(10), pp:1161-1167, IEEE, (1998)

[134] H. Brunner, A. Curgier, and M. Hofstetter, "On computing multiplicative inverses in $GF(2^m)$", *IEEE Transactions on Computers*, Vol. 42(8), pp:1010-1015, IEEE, (1993)

[135] E. R. Berlekamp, "Algebraic Coding Theory", Aegean Park Press, (1968)

[136] K. Araki, I. Fujita, and M. Morisue, “Fast inverters over finite fields based on Euclids algorithm”, *Transactions of the Institute of Electronics, Information and Communication Engineers E*, Vol. E72(11), pp:1230-1234, IEICE, (1989)

[137] M. Oloffson, “VLSI Aspects on Inversion in Finite Field”, *PhD Thesis*, Department of Electrical Engineering, Linköping University, (2002)

[138] Y. Watanabe, N. Takagi, and K. Takagi, “A VLSI algorithm for division in $GF(2^m)$ based on the extended binary GCD algorithm”, *IEICE Transactions on fundamentals of Electronics, Communications and Computer Sciences*, Vol. E85-A(5), pp:994-999, IEICE, (2002)

[139] S. C. Shantz, “From Euclid’s GCD to Montgomery Multiplication to the Great Divide”, *Techchnical Report SMLI TR-2001-95*, pp:1-10, Sun Microsystems, (2001)

[140] T. Kerins, “Architectures for Cryptography Based on Elliptic Curves”, *PhD Thesis*, Electrical & Electronic Engineering Department, University College Cork, (2005)

[141] M. Keller, T. Kerins, W. P. Marnane, “FPGA implementation of a $GF(2^{4m})$ multiplier for use in pairing based cryptosystems”, *Reconfigurable Computing: Architectures and Applications - ARC ’05*, LNCS Vol. 3985, pp:358-369, Springer, (2005)

[142] C. H. Lim and H. S. Hwang, “Fast implementation of elliptic curve arithmetic in $GF(p^n)$”, *Public Key Cryptography - PKC ’00*, LNCS Vol. 1751, pp:405-421, Springer, (2000)

[143] M. Keller, “Efficient Architectures for Elliptic Curve Based Cryptography”, *PhD Thesis*, Department of Electrical and Electronic Engineering, University College Cork, (2008)

[144] C. Whelan and M. Scott, “Side channel analysis of practical pairing implementations: Which path is more secure?”, *Progress in Cryptology - VIETCRYPT 2006*, LNCS Vol. 4341, pp:81-98, Springer, (2006)

[145] T. H. Kim, T. Takagi,D. G. Han, H. W. Kim, and J. Lim, “Side Channel Attacks and Countermeasures on Pairing Based Cryptosystems over Binary Fields”, *Cryptology and Network Security - CANS 2006* LNCS Vol. 4301, pp:168-181, Springer, (2006)

[146] D. Montgomery and G. Runger, "Applied Statictic and Probability for Engineers", John Wiley & Sons, Inc., (2002)

[147] F. X. Standaert, S. B. Örs, J. J. Quisquater, and B. Preneel , "Power Analysis Attacks against FPGA Implementations of the DES", *Field-Programmable Logic and Applications - FPL '04*, LNCS Vol. 3203, pp:84-94, Springer, (2004)

[148] T. S. Messerges, E. A. Dabbish, and R. H. Sloan, "Investigations of Power Analysis Attacks on Smartcards", *Proceedings of USENIX Workshop on Smartcard Technology - WOST '99*, pp:151-162, USENIX, (1999)

[149] J. Pan, J. I. den Hartog, and E. P. de Vink, "An Operation-Based Metric for CPA Resistance", *World Computer Congress - WCC '08*, IFIP Vol. 278, pp:429-443, Springer, (2008)

[150] J.-S. Coron, P. Kocher, and D. Naccache, "Statistics and secret leakage.", *Financial Cryptography - FC '00*, LNCS Vol. 1972, pp:157-173, Springer, (2001)

[151] Research Center for Information Security, National Institute of Advanced Industrial Science and Technoloty, "Side-channel Attack Standard Evaluation Board SASEBO-GII Specification", Version 1.0, RCIS, (2009)

[152] Mathworks, "System Requirements - Release 2008a", *Available from http://www.mathworks.co.uk/support/sysreq/release2008a/*, accessed 22/03/2013, (2008)

[153] K. Fong, D. Hankerson, J. López, and A. Menezes, "Field inversion and point halving revisited", *IEEE Transactions on Computers 2004*, Vol. 53(8), pp:1047-1059, IEEE, (2004)

[154] C. H. Lim and H. S. Hwang, "Fast implementation of elliptic curve arithmetic in $GF(p^n)$", *Public Key Cryptography - PKC '00*, LNCS Vol. 1751, pp:405-421, Springer, (2000)

[155] D. Eastlake, S. Crocker, J. Schiller, "Randomness Recommendations for Security", RFC Editor, (1994)

[156] S. H. M. Kwok, and E. Y. Lam, "FPGA-based High-speed True Random Number Generator for Cryptographic Applications", *IEEE Region 10 Conference - TENCON '06*, pp:1-4, IEEE, (2006)

[157] V. Fischer, and M. Drutarovský, "True Random Number Generator Embedded in Reconfigurable Hardware", *Cryptographic Hardware and Embedded Systems - CHES '02*, LNCS Vol. 2523, pp:415-430 , Springer, (2003)

[158] J. Kelsey, B. Schneier, D. Wagner, and C. Hall, "Cryptanalytic Attacks on Pseudorandom Number Generators", *Fast Software Encryption - FSE '98*, LNCS Vol. 1372, pp:168-188 , Springer, (1998)

[159] W. Pan and W. P. Marnane, "A Reconfigurable Implementation of the Tate Pairing Computation over $GF(2^m)$", *Reconfigurable Computing: Architectures, Tools and Applications - ARC '10* LNCS Vol. 5992, pp:80-91, Springer, (2010)

[160] W. Pan, and W. P. Marnane, "A Correlation Power Analysis Attack against Tate Pairing on FPGA", *Reconfigurable Computing: Architectures, Tools and Applications - ARC '11*, LNCS Vol. 6578, pp:340-349, Springer, (2011)

[161] I. Biehl, B. Meyer, and V. Muller, "Differential fault analysis on elliptic curve cryptosystems", *Advances in Cryptology - CRYPTO '00*, LNCS Vol. 1880 pp:131-146, Springer, (2000)

[162] D. Boneh, R. A. DeMillo, and R. J. Lipton, "On the importance of checking cryptographic protocols for faults", *Advances in Cryptology - EUROCRYPT '97*, LNCS Vol. 1233 pp:37-51, Springer, (1997)

[163] S. Chari, J. R. Rao, P. Rohatgi, "Template Attacks", *Cryptographic Hardware and Embedded Systems - CHES '02*, LNCS Vol. 2523, pp:13-28, Springer, (2003)

[164] T. S. Messerges, "Using Second-Order Power Analysis to Attack DPA Resistant Software", *Cryptographic Hardware and Embedded Systems - CHES '00*, LNCS Vol. 1965, pp:238-251, Springer, (2000)