A Microcoded Elliptic Curve Cryptographic Processor

LEUNG Ka Ho

A Thesis Submitted in Partial Fulfillment
of the Requirements for the Degree of
Master of Philosophy
in
Computer Science and Engineering

August 2001

A Microcoded Elliptic Curve Cryptographic Processor

Submitted by

LEUNG Ka Ho

for the degree of Master of Philosophy
at The Chinese University of Hong Kong in August 2001

Abstract

Elliptic curve cryptography is a public key cryptosystem based on the elliptic curve discrete logarithm problem. The reason for the attractiveness of ECC is that there is no sub-exponential algorithm known to solve the elliptic curve discrete logarithm problem. This leads to lower memory requirements, lower computational requirements and higher security than other public key cryptosystems.

This thesis describes an implementation of a microcoded elliptic curve processor using FPGA technology. This processor implements optimal normal basis field operations in F_{2^n}. The design is generated by a parameterized module generator which can accommodate arbitrary n and also produce field multipliers with different speed/area tradeoffs. The control part of the processor is microcoded, enabling curve operations to be incorporated into the processor hence reducing the chip's I/O requirements. The microcoded approach also facilitates rapid development and algorithmic optimization; for example, projective and affine coordinates were supported using different sets of microcodes. The design was successfully tested on a Xilinx Virtex XCV1000–6 device and could perform an elliptic curve multiplication over the field F_{2^n} using affine and projective coordinates for $n = 113$, 155, 173, 281, 371 and 473.

Elliptic curve processor is particularly suitable for resource limited devices

because of its lower requirements of memory and computation. It is applicable to be implanted into smart cards, cellular phones and other hand-held devices so that secure communication can be provided.

Acknowledgments

The work presented in this thesis would not be possible without the help of many people. I would firstly like to acknowledge the support of my final year project supervisor and Master degree supervisor, Professor Leong Heng Wai Philip for his invaluable advice and ideas on the research. His support and expertise point me in the right direction to solve many problems in the past two years.

I thank Professor Wei Keh Wei Victor for his advice and courses which enriched my background knowledge in cryptography.

I am grateful to Professor Wong Chak Kuen, my former member of my dissertation committee, for the invaluable suggestions and advice.

I would like to thank Mr. Leong Monk Ping Norris for his help on the development of the Wildstar hardware interface and his advice. It gives a great convenience to me and makes me work efficiently.

I would also like to thank my colleagues in office 1026 who by now have become my good friends. I would like to show my appreciation to Mr. Cheung Chak Chung Ray, Mr. Sze Chin Ngai Cliff, Mr. Leong Monk Ping Norris, Miss Yuen Wing Seung, Miss Wan Po Man Polly and Mr. Wong Hiu Yung for their kind and warm-hearted help. I thank to all of them who provide a great atmosphere to work and give me an enjoyable, colorful and unforgettable life.

Finally, I would like to give special thanks to my family. I thank my mother Koo Wai Fong for her warmth and sacrifices; my father Leung Kwok Kit for his support and encouragement; my brother Leung Ka Leung for his

generosity. Last but not least, I would like to express my love to my best friend, Miss Cheung Pui King Catherine for all the time that she spent to support, encourage and understand me. To all of you thank you very much.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

The Internet is growing at an exponential rate and has currently over 400 million users compared with 45 million in 1996 [nua]. This figure is predicted to reach one billion by 2005. In order to facilitate the secure transmission of funds over the Internet, cryptography must be used. *Cryptography* is therefore a key to enable technology for the Internet and E-commerce systems. RSA is the most widely used public key cryptosystem today and the RSA algorithm is licensed by over 700 companies and has an estimated installed base of around 500 million users [rsaa]. *Elliptic curve cryptography* (ECC) is an alternative public key cryptosystem to RSA. It was proposed in the mid-1980s and has become attractive in recent years because of its lower memory requirements, lower computational requirements and higher security, making it suitable for smart cards, cellular phones or any other resource constrained applications.

Hardware cryptographic implementations give more efficient performance than software implementations. It is due to software cryptographic programs are compiled to run a sequence of instructions dedicated to the specific microprocessor. Hardware cryptographic implementations can utilize the resources

and customize the architecture to maximize the efficiency of the implementations. Field–Programmable Custom Computing Machines (FCCM) use *Field–Programmable Gate Arrays* (FPGAs) which are featured with logic components and routing resources. FPGAs can be programmed as hardware designs with different purposes and erased under the field for re-programming.

In the last decade, FPGA technology has considerably improved. A state-of-the-art Xilinx Virtex FPGA XCV3200E contains 32,448 slices which is equivalent to $4M$ system gates, and it has become possible to fit high performance cryptographic systems on a single device. Such a reconfigurable hardware is particularly suitable for cryptographic applications because of its high flexibility when compared with traditional ASIC and VLSI designs. There are several advantages in using a reconfigurable implementation:

- it is possible to implement many different cryptosystems on the same hardware platform. Therefore it can reduce the cost of development, and support future algorithms

- the turnaround time of developing a reconfigurable hardware is usually shorter. Productivity can be increased and more sophisticated algorithm can be used

- the technology of FPGAs and deep sub-micron are continuous improving. Therefore faster and larger devices are guaranteed in the future. These can improve the performance of FPGA designs with nearly no re-engineering work

- the parameters of the cryptographic algorithm can be changed at any time. For most cryptosystems, the level of security is defined by the length of the key. With reconfigurable hardware, the key size can be easily modified. For example, in an elliptic curve cryptosystem reconfigurable hardware provides the capability to use parameter dependent

designs specilized for different key sizes, curve parameters and underlying arithmetic functions.

1.2 Aims

The main aim of this work was to develop an FPGA based elliptic curve cryptographic processor using an optimal normal basis. In particular, a design with the following features was desired:

- use advantages associated with reconfigurable hardware to develop specialized processors with arbitrary key size which are fast and flexible
- use modular design to allow tradeoffs between space and speed
- provide high performance by minimizing the I/O overhead and maximizing the clock rate.

1.3 Contributions

This thesis presents an elliptic curve cryptographic processor using an optimal normal basis. The work described in this thesis has the following features which distinguishes it from all previous designs:

- the higher level curve operations as well as the field operations were implemented on the chip. This makes the I/O bandwidth requirements several orders of magnitude lower than for chips which only implement the field operations
- the curve operations are implemented as sequences of field operations which are programmed in microcode. This allows algorithmic optimizations to the design to be made without changing the hardware

- a study of the performance of using projective verses affine coordinates for implementing curve operations was made
- the entire design is generated by a module generator which can generate arbitrary key size ECC systems. Thus ECC systems of arbitrary size over an optimal normal basis can be generated (provided they fit on the FPGA device)
- a detailed profile of instruction usage during execution was made
- the parallelism of the field multiplier can also be controlled by the module generator, greatly improving performance
- the initialization of the inputs of the curve multiplication to be computed was performed using bitstream reconfiguration which results in a savings in hardware and could lead to an improvement in speed.

1.4 Thesis Outline

An introduction to cryptography is presented in Chapter 2. This is followed with an overview of the background mathematical theory for elliptic curve cryptosystems in Chapter 3.

Chapter 4 contains a description of previous work related to this research. Previous hardware and software implementations of elliptic curve cryptosystems will be discussed.

Chapter 5 provides some history and background of elliptic curve operations. The algorithms applied to the elliptic curve processor is also introduced. Moreover, security that can be realized by using elliptic curve cryptosystems is detailed.

Chapter 6 describes the tools and reconfigurable hardware that was used, including the tools that developed to facilitate the design. In addition, the

implementation of elliptic curve processor is introduced and the architecture of the design is discussed.

In Chapter 7, results are presented and conclusion and some recommendations for future research in this area are given in Chapter 8.

Chapter 2

Cryptography

2.1 Introduction

In this chapter, a brief introduction to cryptography is given. It will cover the important concepts of secret key and public key cryptosystems which are the two main classes of cryptosystems.

This chapter begins with an introduction to cryptography. It is followed by discussing two kinds of cryptosystems, secret key and public key cryptosystems. The details about public key cryptosystems are given next. The major components, one–way hashing functions, certification authorities, the discrete logarithm problem, key exchange protocols and digital signatures are discussed. Finally, a brief comparison of RSA and ECC is given.

2.2 Foundations

Cryptography is a technique used for keeping a message secret during transmission through an untrusted and insecure channel. During encryption, the original message (called the *plaintext*) is encoded into an encrypted message (called the *ciphertext*). Similarly, the process that retrieves the plaintext from the ciphertext is called decryption. Encryption and decryption are usually associated with a *key*. In a properly designed cryptosystem, the decryption

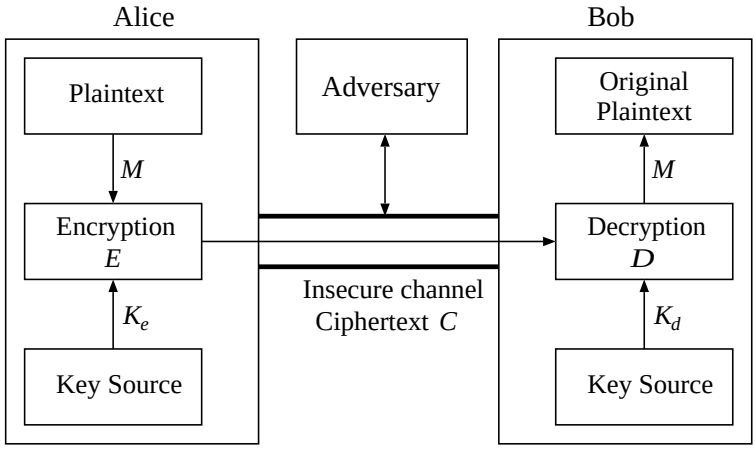


Figure 2.1: Overview of cryptographic transmission process.

process can be performed successfully only by knowing the right key. The level of the security of a cryptographic system depends on the algorithm as well as the key size. Longer keys are less likely to be guessed and for each additional bit in a key, the time for an exhaustive search is doubled. An overview of a cryptographic transmission process is shown in Figure 2.1

The transmission process can be formulated as follow.

$$\begin{aligned} E_{K_e}(M) &= C \\ D_{K_d}(C) &= M \\ D_{K_d}(E_{K_e}(M)) &= M \end{aligned}$$

where M is the plaintext, C is the ciphertext, K_e is the encryption key and K_d is the decryption key. $E_K(Z)$ and $D_K(Z)$ are the encryption and decryption of Z with key K respectively.

The aim of the encryption is to make it intractable for an adversary who is eavesdropping on the insecure channel to deduce the secret message, or in other words, make it impossible to derive the plaintext from ciphertext without the decryption key K_d.

Cryptographic algorithms can be divided into two types, namely symmetric algorithms and asymmetric algorithms (also known as secret key algorithms

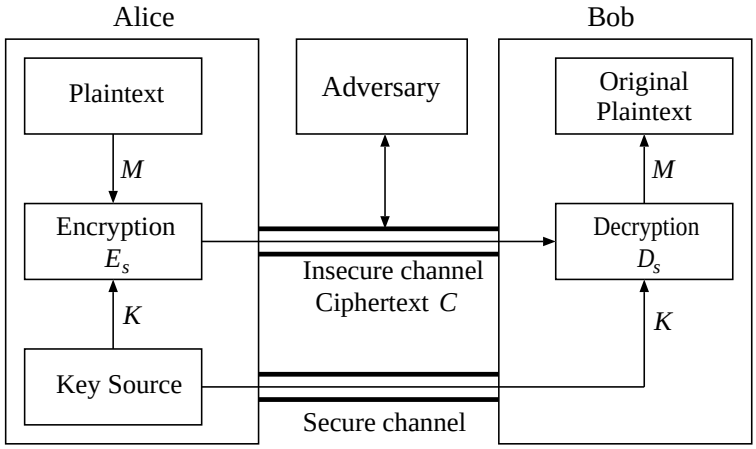


Figure 2.2: Secret key cryptosystem.

and public key algorithms respectively). Symmetric algorithms make use of a single key for both encryption and decryption processes ($K_e = K_d$) while asymmetric algorithms have different keys for encryption and decryption (in general $K_e \neq K_d$).

2.3 Secret Key Cryptosystems

In a secret key cryptosystem, the encryption and decryption key are the same ($K_e = K_d$ in Figure 2.1). The decryption process is just the reverse of encryption process with the same key. An example of secret key cryptosystems is data encryption standard (DES) [MOV99] which has been widely used for over 20 years. This is illustrated in Figure 2.2.

Secret key cryptosystems can be summarized by the following formulae where E_s and D_s are the encryption and decryption processes of the secret key cryptosystem respectively.

$$\begin{aligned} E_{s,K}(M) &= C \\ D_{s,K}(C) &= M \\ D_{s,K}(E_{s,K}(M)) &= M \end{aligned}$$

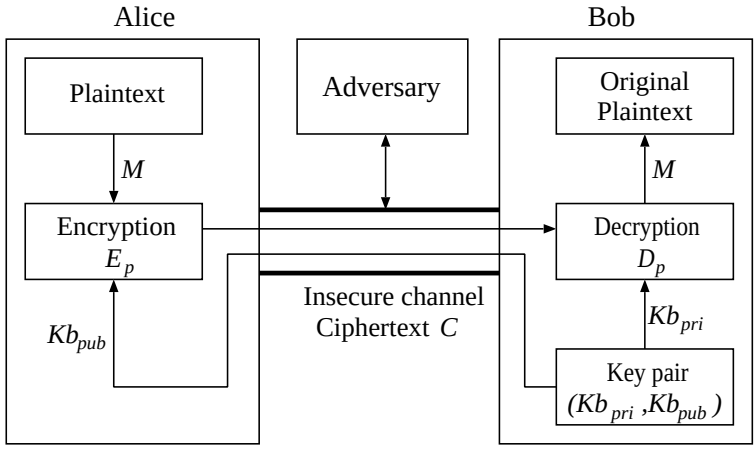


Figure 2.3: Public key cryptosystem.

2.4 Public Key Cryptosystems

By careful design of cryptographic algorithm, secret key cryptosystems can be very secure. However, when a secret key cryptosystem is used, the key distribution problem must be addressed. The receiver's keys must be distributed in secret and anyone with the key will know the plaintext also. A key could be distributed through a trusted courier, a phone system or some other secure channel.

The generation, transmission and storage of keys is called *key management*. It is often difficult to provide secure key management, especially in open and diverse systems with a larger number of users, for instance, the Internet. In order to solve the key distribution problem, public key cryptosystems were introduced by Whitfield Diffie and Martin Hellman [DH76] in 1976 with different keys for encryption and decryption.

Figure 2.3 shows how a public key cryptosystem works. Since encryption keys and decryption keys are different, the keys are generated in pairs, one which is made public and the other kept private. Suppose Alice and Bob have their own key pairs. The public keys are denoted with subscript "pub" while subscript "pri" is used to represent the private keys of each party. Here

E_p and D_p are the encryption and decryption algorithms of the public key cryptosystem. If Alice wants to send a secret message to Bob, she will use the public key of Bob (Kb_{pub}) to encrypt the message and then send the ciphertext to Bob. When Bob receives the ciphertext, he can decrypt the ciphertext by using his private key (Kb_{pri}) to get back the original plaintext.

An adversary who only knows Kb_{pub} and C but not Kb_{pri}, cannot decrypt the ciphertext since the cryptosystem is designed to make that process intractable.

2.4.1 One-way Function

Public key cryptosystems are based on a problem that is *difficult* to solve. "Difficult" in this case refers to the computational requirements in finding a solution. These hard problems are known as a one-way function in the terminology of cryptography.

A one-way function is a mathematical function that is significantly easier to compute in one direction than in the reverse direction. It might be possible, for example, to compute the function in the forward direction in a second but to compute its inverse could take several million years. A one-way function for a public key cryptosystems is computational infeasible in the inverse direction unless one has additional information (known as "trap door"), which makes the inverse computation easy. The trap door, in fact, is the private key of the cryptosystem.

2.4.2 Certification Authority

Although public key cryptography solves the key management problem, it does not address the problem of someone using fake key to pretend to be someone else. To solve this problem, certificates are used. Certificates are digital documents used to bind a public key to an individual. They allow

verification of the claim that a specific public key does, in fact, belong to a specific individual.

Certificates are issued by a certification authority (CA), a trusted central administration. A company may issue certificates to its employees, or a university to its students. If a person wants to get someone's public key in order to send message to them, they can check the certificate of the key. If he believes that the certificate was issued by the CA, they can trust the public key belongs to that person.

The most widely accepted format for certificates is defined by the ITU-*T* *X*.509 [CCI88]. A detailed discussion of certificate formats can be found in [Ken93].

2.4.3 Discrete Logarithm Problem

The discrete logarithm problem (DLP) is a one-way function that based on the difficulty of finding a logarithm in a group (see Section 3.2 for a reviews of group theory). The DLP has been extensively studied and has been the basis of several public key cryptosystems. It is defined as follows.

Given elements g and y in a group G, (an introduction to group theory is given in Chapter 3), the discrete logarithm problem involves finding a non–negative integer x such that

$$g^x = y.$$

Some efficient algorithms for the discrete logarithm problem have been reported. They include the Pollard's rho algorithm of logarithms [Pol74], the Pohlig–Hellman algorithm [PH78] and the index-calculus methods [HR82]. The index-calculus algorithm is the most efficient method known for computing discrete logarithms. It is a sub-exponential time algorithm for some groups [MOV99] but not elliptic curves.

2.4.4 RSA vs. ECC

The most commonly used public key cryptosystem is RSA. It was developed by Ron Rivest, Adi Shamir and Leonard Adleman in 1977 [RSA78]. It is an algorithm that depends on a one-way function involving the factorization of a large integer n. Here n is a product of two large prime numbers, p and q. The one-way function of RSA is to calculate n relatively easily if p and q are known. However, it is intractable to compute p and q if only know n (this problem is the problem of factorization n).

Elliptic curve cryptography is another public key cryptosystem. Both RSA and ECC can provide secure communications, however, ECC has advantages over RSA or even other commonly used public key cryptosystems. RSA is based on the integer factorization problem while ECC is based on the discrete logarithm problem [Odl84]. ECC has the following advantages over RSA:

- ECC always has a shorter key length than any known public key cryptosystems with similar strength of security. Strength of security is said to be in term of the time to break the cryptosystem

- ECC is probably more secure than RSA, the largest RSA and ECC challenges solved being 512-bit and 108-bit respectively. The solution to 108-bit ECC challenge is believed to be the largest effort ever expended in a public-key cryptography challenge. It took four months and involved approximately 9,500 machines. The amount of work required to solve the problem was about 50 times of the 512-bit RSA [cer].

Figure 2.4 shows how long should it take to break the RSA and ECC cryptosystems of different key length [cer]. The hard problem of RSA is factorization of a large integer while solving the discrete logarithm problem is needed to break ECC. For the same security level, the key size of ECC is much shorter than RSA's (Figure 2.4). In other words, ECC provides a more secure

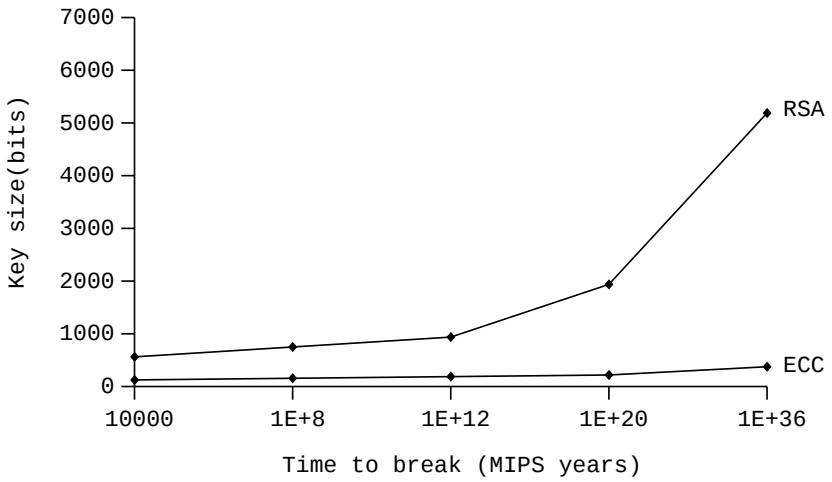


Figure 2.4: Time to break the keys of ECC verses RSA.

cryptosystem for the same key length as RSA. In practice, 160 to 192–bit ECC keys are used nowadays. However, the acceptably secure key size depends upon applications. Current commercial systems which use 1024-bit RSA for commercial transactions and 2048-bit RSA for high security applications [rsab]. Then roughly correspond to 134-bit and 173-bit ECC respectively [LV00].

2.4.5 Key Exchange Protocol

Public key cryptosystems can be used to securely transmit a common key between 2 parties. The protocol is shown in Figure 2.5. First, Alice generates a random key K. After that she uses Bob's public key to encrypt K and then sends the ciphertext to Bob. Bob can retrieve the key K by decrypting the ciphertext with his own private key. By using this method, Alice and Bob can compromise on a common key K. Afterwards, they can communicate with one another by using a secret key cryptosystem with the key K.

Diffie–Hellman Key Exchange

The Diffie–Hellman key exchange protocol was developed in 1976 [DH76] and it allows two parties to exchange a secret key over an insecure channel. The

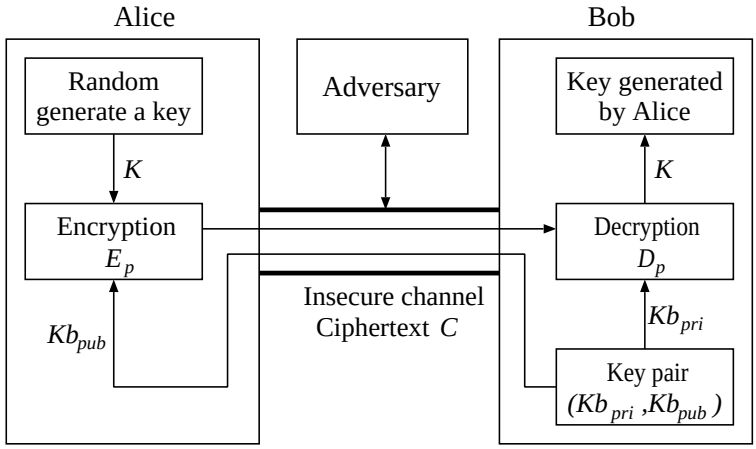


Figure 2.5: Key exchange protocol.

protocol is as follows.

Suppose Alice and Bob want to agree on a shared secret key k. First of all, there are public parameters p and $g < p$ where p is prime. In addition, there exists k such that $g^k = n \mod p$ for every $n \in \{1, \ldots, p-1\}$ (modular arithmetic is discussed in Chapter 3):

1. Alice generates a random private value $a \in \{1, \ldots, p-2\}$, calculates $g^a \mod p$ and sends it to Bob

2. Bob generates a random private value $b \in \{1, \ldots, p-1\}$, calculates $g^b \mod p$ and sends it to Alice

3. Alice computes $g^{ab} = \left(g^b\right)^a \mod p$ while Bob computes $g^{ba} = \left(g^a\right)^b \mod p$

Since $g^{ab} = g^{ba} = k$, Alice and Bob now have a shared secret key k.

An adversary who eavesdrops on the channel can capture p, g, g^a and g^b but not k. To deduce k, they would have to solve the discrete logarithm problem.

2.4.6 Digital Signature

One important application of public key cryptography is digital signatures. A digital signature of a document is a piece of information based on both the

document and the signer's private key. The functions of digital signatures are similar to traditional hand-written signatures. They have followings similarities:

- Authentication: a digital signature guarantees that the signer is the originator of the item
- Unforgeable: a signature is a proof that the signer deliberately signed the document
- Not reusable: a digital signature cannot be moved to another item
- Unalterable: after the item is signed, it cannot be altered
- Cannot be repudiated: the signer cannot claim that he didn't sign it later

For a person to sign a document, they can encrypt the document with their private key. The signed document (ciphertext produced by encrypting the document) is then sent together with the original document to the receiver. If others want to verify the signature of the signer, they can use the signer's public key to decrypt the signed document and compare the decrypted document with the original document. If they are the same, the document must have been signed by the particular signer. On the other hand, the signer cannot deny that they signed the document in case they are only person that can produce a digital signature for that document.

In order to improve the efficiency of the authentication process for digital signature, signers usually do not sign the message directly. A hashing function is applied to the original message and produced a string, the *message digest* (MD). The message digest is usually considerably shorter than the original message. The signers can sign the short message digest instead of the long message making the verification process faster. However, the hashing function

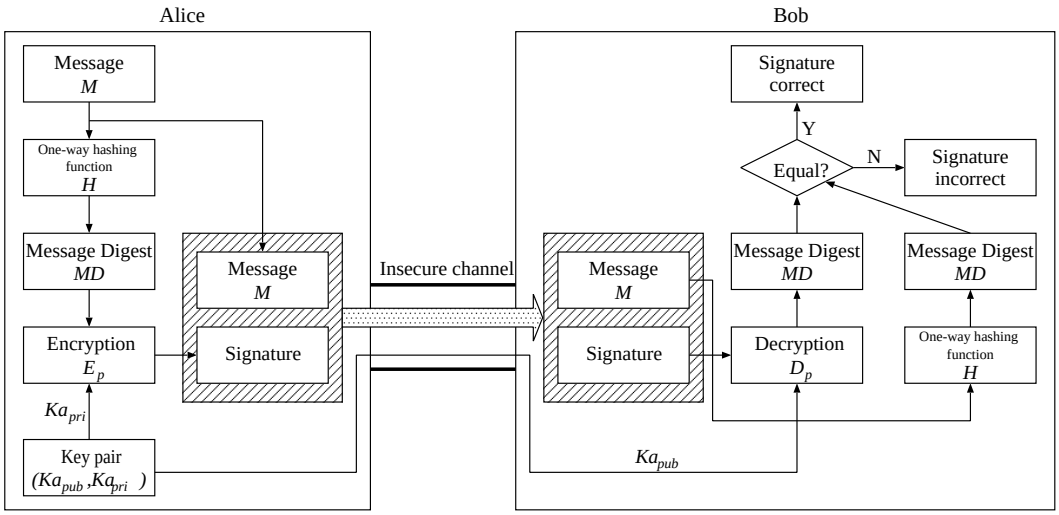


Figure 2.6: Digital signature with message digest.

must be designed carefully so that it is hard to find two messages that are hashed to the same message digest.

The digital signature scheme shown in Figure 2.6 has the problem that a plaintext message is sent through an insecure channel. Figure 2.7 shows the digital signature process with message digest and encryption. In this case, Alice encrypts the message before it is sent to Bob and only encrypted messages are sent.

2.5 Secret Key vs. Public Key Cryptography

Besides the key management problem, secret key and public key cryptosystems have different computational requirements. Slow cryptosystems can be a bottleneck for a computer system. Therefore the speed and the efficiency of a cryptosystem are important considerations.

A software RSA implementation, RSA Data Security's cryptographic toolkit BSAFE 3.0 [rsaa], has a throughput for public key operations of 21.6 *k*bits per second with 512-bit key length and 7.4 *k*bits per second with a 1024-bit

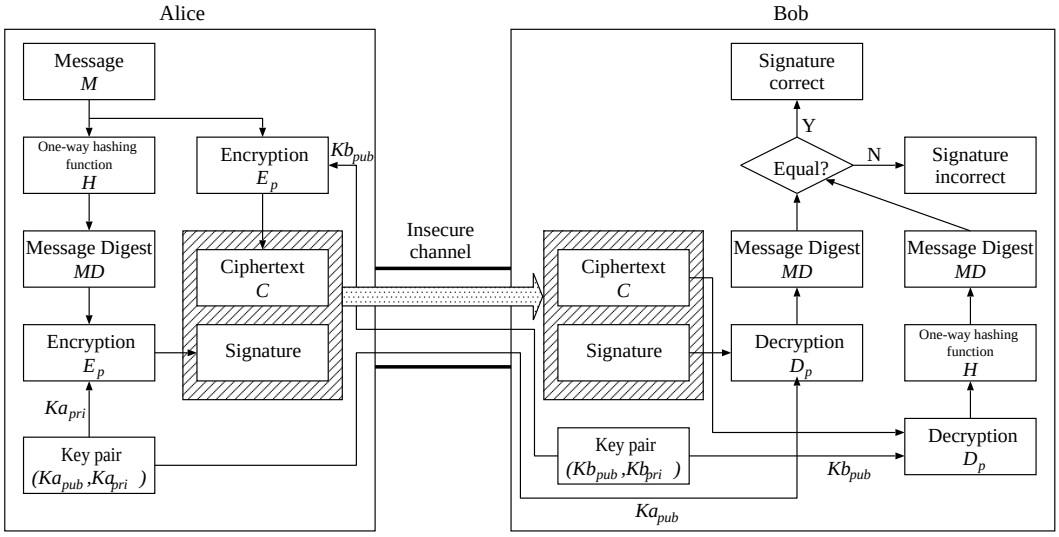


Figure 2.7: Digital signature with message digest and encryption.

key length on a 90 MHz Pentium machine. The fastest reported RSA hardware [SV93] has a throughput greater than 600 $k$bits per second with a 512-bit key length and 165 $k$bits per second with key size of 1024 bits.

DES is much faster than RSA. In software implementation, a DES cryptographic engine had been reported in [Bih97]. Its speed is 137 Mbits per second on a 300 MHz Alpha 8400 processor. In hardware implementation, DES is also much faster than RSA. In [Pat00], an implementation of DES on an FPGA is reported with a throughput of over 10 Gbits per second with clock rate of 168 MHz. Therefore, in order to transmit a large amount of information, secret key cryptosystems are preferable.

Public key cryptosystems solve the key management problem while secret key cryptosystems have a higher throughput rate. In practice, a hybrid method is used. Figure 2.8 illustrates a practical system. Before the actual transmission of the data, Alice and Bob first agree on a common, session key K_s, using a public key cryptosystems. Since the key size is usually much shorter than the plaintext, the key exchange process will not take a long time. After that, the transmission of secret information is done using a secret key cryptosystem.

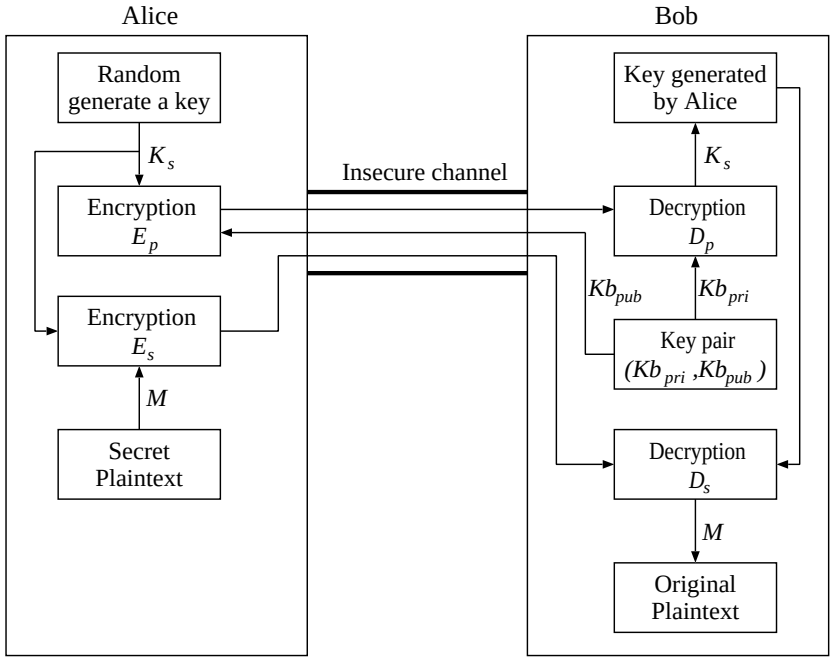


Figure 2.8: Practical cryptography.

2.6 Summary

In this chapter, an introduction to secret key and public key cryptosystems was presented. The algorithms and the applications of public key cryptosystems were detailed and a comparison between RSA and ECC was presented.

Secret key cryptosystems are efficient but there is a problem of key distribution. Since public key cryptosystems solve the problem of key distribution, a hybrid cryptosystems is preferred so that high speed and secure communications can be provided.

The key exchange protocol and digital signatures are the two main applications of public key cryptosystems. Key exchange is used to negotiate a session key between the sender and receiver. Digital signatures can be implemented using public key algorithm to perform a digital equivalent of a personal signature.

Chapter 3

Mathematical Background

3.1 Introduction

Elliptic curve cryptosystems use underlying operations which are based on abstract algebra and in particular finite fields. In this chapter, an overview of the abstract algebra required to understand the elliptic curve processor is presented. Finite fields have applications in coding, error correction and an cryptography [Kob87a, LN94].

This chapter begins with introduction to groups, rings and fields. The two representations of elements (polynomial and normal basis) are then presented. Finally, operations in an optimal normal basis are discussed in detail.

3.2 Groups and Fields

A binary operation $+$ on set S is a mapping from $S \times S$ to S. That is, $+$ is a rule which assigns to each ordered pair of elements from S.

A *group* $(G, +)$ consists of a set of numbers G together with a binary operation $+$ on G which satisfies the following three axioms:

- Closure: for all $a, b \in G$, $a + b \in G$
- Associativity: the group operation is associative. That is, $(a + b) + c = a + (b + c)$ for all $a, b, c \in G$

- Identity: there is an identity element $0 \in G$ such that $a + 0 = 0 + a$ for all $a \in G$
- Inverse: for each $a \in G$ there exists an element $-a \in G$, called the inverse of a, such that $-a + a = a + (-a) = 0$.

The group G is said to be *abelian* (or commutative) if

- $a + b = b + a$ for all $a, b \in G$

Note that addition group notation has been used for the group operation. If the group operation is multiplication, then the group is said to be a multiplicative group with identity element is denoted by 1 and inverse of a is a^{-1}.

The notation $\#G$ will be used to denote the number of elements in a group G. A group is said to be finite if $\#G$ is finite. The number of elements in a finite group is called its order.

A *ring* $(R, +, \times)$ consists of a set R with two binary operations arbitrarily denoted $+$ (addition) and $\times$ (multiplication) on R, satisfying the following axioms:

- $(R, +)$ is an abelian group with identity element 0
- The operation $\times$ is associative. That is, $(a \times b) \times c = a \times (b \times c)$ for all $a, b, c \in R$
- There exists an identity $1 \in R$ with $1 \neq 0$ such that $1 \times a = a \times 1 = a$ for $a \in R$
- The operation $\times$ is distributive over $+$ i.e. $a \times (b + c) = (a \times b) + (a \times c)$ and $(b + c) \times a = (b \times a) + (c \times a)$ for all $a, b, c \in R$

The ring is a commutative ring if $a \times b = b \times a$ for all $a, b \in R$

A *field* is a commutative ring in which all non-zero elements have multiplicative inverses, i.e. for every $a \neq 0, a \in F$ there exists an element $a^{-1} \in F$ such that $a^{-1} \times a = a \times a^{-1} = 1$.

3.3 Finite Fields

A *finite field* is a field F which contains a finite number of elements. The order of F is the number of elements in F ($\#F$).

Let F_p be the finite field with p elements. If p is a prime number, then it is called a prime field. Numbers in the field F_2 can be represented by $\{0, 1\}$. If $p = 2^n$ (called binary finite field or Galois Fields), numbers in F_{2^n} can be represented as n-bit binary numbers. The field F_{2^n} (or $GF(2^n)$) has particular importance in cryptography due to its binary nature. It leads to particularly efficient hardware implementations. Elements of the field are represented in terms of a basis. Most implementations either use a polynomial basis or a normal basis. Finite field F_p is of characteristic p while F_{q^n} is of characteristic q.

3.4 Modular Arithmetic

If a, b and n are elements in a field F, then a is said to be congruent to b modulo n if there exists $k \in F$ such that $a = nk + b$. It is denoted by

$$a \equiv b \mod n$$

in which b is called the remainder of a divided by n.

The multiplicative inverse of a modulo n is an $x \in F$ such that $ax \equiv 1 \mod n$. If x exists, then it is unique and inverse of a is denoted by a^{-1}.

3.5 Polynomial Basis

A *polynomial* is a mathematical expression consisting of a sum of terms, each term including a variable or variable raised to a power and multiplied by a coefficient. The simplest polynomials have one variable, shown as follows

$$a_n x^n + a_{n-1} x^{n-1} + \cdots + a_2 x^2 + a_1 x^1 + a_0 x^0$$

where a_i's represent the coefficients and x represents the variable which is a complex number. An example of a polynomial is

$$x^5 + 2x^4 + 7x^2 + 9x + 2.$$

Polynomial basis arithmetic operates the polynomials in a field of order p according to the same powers of the variable x with the coefficients are modulo p. Suppose another polynomial

$$x^3 + 5x^2 + 3x + 9$$

in F_{10} is added to the above example.

$$\begin{aligned} & (x^5 + 2x^4 + 7x^2 + 9x + 2) + (x^3 + 5x^2 + 3x + 9) \\ = \quad & x^5 + 2x^4 + x^3 + 2x^2 + 2x + 1. \end{aligned}$$

For polynomial basis arithmetic, if the coefficients are modulo a prime number, it is said to be a polynomial basis over a finite field. However normal basis arithmetic (described in the next subsection) is used in the work because it leads to more efficient hardware implementations.

3.6 Optimal Normal Basis

A *normal basis* is more suitable for hardware implementation than polynomial basis because operations are mainly comprised of rotation, shifting and exclusive-OR operations which are very efficient in hardware. Moreover, there is an optimal normal basis which is a special case of normal basis with minimum complexity. Therefore for the implementation described in this thesis, a normal basis was chosen. It will be discussed in this section.

Suppose β is an element in the field F_{p^m}, the polynomial representation is

$$\beta = a_n x^n + \cdots + a_1 x + a_0$$

where $n < m$.

A normal basis can be formed using the set

$$\{\beta^{p^{m-1}}, \ldots, \beta^{p^2}, \beta^p, \beta\}.$$

A finite field of characteristic 2 (i.e. $p = 2$) is chosen because of its binary nature. Every element A in the field F_{2^n} can be uniquely represented in the form

$$A = \sum_{i=0}^{n-1} a_i \beta^{2^i}$$

where $a_i \in F_2$ and $\beta \in F_{2^n}$.

There are several operations among the elements over F_{2^n}. They are addition, squaring (element multiplies itself), multiplication and inversion (sequences of addition and multiplication) which are discussed below.

3.6.1 Addition

Suppose A, B are elements in the field F_{2^n}

$$A = \sum_{i=0}^{n-1} a_i \beta^{2^i},$$
$$B = \sum_{j=0}^{n-1} b_j \beta^{2^j}.$$

Addition is defined by

$$A + B = \left(\sum_{i=0}^{n-1} a_i \beta^{2^i} \right) + \left(\sum_{j=0}^{n-1} b_j \beta^{2^j} \right). \tag{3.1}$$

Since every power of β is linearly independent, Equation 3.1 can be written as

$$A + B = \sum_{i=0}^{n-1} (a_i + b_i) \beta^{2^i}$$

where a_i, b_i are added modulo 2. Note that there is no carry in finite field arithmetics. So the addition can be implemented as a bit-wise exclusive-OR (XOR) operation.

3.6.2 Squaring

Since every power of β is linearly independent, then

$$\begin{aligned} A^2 &= \left(\sum_{i=0}^{n-1} a_i \beta^{2^i}\right)^2 \\ &= \sum_{i=0}^{n-1} a_i \left(\beta^{2^i}\right)^2 \\ &= \sum_{i=0}^{n-1} a_i \beta^{2^{i+1}} \\ &= \sum_{i=0}^{n-1} a_{i-1} \beta^{2^i}. \end{aligned}$$

Moreover,

$$\left(\beta^{2^{n-1}}\right)^2 = \beta^{2^n} = \beta.$$

Therefore squaring an element over F_{2^n} involves shifting each coefficient up to the next term and rotating the most significant coefficient down to the least significant position that is, rotate left operation.

3.6.3 Multiplication

Multiplication over field F_{2^n} is defined as follows. Let

$$\begin{aligned} A &= \sum_{i=0}^{n-1} a_i \beta^{2^i}, \\ B &= \sum_{i=0}^{n-1} b_i \beta^{2^i} \end{aligned}$$

and

$$C = A \times B = \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} a_i b_j \beta^{2^i} \beta^{2^j} = \sum_{i=0}^{n-1} c_i \beta^{2^i}$$

therefore

$$\beta^{2^i} \beta^{2^j} = \sum_{k=0}^{n-1} \lambda_{ijk} \beta^{2^k} \tag{3.2}$$

where $\lambda_{ijk} \in \{0, 1\}$. Then multiplication can be written to be

$$c_k = \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} \lambda_{ijk} a_i b_j \tag{3.3}$$

where $0 \leq k \leq n-1$. By raising both sides of Equation 3.2 to the power of 2^{-l}, then

$$(\beta^{2^i} \beta^{2^j})^{2^{-l}} = \beta^{2^{i-l}} \beta^{2^{j-l}} = \sum_{k=0}^{n-1} \lambda_{i-l,j-l,k} \beta^{2^k} = \sum_{k=0}^{n-1} \lambda_{ijk} \beta^{2^{k-l}}. \tag{3.4}$$

Equating the coefficients of β^{2^0} in the above equation, yields

$$\lambda_{ijl} = \lambda_{i-l,j-l,0} \quad \text{for all } 0 \leq i, j, l \leq n-1. \tag{3.5}$$

Therefore Equation 3.3 can be written as

$$c_k = \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} \lambda_{i-k,j-k,0} a_i b_j = \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} \lambda_{ij0} a_{i+k} b_{j+k}. \tag{3.6}$$

An *optimal normal basis* (ONB) [MOVW89] is one with the minimum number of nonzero terms in Equation 3.3, or equivalently, the minimum possible number of nonzero terms in λ_{ij} for a specific k. This value was proved to be $2n-1$ [ABV89] and since it allows multiplication with minimum complexity, such a basis normally leads to a more efficient hardware implementation.

Derivation of the λ_{ijk} values in Equation 3.3 is dependent on n. There are two types of optimal normal bases [MOVW89], Type I and Type II. For a Type I ONB, there exists an optimal normal basis in F_{2^n} if

1. $n+1$ is a prime;

2. 2 is a primitive in F_{n+1}.

Rule 2 means 2 raised to any power in the range 0 to n modulo $n+1$ must result in a unique integer in the range 1 to n.

On the other hand, for a Type II ONB, an optimal normal basis exists in F_{2^n} if

2	3	4	5	6	9	10	11	12	14	18	23
26	28	29	30	33	35	36	39	41	50	51	52
53	58	60	65	66	69	74	81	82	83	86	89
90	95	98	99	100	105	106	113	119	130	131	134
135	138	146	148	155	158	162	172	173	174	178	179
180	183	186	189	191	194	196	209	210	221	226	230
231	233	239	243	245	251	254	261	268	270	273	278
281	292	293	299	303	306	309	316	323	326	329	330
338	346	348	350	354	359	371	372	375	378	386	388
393	398	410	411	413	414	418	419	420	426	429	431
438	441	442	443	453	460	466	470	473	483	490	491
495	508	509	515	519	522	530	531	540	543	545	546
554	556	558	561	562	575	585	586	593	606	611	612
614	615	618	629	638	639	641	645	650	651	652	653
658	659	660	676	683	686	690	700	708	713	719	723
725	726	741	743	746	749	755	756	761	765	771	772
774	779	783	785	786	791	796	803	809	810	818	820
826	828	831	833	834	846	852	858	866	870	873	876
879	882	891	893	906	911	923	930	933	935	938	939
940	946	950	953	965	974	975	986	989	993	998	

Table 3.1: Values of $n \leq 1000$ with an optimal normal basis.

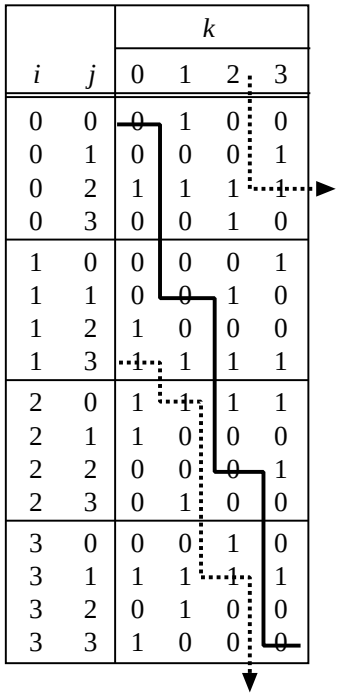
1. $2n + 1$ is prime, and either

2a. 2 is a primitive in F_{2n+1}, or

2b. $2n + 1 \equiv 3(\mod 4)$ and 2 generates the quadratic residues in F_{2n+1}.

Similarly, rule 2a means that every 2^k modulo $2n+1$, in the range 1 to $2n$ ($0 \leq k \leq 2n-1$). Therefore 2 is called the generator for all the possible locations in the $2n + 1$ field. Rule 2b means that even if $2^k \mod 2n + 1$ does not generate every element in the range 1 to $2n$, however, half of points in the field of form by rule 2a can be hit. It is because $\sqrt{2^k} \mod 2n+1$ can be taken. The points generated by rule 2b are in the form of perfect squares. [Rie87, Ros98a]

An ONB exists in F_{2^n} for 23% of all possible values of $n \leq 1000$ [MOVW89]. Table 3.1 lists these values.



		k			
i	*j*	0	1	2	3
0	0	0	1	0	0
0	1	0	0	0	1
0	2	1	1	1	1
0	3	0	0	1	0
1	0	0	0	0	1
1	1	0	0	1	0
1	2	1	0	0	0
1	3	1	1	1	1
2	0	1	1	1	1
2	1	1	0	0	0
2	2	0	0	0	1
2	3	0	1	0	0
3	0	0	0	1	0
3	1	1	1	1	1
3	2	0	1	0	0
3	3	1	0	0	0

Table 3.2: Multiplication table, λ_{ijk}, of Type I ONB in $GF(2^4)$.

The multiplication table λ_{ijk} can be constructed by a k-fold cyclic shift to λ_{ij0} [MOVW89] using Equation 3.5. However, λ_{ij0} is derived differently for the two different types of ONB described above. For the Type I ONB, $\lambda_{ij0} = 1$ iff i and j satisfy one of the two congruences [Ros98a],

$$\left.\begin{aligned} 2^i + 2^j &\equiv 1 \mod n+1, \\ 2^i + 2^j &\equiv 0 \mod n+1. \end{aligned}\right\} \tag{3.7}$$

The multiplication table of Type I ONB in $GF(2^4)$ is shown in Table 3.2. λ_{ij0} of the table are built according to the two congruences 3.7. For example, $\lambda_{120} = 1$ since $2^1 + 2^2 \equiv 1 \mod 5$. Due to the fact that the two congruences are symmetric in i and j, if λ_{ij0} equals to 1, then λ_{ji0} equals to 1.

Table 3.2 shows how to construct λ_{ijk} for k in the range 1 to $n-1$ from λ_{ij0} (Equation 3.5). For example, the solid line in Table 3.2 shows that

$$\lambda_{000} = \lambda_{111} = \lambda_{222} = \lambda_{333}.$$

The subscripts of λ_{ijk} are modulo n therefore, for instance,

$$\lambda_{130} = \lambda_{201} = \lambda_{312} = \lambda_{023}$$

shown by the dotted line.

For a Type II ONB, $\lambda_{ijk} = 1$ iff i and j satisfy one of the four congruences [Ros98a],

$$\left.\begin{array}{rcl} 2^i + 2^j & \equiv & 2^k \mod 2n+1, \\ 2^i + 2^j & \equiv & -2^k \mod 2n+1, \\ 2^i - 2^j & \equiv & 2^k \mod 2n+1, \\ 2^i - 2^j & \equiv & -2^k \mod 2n+1. \end{array}\right\} \tag{3.8}$$

Therefore, $\lambda_{ij0} = 1$ iff i and j satisfy one of the four congruences $2^i \pm 2^j \equiv \pm 1 \mod 2n+1$ [MOVW89].

Table 3.3 is the multiplication table of Type I ONB in $GF(2^5)$. It is constructed according to the Equations 3.8.

The lines in Table 3.3 show the cyclic shift relations on the multiplication tables. The solid and dotted lines represent

$$\lambda_{000} = \lambda_{111} = \lambda_{222} = \lambda_{333} = \lambda_{444}$$

and

$$\lambda_{440} = \lambda_{001} = \lambda_{112} = \lambda_{223} = \lambda_{334}$$

respectively. Note that $GF(2^4)$ and $GF(2^5)$ are ONB so that the number of '1's in every column of the table is less than or equal to $2n-1$ [MOVW89].

		k				
i	*j*	0	1	2	3	4
0	0	0	1	0	0	0
0	1	1	0	0	1	0
0	2	0	0	0	1	1
0	3	0	1	1	0	0
0	4	0	0	1	0	1
1	0	1	0	0	1	0
1	1	0	0	1	0	0
1	2	0	1	0	0	1
1	3	1	0	0	0	1
1	4	0	0	1	1	0
2	0	0	0	0	1	1
2	1	0	1	0	0	1
2	2	0	0	0	1	0
2	3	1	0	1	0	0
2	4	1	1	0	0	0
3	0	0	1	1	0	0
3	1	1	0	0	0	1
3	2	1	0	1	0	0
3	3	0	0	0	0	1
3	4	0	1	0	1	0
4	0	0	0	1	0	1
4	1	0	0	1	1	0
4	2	1	1	0	0	0
4	3	0	1	0	1	0
4	4	1	0	0	0	0

Table 3.3: Multiplication table, λ_{ijk}, of Type II ONB in $GF(2^5)$.

Using Type II ONB as an example, Equation 3.6 can be expressed as,

$$\begin{aligned} c_k = & \overbrace{a_k b_{1+k}}^{\lambda_{010}} + \overbrace{a_{1+k} b_k}^{\lambda_{100}} + \overbrace{a_{1+k} b_{3+k}}^{\lambda_{130}} + \overbrace{a_{2+k} b_{3+k}}^{\lambda_{230}} + \overbrace{a_{2+k} b_{4+k}}^{\lambda_{240}} \\ & + \underbrace{a_{3+k} b_{1+k}}_{\lambda_{310}} + \underbrace{a_{3+k} b_{2+k}}_{\lambda_{320}} + \underbrace{a_{4+k} b_{2+k}}_{\lambda_{420}} + \underbrace{a_{4+k} b_{4+k}}_{\lambda_{440}} \end{aligned} \tag{3.9}$$

where $0 \leq k \leq 4$ and all subscripts are modulo 5. The corresponding λ of each product term of a and b are highlighted. Therefore c_k can be calculated by the appropriate shifting of a and b and the multiplication can be computed in n cycles.

3.6.4 Inversion

Inversion of a is denoted by a^{-1} and is defined as below

$$aa^{-1} \equiv 1 \mod n$$

where a and n are elements in field F. The algorithm used for inversion is derived from Fermat's Theorem [Men93]

$$a^{-1} = a^{2^n-2} = (a^{2^{n-1}-1})^2 \tag{3.10}$$

for all $a \neq 0$ in F_{2^n}. The method used was proposed by Itoh and Tsujii [IT88], based on the following decomposition which minimizes the number of multiplications (squarings are much cheaper in a normal basis). If n is odd, then

$$2^{n-1} - 1 = \left(2^{\frac{n-1}{2}} - 1\right)\left(2^{\frac{n-1}{2}} + 1\right)$$

and

$$a^{2^{n-1}-1} = \left(a^{2^{\frac{n-1}{2}}-1}\right)^{2^{\frac{n-1}{2}}+1} \tag{3.11}$$

and can be computed using one field multiplication provided $a^{2^{\frac{n-1}{2}}-1}$ is given. The cost of squaring is ignored because it is insignificant compared with multiplication. On the other hand, if n is even, then

$$2^{n-1} - 1 = 2\left(2^{n-2} - 1\right) + 1 = 2\left(2^{\frac{n-2}{2}} - 1\right)\left(2^{\frac{n-2}{2}} + 1\right) + 1$$

therefore

$$a^{2^{n-1}-1} = a^{2\left(2^{\frac{n-2}{2}}-1\right)\left(2^{\frac{n-2}{2}}+1\right)+1} \tag{3.12}$$

which takes two field multiplications if $a^{2^{\frac{n-2}{2}}-1}$ is given. By decomposing the power of a in Equation 3.11 and 3.12, an inversion can be expressed as a series of squarings and multiplications.

An example showing the recursive decomposition of an inversion to multiplications is now presented. Consider the field $F_{2^{173}}$ and element $a \in F_{2^{173}}$, $a \neq 0$. Recall Equation 3.10,

$$a^{-1} = a^{2^n - 2}.$$

For clearer illustration, only the exponent part is shown. So

$$\begin{aligned}
\text{(Iteration 6)} \quad 2^{173} - 2 &= 2\,(2^{86} - 1)\,(2^{86} + 1) \\
\text{(Iteration 5)} \quad 2^{86} - 1 &= (2^{43} - 1)\,(2^{43} + 1) \\
\text{(Iteration 4)} \quad 2^{43} - 1 &= 2\,(2^{21} - 1)\,(2^{21} + 1) + 1 \\
\text{(Iteration 3)} \quad 2^{21} - 1 &= 2\,(2^{10} - 1)\,(2^{10} + 1) + 1 \\
\text{(Iteration 2)} \quad 2^{10} - 1 &= (2^5 - 1)\,(2^5 + 1) \\
\text{(Iteration 1)} \quad 2^5 - 1 &= 2\,(2^2 - 1)\,(2^2 + 1) + 1 \\
\text{(Iteration 0)} \quad 2^2 - 1 &= (2 - 1)\,(2 + 1) = (2 + 1)\,.
\end{aligned}$$

and so an inversion over $F_{2^{173}}$ takes 10 field multiplications.

Iteration	s	r	decimal of r	Calculation
0	6	10	2	$2^2 - 1 = (2 - 1)\,(2 + 1)$
1	5	101	5	$2^5 - 1 = 2\,(2^2 - 1)\,(2^2 + 1) + 1$
2	4	1010	10	$2^{10} - 1 = (2^5 - 1)\,(2^5 + 1)$
3	3	10101	21	$2^{21} - 1 = 2\,(2^{10} - 1)\,(2^{10} + 1) + 1$
4	2	101011	43	$2^{43} - 1 = 2\,(2^{21} - 1)\,(2^{21} + 1) + 1$
5	1	1010110	86	$2^{86} - 1 = (2^{43} - 1)\,(2^{43} + 1)$
6	0	10101100	172	$2^{173} - 2 = 2\,(2^{86} - 1)\,(2^{86} + 1)$

Table 3.4: Intermediate values of s and r during inversion ($n = 173 = 10101101_2$).

The field inversion can be expressed in pseudo-code as

INPUT: $k \in F_{2^n}$

OUTPUT: $l = k^{-1}$, $l \in F_{2^n}$

1. Set $s \leftarrow \lfloor \log_2 n \rfloor - 1$
2. Set $p \leftarrow k$
3. For i from s downto 0
   - 3.1 Set $r \leftarrow$ shift n to right by s bit(s)
   - 3.2 Set $q \leftarrow p$
   - 3.3 Rotate q to left by $\lfloor r/2 \rfloor$ bit(s)
   - 3.4 Set $t \leftarrow$ multiply p by q
   - 3.5 If last bit of r is set then
     - 3.5.1 Rotate t to left by 1-bit
     - 3.5.2 $p \leftarrow$ multiply t by k

     else
     - 3.5.3 $p \leftarrow t$
   - 3.6 $s \leftarrow s - 1$
4. Rotate p to left by 1-bit
5. Set $l \leftarrow p$
6. Return l

$$2^5 - 1 = \underbrace{2}_{3.5.1}\underbrace{(2^2-1)}_{p}(\underbrace{2^2}_{3.3}\underbrace{+1}_{3.4})\underbrace{+1}_{3.5.2} \tag{3.13}$$

Table 3.4 helps to understand the algorithm where s represents which iteration is being performed (in reverse order), r is the shifted value of s in line 3.1 of the algorithm. Equation 3.13 illustrates the intermediate calculations for iteration 1. The algorithm starts by initializing s which represents the total number of iterations and r in line 3.3 helps to calculate the squarings shown in

Equation 3.13. After that, in line 3.4, a multiplication is needed to calculate the "+1" of Equation 3.13. Then the value of r is checked, if r is odd, an extra squaring and multiplication is required (line 3.5.1 and 3.5.2).

The total number of multiplies M required to perform an inversion in F_{2^n} using the above algorithm is

$$M(n) = \lfloor \log_2(n-1) \rfloor + \nu(n-1) - 1$$

where $\nu(x)$ is the number of nonzero bits in the binary representation of x.

3.7 Summary

The operations of elliptic curve cryptography is based on calculations in a finite field where there are two representations of the elements. An optimal normal basis leads to a more efficient hardware implementation, and was chosen for the elliptic curve processor. Optimal normal basis is the special case of normal basis such that it gives minimum hardware complexity. The design presented in this thesis assumes a key size n which has an ONB.

Chapter 4

Literature Review

4.1 Introduction

This chapter begins with reviewing previous notable hardware elliptic curve implementations. They are divided into two categories, field processors and curve processors. Field processors can operate field operations while curve processors can operate curve operations as well. Afterwards, previous software elliptic curve implementations are described.

4.2 Hardware Elliptic Curve Implementation

4.2.1 Field Processors

Previous implementations of ECC processors have mostly implemented coprocessors for performing the underlying field operations. An optimal normal basis multiplier over $F_{2^{593}}$ was reported in 1988. It used 90,000 transistors and occupied 0.3 inches on a side. This chip together with a Motorola DSP 56000 was used to implement a complete ECC system which could calculate 5 points a second on a supersingular curve [AMV93]. In 1993, the same team developed a processor for operating over the Galois field $F_{2^{155}}$ [AMV93]. Their multipliers of the processor was operated on normal basis. A coprocessor was built by

restricting the functionality of the device to the bare minimum necessary to implement the elliptic curve system. The coprocessor used 11,000 transistors and could operate at 40 MHz [AMV93]. The size of the coprocessor is less than 15 percent of the area of the smart cards available at that time. This implementation was intended to be compact yet secure.

A compact super–serial multiplier for FPGAs is reported in 1999 [OP99]. Their design offers fine–grained scalability and is a sliced design which can trade off performance and area. In a polynomial basis over $F_{2^{167}}$. They first implemented a serial multiplier (SM) which contains k slices and computes $GF(2^k)$ multiplication in k cycles. The super–serial multiplier (SSM) emulates the operation of the serial multiplier using a smaller number of slices but requires more clock cycles. It uses m ($m < k$) slices and computes $GF(2^k)$ multiplication in $k \times \lceil k/m \rceil$ cycles. The SSM was reported to use 2.76 times less function generators, 6.84 times less flip–flops and 5.78 times less CLBs than the SM.

A multiplier for dual-field arithmetic was introduced by Großschädl [Gro01]. They used a redundant representation (two n-bit binary numbers to represent a n-bit binary number) to address the carry propagation problem. The estimated number of clock cycles for a multiplication in $GF(p)$ is $1.5n$ (where n is the number of bits), while m clock cycles are required in $GF(2^m)$.

All the above implementations are Galois field processors which have the disadvantage that a high bandwidth interface is required to supply the coprocessor with its data. As an example, a curve multiplication may require several thousand field operations which must be sent to a field processor. For a curve processor, these overhead of the transfers is avoided. Another limitation of previous designs is that the field operations are restricted to certain groups (i.e. the subfield and extension fields of $F_{2^{155}}$) and these cannot be changed without fabricating a new chip.

4.2.2 Curve Processors

A FPGA based processor for elliptic curve cryptography in a composite Galois field $F_{(2^n)^m}$ was developed by Rosner [Ros98b]. It could perform elliptic curve operations over a composite Galois fields $GF((2^n)^m)$ in a polynomial basis representation. Moreover, it was developed on a reconfigurable FPGA platform, it is possible to easily change all algorithm parameters such as curve coefficients.

A scalable, FPGA-based elliptic curve processor over $GF(p)$ was reported by Orlando and Paar [OP01]. Their processor uses a new type of high-radix Montgomery multiplier which allows precomputation of frequently used values so that the complexity of the multiplier is reduced. The frequency of operation of the processor is 40 MHz on Xilinx Virtex-E FPGA.

4.3 Software Elliptic Curve Implementation

A fast key exchange using an elliptic curve system was reported in 1995 [SOOS95]. It is a software implementation using the group of points on an elliptic curve over $GF(2^{155})$ in a polynomial basis representation. This software implementation was optimized by a fast inversion algorithm, "Almost Inverse Algorithm". In fact, it is an optimized version of the Euclidean algorithm which only takes about three multiplication times for an field inversion. The implementation was tested on two platforms, on a SUN SPARC IPC (25 MHz, 32–bit architecture), it performed a elliptic curve multiplication in 124 ms and on a DEC Alpha 3000 (175 MHz, 64–bit architecture), it took 9.9 ms.

In 1997, Guajardo and Paar reported three new efficient algorithms of the implementations of elliptic curve cryptosystems [GP97]. They are one high–level algorithm for point multiplication on elliptic curve and two low–level algorithms for finite field inversion and multiplication. For the point multiplication, they introduced a new approach that works in conjunction with

the k–ary and the sliding window methods. Their method can be applied to elliptic curve over any field, however they described their new approach with non–supersingular elliptic curves over $GF(2^k)$. It was based on the adapted version of left–to–right k-ary exponentiation algorithm stated in [MOV99]. The algorithm was modified so that it applied to EC. A new approach to point doubling operation was introduced. It allows computation of $2^kP = (x_k, y_k)$ directly from $P = (x, y)$ without computing the intermediate points $2P$, 2^2P,$\cdots$,$2^{k-1}P$. Such direct formulae can be obtained by substituting x_3, y_3 in Equation 5.6 into one another. By doing that, the point doubling, so as the multiplication, can be accelerated. For the two new purposed finite field operations, they work on ECs over a composite Galois field $GF((2^n)^m)$. The new inversion algorithm is based on the idea in [IT88] but is optimized for a polynomial basis representation and for field of characteristic two. The algorithm reduces inversion in the composite field to inversion in the field $GF(2^n)$. On the other side, the new multiplication algorithm is based on the Karatsuba–Ofman Algorithm (KOA) to polynomials over $GF((2^n)^m)$.

There is a full ECC package in [Ros98a] that was used to evaluate the performance of the design in this thesis. This package implemented main protocols in cryptography, for example, ElGamal key sharing [MOV99], IEEE P1363 MQV key sharing [ieee], digital signature algorithm (DSA) [ieee], Nyberg–Rueppel signature algorithm [ieee], etc. The package is an optimized software implementation for both polynomial and optimal normal basis representations. Optimizations include efficient scalar multiplication using a balanced binary expansion [Kob91] and the "Almost Inverse Algorithm" [SOOS95] described above.

4.4 Summary

In this chapter, previous hardware and software implementations of elliptic curve cryptosystems were presented. There are two kinds of hardware implementation namely field processors and curve processors.

Field processors have the disadvantage that a high bandwidth interface is required to supply the coprocessor with its data. Previous curve processors have concentrated on $GF(p)$ and the author is not aware of any curve processors which operate over $GF(2^n)$.

Chapter 5

Introduction to Elliptic Curves

5.1 Introduction

The chapter begins with a background to elliptic curves. It is followed by a discussion of the curve addition and curve doubling operations on elliptic curve over real numbers. Afterwards, elliptic curve over finite fields as well as their operations are detailed. Finally, elliptic curve operations applied to the discrete logarithm problem and key exchange protocol are shown.

5.2 Historical Background

Elliptic curves were first discovered after the 17th century in the form of Diophantine equation [ST92]

$$y^2 - x^3 = c$$

for $c \in \mathbb{Z}$. In 1955, Yutaka Taniyama asked some questions about elliptic curves, i.e. curves of the form $y^2 = x^3 + ax + b$ for constants a and b [RS97] and Hellegouarch studies the application of elliptic curves for solving Fermat's Last Theorem in 1971 [Hel71]. Elliptic curves were proposed for cryptographic purposes by Koblitz [Kob87b] and Miller [Mil86] in 1985. The discrete logarithm problem over the group of points on an elliptic curve over finite field

is an attractive one way function because there is no sub-exponential attack known for solving this problem. The advantages of EC over other public key cryptosystems had been discussed in Section 2.4.

5.3 Elliptic Curves over $\mathbb{R}^2$

Elliptic curves E over the real numbers ($\mathbb{R}$) are sets of points in the form of (x, y), $x, y, a_4, a_6 \in \mathbb{R}$ that satisfy the equation

$$y^2 = x^3 + a_4 x + a_6 \tag{5.1}$$

together with a special point $\mathcal{O}$, called the point at infinity which is an identity element. The variables x and y represent a point on elliptic curve and cover a two dimensional (affine) coordinate plane, $\mathbb{R} \times \mathbb{R}$. Elliptic curve E over $\mathbb{R}^2$ is said to be defined over $\mathbb{R}$, denoted by $E(\mathbb{R})$. Elliptic curve over reals can be used to form a group $(E(\mathbb{R}), +)$ consisting of the set of points $(x, y) \in \mathbb{R} \times \mathbb{R}$ together with an addition operation $+$ on $E(\mathbb{R})$.

Figure 5.1 shows a plot of an elliptic curve over $\mathbb{R}$.

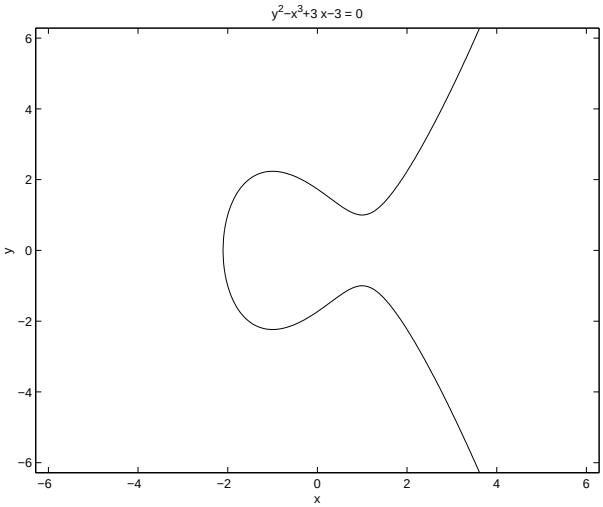


Figure 5.1: Plot of elliptic curve $y^2 = x^3 - 3x + 3$.

5.3.1 Curve Addition and Doubling

The point addition (ESUM, also known as curve addition) operation + is defined on the set $E(\mathbb{R})$ of points (x, y). By the rule of identity, the point at infinity $\mathcal{O}$ is the point that added to any point on the elliptic curve, gives the same point. Therefore for all $P = (x, y) \in E(\mathbb{R})$,

$$P + \mathcal{O} = \mathcal{O} + P = P.$$

For each point $P(x, y) \in E(\mathbb{R})$, the square root of Equation 5.1 gives

$$\pm y = \sqrt{x^3 + a_4 x + a_6}.$$

Therefore two y-coordinate values are given by each unique value of x to Equation 5.1. The point $(x, -y)$, denoted $-P \in E(\mathbb{R})$, is called the negative of point P and specified as

$$P + (-P) = (x, y) + (x, -y) = \mathcal{O}. \tag{5.2}$$

Addition on $E(\mathbb{R})$ is defined geometrically. Suppose there are two distinct points P and Q, $P, Q \in E(\mathbb{R})$. The law of addition in the elliptic curve group is $P + Q = R$, $R \in E(\mathbb{R})$. The geometric relationship is shown in Figure 5.2.

In order to find the point R, first connect the points P and Q by a line L. By simultaneously solving equations L and E, an equation of degree three is derived with exactly three solutions. Therefore the line L is guaranteed to intersect the curve E on a third point, say $-R \in E(\mathbb{R})$. The point R can be obtained by negating the y-coordinate of $-R$.

In the case that the two points are P and $-P$ (Figure 5.3), the line connecting P and $-P$ intersects the elliptic curve at a third point which is the special point $\mathcal{O}$ lying on every vertical line in the coordinates plane.

In an operation of point addition, if points $P, Q \in E(\mathbb{R})$ are added where $P = Q$, then the tangent line to the elliptic curve at point P is taken instead

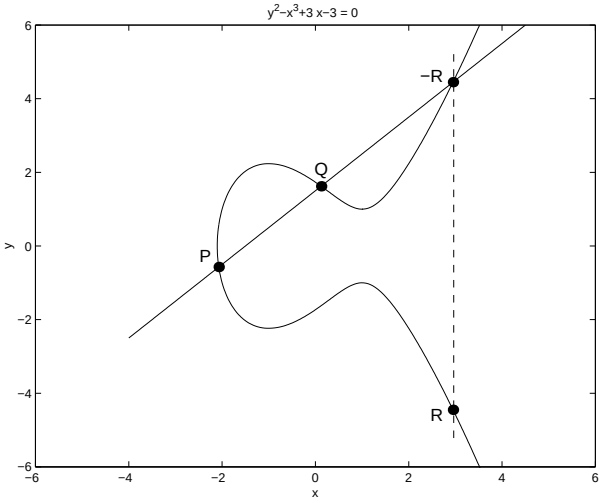


Figure 5.2: Addition of EC points.

(shown in Figure 5.4). For this case, it is a point doubling (EDBL) operation where $R = 2P$.

In order to express the definition of point addition and doubling mathematically,

$$\begin{aligned} P &= (x_1, y_1), \\ Q &= (x_2, y_2), \\ R &= P + Q = (x_3, y_3) \end{aligned}$$

where $P, Q, R \in E(\mathbb{R})$, and

$$\begin{aligned} x_3 &= \theta^2 - x_1 - x_2, \\ y_3 &= \theta(x_1 + x_3) - y_1, \\ \theta &= \frac{y_2 - y_1}{x_2 - x_1} \qquad \text{if} \qquad P \neq Q \end{aligned}$$

or

$$\theta = \frac{3x_1^2 + a_4}{2y_1} \qquad \text{if} \qquad P = Q.$$

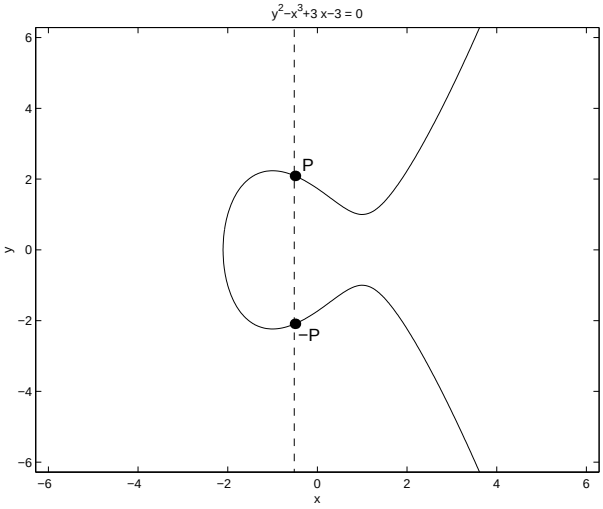


Figure 5.3: Addition of points P and $-P$ EC.

The algorithm of curve addition on $E(\mathbb{R})$ can be expressed as follows.

INPUT: Elliptic curve $E(\mathbb{R})$ with parameters $a_4, a_6 \in \mathbb{R}$,

Points $P = (x_1, y_1) \in E(\mathbb{R})$ and $Q = (x_2, y_2) \in E(\mathbb{R})$

OUTPUT: $R = P + Q$, $R = (x_3, y_3) \in \mathbb{R}$

1. If $P = \mathcal{O}$, then set $R \leftarrow Q$ and return R
2. If $Q = \mathcal{O}$, then set $R \leftarrow P$ and return R
3. If $x_1 = x_2$ then

   3.1 If $y_1 = y_2$ then

   3.2.1 Set $\theta \leftarrow \frac{3x_1^2 + a_4}{2y_1}$

   else set $R \leftarrow \mathcal{O}$ and return R, $\because y_1 = -y_2$

   else set $\theta \leftarrow \frac{y_2 - y_1}{x_2 - x_1}$
4. Set $x_3 \leftarrow \theta^2 - x_1 - x_2$
5. Set $y_3 \leftarrow \theta(x_1 + x_3) - y_1$
6. Return $(x_3, y_3) = R$

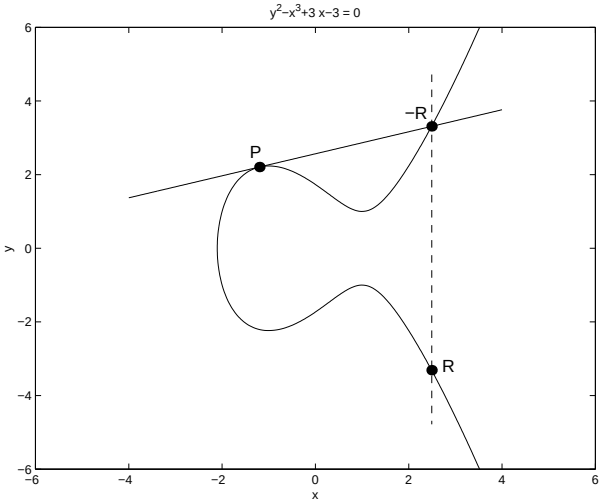


Figure 5.4: Doubling of EC point.

5.4 Elliptic Curves over Finite Fields

Elliptic curve cryptography is based on the elliptic curve over finite fields. There are two finite fields which are particularly attractive, they are prime fields (F_p) and binary finite fields (F_{2^n}). The derivations of elliptic curve equations from the Weierstrass equation is given in Appendix A.

5.4.1 Elliptic Curves over F_p with $p > 3$

A prime field F_p is generated using a large prime p [LN94]. The operations of elliptic curve over F_p is similar to $E(\mathbb{R})$. Instead of calculations on real numbers, the calculations modulo a large prime are taken. Therefore an elliptic curve E is defined over F_p, denoted by $E(F_p)$, if $x, y, a_4, a_6 \in F_p$ and $4a_4^3 + 27a_6^2 \neq 0$ satisfying the equation

$$y^2 = x^3 + a_4x + a_6.$$

Points on this curve form a group. Therefore the elliptic curve group $(E(F_p), +)$ is set of points (x, y) (for $x, y \in F_p$) and an operation $+$ (addition) which satisfies the axioms in Section 3.2.

The order of a point A on $E(F_p)$ is the smallest positive integer r such that

$$\underbrace{A + A + \cdots + A}_{r \text{ times}} = \mathcal{O}.$$

The order of the curve is the number of points on $E(F_p)$, denoted by $\#E(F_p)$. By Hasse's theorem [Kob87a, Men93], $\#E(F_p) = p + 1 - t$, where $|t| \leq 2\sqrt{p}$.

5.4.2 Elliptic Curves over F_{2^n}

A non-supersingular elliptic curves E defined over a finite field F_{2^n} (characteristic = 2), denoted by $E(F_{2^n})$, is the set of solutions, (x, y) for $x, y \in F_{2^n}$, to the simplified forms of the *Weierstrass equation*

$$y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6 \tag{5.3}$$

where $a_1, a_2, a_3, a_4, a_6 \in F_{2^n}$, namely

$$y^2 + xy = x^3 + a_2x^2 + a_6 \tag{5.4}$$

where $a_6 \neq 0$ and $x, y, a_2, a_6 \in F_{2^n}$. Again, an identity element (point of infinity) $\mathcal{O}$, is included in both curve. Elliptic curve E over F_{2^n} also forms a group $(E(F_{2^n}), +)$ that satisfies the axioms in Section 3.2.

The curves of Equation 5.4 are called non-supersingular curves and are suitable for cryptographic applications [BSS99]. From a hardware implementation perspective, ECs over F_{2^n} are thought to be very practical. The benefits of using ECC are:

- ECC offers the highest security per bit of any known public key cryptosystem so a smaller memory can be used
- ECC hardware implementations use less transistors, as an example, a VLSI implementation of a 155-bit ECC processor has been reported which uses only 11,000 transistors [AMV93], compared with an equivalent strength 512-bit RSA processor which used 50,000 transistors [PID92].

5.4.3 Operations of Elliptic Curves over F_{2^n}

A non–supersingular elliptic curve E over F_{2^n}, $E(F_{2^n})$ was selected for the implementation of elliptic curve processor. $E(F_{2^n})$ is the set of all solutions to the Equation 5.4 with coordinates in the algebraic closure of $E(F_{2^n})$ [Men93], where $a_2, a_6 \in F_{2^n}$ and $a_6 \neq 0$. Such an elliptic curve is a finite abelian group [Men93]. The number of points in this group is denoted by $\#E(F_{2^n})$.

The processor was implemented in two different coordinates systems, affine and projective coordinates. For different coordinates systems, the computation of the curve operations are also different. They are discussed in next section.

Elliptic Curve Operations in Affine Coordinates

The elliptic curve cryptosystems that will be used are based on the discrete logarithm problem over $E(F_{2^n})$ and the basic computation which must be made is curve multiplication. Curve multiplication is expressed as a sequence of point additions and point doublings. Similar to EC over real numbers, point addition and point doubling are defined geometrically. It is hard to represent an elliptic curve over a finite field graphically, however, the method of finding the point of addition and doubling are the same as shown at Section 5.3.1.

Assume a non-supersingular elliptic curve E over F_{2^n} given in affine coordinates and P, Q are two points on $E(F_{2^n})$. Let $P = (x_1, y_1)$, $Q = (x_2, y_2)$, then negative of P is $-P = (x_1, y_1 + x_1) \in E(F_{2^n})$. From [SOOS95], if $Q \neq -P$, then $P + Q = R = (x_3, y_3) \in E(F_{2^n})$.

If $P \neq Q$

$$\left.\begin{array}{rcl} \theta & = & \frac{y_1+y_2}{x_1+x_2}, \\ x_3 & = & \theta^2 + \theta + x_1 + x_2 + a_2, \\ y_3 & = & (x_1 + x_3)\theta + x_3 + y_1. \end{array}\right\} \tag{5.5}$$

Otherwise if $P = Q$

$$\left.\begin{aligned} \theta &= \tfrac{y_1}{x_1} + x_1, \\ x_3 &= \theta^2 + \theta + a_2, \\ y_3 &= (x_1 + x_3)\theta + x_3 + y_1. \end{aligned}\right\} \tag{5.6}$$

Therefore, in affine coordinates, both point addition (ESUM) and point doubling (EDBL) require two multiplications and one field inversion (note that field inversion is far more expensive than field multiplication). The following algorithm shows curve addition on $E(F_{2^n})$ in terms of affine coordinates.

INPUT: A non-supersingular elliptic curve $E(F_{2^n})$ with parameters $a_2, a_6 \in F_{2^n}$, Points $P = (x_1, y_1) \in E(F_{2^n})$ and $Q = (x_2, y_2) \in E(F_{2^n})$

OUTPUT: $R = P + Q$, $R = (x_3, y_3) \in F_{2^n}$

1. If $P = \mathcal{O}$, then set $R \leftarrow Q$ and return R
2. If $Q = \mathcal{O}$, then set $R \leftarrow P$ and return R
3. If $x_1 = x_2$ then
   - 3.1 If $y_1 = y_2$ then
     - 3.2.1 Set $\theta \leftarrow \frac{y_1}{x_1} + x_1$ and $x_3 \leftarrow \theta^2 + \theta + a_2$

     else set $R \leftarrow \mathcal{O}$ and return R, $\because y_2 = x_1 + y_1$

   else set $\theta \leftarrow \frac{y_1 + y_2}{x_1 + x_2}$ and $x_3 \leftarrow \theta^2 + \theta + x_1 + x_2 + a_2$
4. Set $y_3 \leftarrow (x_1 + x_3)\theta + x_3 + y_1$
5. Return $(x_3, y_3) = R$

Elliptic Curve Operations in Projective Coordinates

A non-supersingular curve $E(F_{2^n})$ can be equivalently viewed as the set of all points $E'(F_{2^n})$ in the projective plane $P^2(F_{2^n})$ which satisfy [Men93]

$$y^2z + xyz = x^3 + a_2x^2z^2 + a_6z^3. \tag{5.7}$$

By using projective coordinates, the inversion operation which is needed in ESUM and EDBL operating in affine coordinates can be eliminated and it is covered in next section.

Conversion Between Affine and Projective Coordinates

For any point $(a, b) \in E(F_{2^n})$ in affine coordinates can be viewed as a 3-tuple $(x, y, z) \in E'(F_{2^n})$ in projective coordinates with $x = a$, $y = b$ and $z = 1$. Moreover, a point (tx, ty, tz) in projective coordinates with $t \neq 0$, is regarded as the same point as (x, y, z). Therefore the conversion methods between affine and projective coordinates are given as follows

$$M(a, b) = M'(a, b, 1)$$
$$N'(p, q, r) = N'\left(\frac{p}{r}, \frac{q}{r}, 1\right) = N\left(\frac{p}{r}, \frac{q}{r}\right) \quad \text{for } r \neq 0$$

where M, N are point in affine coordinates and M', N' are projective points.

Curve Operations in Projective Coordinates

The method of formulating the equations of addition and doubling in projective coordinates is the same as for affine. In fact, conversion is taken on each projective point and then applied to Equation 5.5 and Equation 5.6.

Let $P' = (x_1 : y_1 : z_1) \in E'(F_{2^n})$, $Q' = (x_2 : y_2 : 1) \in E'(F_{2^n})$ and $P' \neq -Q'$ where P', Q' are in projective coordinates. Since $P' = (x_1/z_1 : y_1/z_1 : 1)$, one can apply Equation 5.5 to point $P(x_1/z_1, y_1/z_1)$ and $Q(x_2, y_2)$ for $E(F_{2^n})$ in affine coordinates to find $P' + Q' = R'(x_3' : y_3' : 1)$. Then

$$\begin{aligned} x_3' &= \frac{B^2}{A^2} + \frac{B}{A} + \frac{A}{z_1} + a_2, \\ y_3' &= \frac{B}{A}\left(\frac{x_1}{z_1} + x_3'\right) + x_3' + \frac{y_1}{z_1} \end{aligned}$$

where $A = (x_2 z_1 + x_1)$ and $B = (y_2 z_1 + y_1)$ [Men93]. In order to eliminate the inversion operations, the denominators of the expressions for x_3' and y_3' have to be eliminated. By setting $z_3 = A^3 z_1$ and from the property of projective coordinates, $x_3 = x_3' z_3$ and $y_3 = y_3' z_3$, if $P' + Q' = (x_3 : y_3 : z_3)$, then

$$\begin{aligned} x_3 &= AD, \\ y_3 &= CD + A^2(Bx_1 + Ay_1), \\ z_3 &= A^3 z_1 \end{aligned}$$

Operation	Affine		Projective	
	ESUM	EDBL	ESUM	EDBL
Field Multiplication	2	2	13	7
Field Inversion	1	1	0	0

Table 5.1: Number of field multiplications and inversions for affine and projective point addition and doubling.

where $C = A + B$ and $D = A^2(A + a_2 z_1) + z_1 BC$.

Similarly, the formulae for $2P = (x_3 : y_3 : z_3)$ are,

$$\begin{aligned} x_3 &= AB, \\ y_3 &= x_1^4 A + B(x_1^2 + y_1 z_1 + A), \\ z_3 &= A^3 \end{aligned}$$

where $A = x_1 z_1$ and $B = a_6 z_1^4 + x_1^4$. The resulting point can be converted back to affine coordinates by multiplying each coordinate by z_3^{-1}. Note that there is no inversion operation when calculating in projective coordinates. Therefore inversion can be eliminated by performing curve multiplication in projective coordinates and using only one inversion after a series of additions and doublings. The number of field multiplications and field inversions for curve addition and doubling are shown in Table 5.1.

5.4.4 Curve Multiplication

Multiplication (EMUL) is defined by repeated addition, i.e.

$$Q = cP \tag{5.8}$$

$$= \underbrace{P + P + \ldots + P}_{c \text{ times}} \tag{5.9}$$

This can be computed using the following "double and add" algorithm.

For affine coordinates,

INPUT: $P \in E(F_{2^n})$ and $c \in F_{2^n}$

OUTPUT: $Q = cP$

1. $c = \sum_{i=0}^{m} b_i 2^i, b_i \in \{0,1\}, b_m = 1, m \leq n$
2. Set $Q \leftarrow P$
3. For i from $m-1$ downto 0
   - 3.1 Set $Q \leftarrow Q + Q$ (Affine EDBL)
   - 3.2 If $b_i = 1$ then
     - 3.2.1 Set $Q \leftarrow Q + P$ (Affine ESUM)
4. Return Q

For projective coordinates,

INPUT: $P \in E(F_{2^n})$ and $c \in F_{2^n}$

OUTPUT: $Q = cP$

1. $c = \sum_{i=0}^{m} b_i 2^i, b_i \in 0,1, b_m = 1, m \leq n$
2. Convert P to projective representation, P'
3. Set $Q' \leftarrow P'$
4. For i from $m-1$ downto 0
   - 4.1 Set $Q' \leftarrow Q' + Q'$ (Projective EDBL)
   - 4.2 If $b_i = 1$ then
     - 4.2.1 Set $Q' \leftarrow Q' + P'$ (Projective ESUM)
5. Convert Q' to affine representation Q
6. Return Q

which requires $m + \nu(c) - 2$ point additions where $\nu(c)$ is the number of nonzero bits in the binary representation of c.

The disadvantage using in projective coordinates is that extra registers are needed to store the intermediate results. However, it trades off an inversion

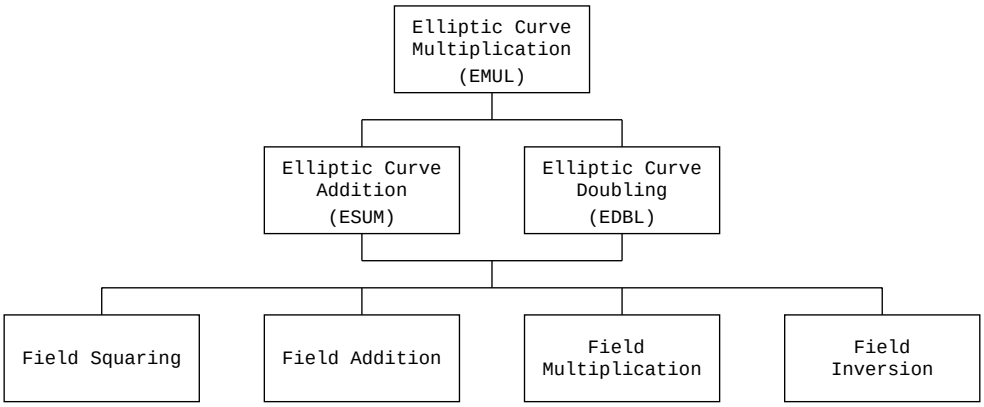


Figure 5.5: The hierarchy of elliptic curve operation.

for more multiplications and other less expensive finite field operations such that the overall performance can be improved.

The hierarchy of elliptic curve operation is shown in Figure 5.5. Curve multiplication is computed via curve additions and doubling which are in turn computed from field operations.

5.5 Elliptic Curve Discrete Logarithm Problem

ECC is based on the discrete logarithm problem applied to elliptic curves over a finite field (Section 2.4.3), known also as the elliptic curve discrete logarithm problem (ECDLP) which is defined as follows.

Given Q and Y, find x for which

$$Q = xY$$

where $x \in \{1, \ldots, \#Y - 1\}$ and Q, Y are points on elliptic curve $E(K)$, K is a finite field.

There is currently no known sub-exponential time algorithms to compute x given Q and Y [BSS99]. Although the index–calculus method is a sub-exponential time algorithm for solving the discrete logarithm problem, it is

not applicable to multiplicative groups in a finite field [BSS99] such as the elliptic curve group. The most efficient algorithm known is the Pollard-ρ method [Pol75]. It is parallelized and the expected running is $\sqrt{\pi n}/(2r)$ with r processors [OW99]. However, the running time is still exponential in n. Therefore the methods for computing ECDLP are much less efficient than those for factoring or DLP (Section 2.4). As a result, ECC provides shorter key sizes than others public key cryptosystems with the same security level.

5.6 Public Key Cryptography

The elliptic curve cryptosystem is a public key cryptosystem. In order to communicate with others using ECC, parties must know the public key of one another. Before the communication, ECC key pairs are generated. The key generation process includes following steps.

First define a set of elliptic curve domain parameters $(E(K), Q, \#Q)$ where $E(K)$ is particular elliptic curve defined over finite field K, Q is a base point on $E(K)$ with its order $\#Q$:

1. Select a random number x where $1 \leq x \leq \#Q - 1$,
2. Compute $P = xQ$,
3. Then P is public key and x is private key.

Suppose Alice wants to send a secret message M to Bob using ECC. Public parameters are the elliptic curve E over finite field K and Q is a point on $E(K)$.

Private key of Bob: x, where $1 \leq x \leq \#Q - 1$.

Public key of Bob: (P, Q), where $P = xQ$ and $P \in E(K)$.

Alice generates a random integer $z \in K$ and calculates the curve multiplication

$$A(x_a, y_a) = zQ$$

and

$$T(x_t, y_t) = zP.$$

In addition, Alice has to embed the message, that she wants to send to Bob, onto the elliptic curve $E(K)$. Suppose $K = F_{2^n}$ and the message is expressed in a binary number. It can be obtained by changing the ASCII message to binary number directly. For F_{2^n}, the longest message that can be embedded in $E(F_{2^n})$ is of size $n - 5$ bits [Ros98a]. Then a n-bit binary number (x_m) is formed by the message with length $n - 5$ bits and 5 "don't care" bits that can be changed freely. This n-bit binary number is substituted into Equation 5.4 to solve y (y_m) using arithmetic on F_{2^n}. In the case there is no solution, the 5 "don't care" bits can be incremented and another attempt to solve the equation is made until y is found. Then the x_m and y_m form a point (x_m, y_m) on $E(F_{2^n})$ [Kob93]. In this case, Alice and Bob must compromise on the distributions of the "don't care" bits. It is safe to disclose these to the public and it does not affect the cryptographic strength. To embed the message to $E(K)$, Alice gets a point

$$M = (x_m, y_m) \in E(K)$$

and, calculates

$$B(x_b, y_b) = (x_t x_m, y_t y_m).$$

(A, B) is the ciphertext and is sent to Bob.

Bob receives a ciphertext which is comprised of two parts (U, V), where

$$U = (x_u, y_u) \quad \text{and} \quad V = (x_v, y_v).$$

He can decrypt the ciphertext by

$$xU = xzQ = zP = T(x_t, y_t)$$

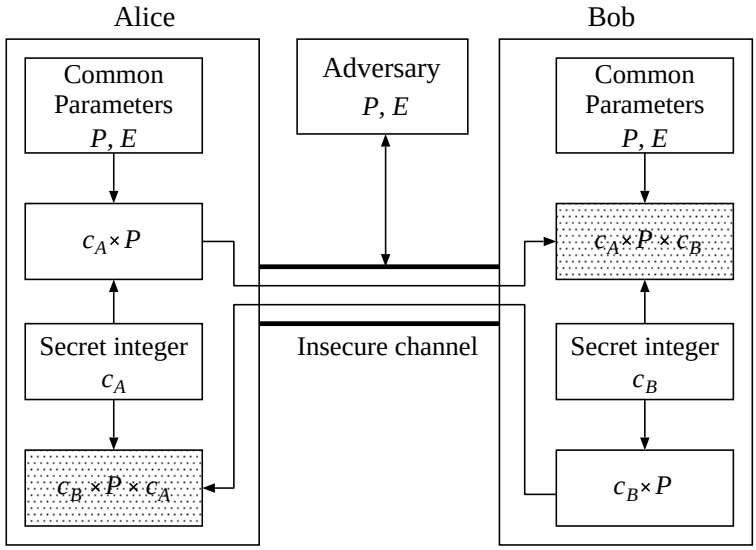


Figure 5.6: Diffie–Hellman key exchange scheme.

and

$$x_m = \frac{x_v}{x_t},$$
$$y_m = \frac{y_v}{y_t}.$$

Therefore Bob can get the point (x_m, y_m). Since Bob also knows the locations of "don't care" bits, he can retrieve the message M by eliminating the bits.

5.7 Elliptic Curve Diffie–Hellman Key Exchange

The discrete logarithm problem can be used as the basis of various public key cryptographic protocols for key exchange, encryption and digital signatures. Elliptic curve Diffie–Hellman key exchange scheme is the elliptic curve analogue of Diffie–Hellman key exchange (Section 2.4.5). An example is given below.

Suppose that Alice and Bob wish to agree on a common key which is used for encryption using a traditional secret key algorithm such as data encryption standard (DES), but need to do so over an insecure channel such as the Internet. Then the following Diffie–Hellman (DH) procedure (illustrated in

Figure 5.6) can be used with a public elliptic curve over finite field K, $E(K)$ and point $P \in E(K)$.

1. Alice generates a secret random integer $c_A \in 1, \ldots \#G - 1$ and sends the point $c_A \times P$ to Bob

2. Bob generates a secret random integer $c_B \in 1, \ldots \#G - 1$ and sends the point $c_B \times P$ to Alice

3. Alice and Bob can both compute the key $k = c_A \times (c_B \times P) = c_B \times (c_A \times P)$

An adversary, Carol eavesdropping on the channel, can only gain the information $E(K)$, P, $c_A \times P$ and $c_B \times P$. For Carol to be able to compute k, she must solve the elliptic curve discrete logarithm problem and the best known algorithm takes fully exponential time. Alice and Bob, however, need only compute elliptic curve multiplications which are comparatively easy.

5.8 Summary

In this chapter, an introduction to elliptic curves over real numbers and finite fields were presented. The basic operations, curve addition and curve doubling of elliptic curve were also described. Afterwards, the elliptic curve discrete logarithm problem on which ECC is based as well as the elliptic curve Diffie–Hellman key exchange were presented. There is no known sub-exponential time algorithm to solve this problem while there are efficient algorithms to solve the factorization problem used by RSA.

Chapter 6

Design Methodology

6.1 Introduction

In this chapter, the hardware platform and tools used to develop the elliptic curve processor is introduced. The implementation of elliptic curve processor (ECP) and its architecture of are also detailed.

At the beginning of this chapter, the computer–aided design (CAD) tools that used are discussed. In next section, the hardware platform that was used to implement the processor is introduced. Finally, the implementations of each component of the processor is discussed. The features of the processor, for instance the parallel multiplier, microcode implementation and module generator, are demonstrated in this chapter.

6.2 CAD Tools

The ECP was developed using the Very High Speed Integrated Circuit Hardware Description Language (VHDL) [Ska96]. It is a language that is used to describe hardware architectures. A considerable advantage of using VHDL is that simulation can be done before the real hardware is built. The chosen implementation platform was an Field Programmable Gate Array (FPGA).

FPGAs are re-programmable via a software downloadable bitstream and provide a short turn around time and low cost. The synthesis and implementation tools will generate a bitstream from the VHDL description that can be downloaded to the reconfigurable hardware. The synthesis and implementation tools used were Synopsys FPGA Express™ 3.4 and Xilinx Foundation™ 3.2i respectively.

The development cycle is shown in Figure 6.1. FPGAs are suitable for hardware prototyping because they provide a short turnaround time. If the simulation of the VHDL description does not work properly, the code can be corrected and simulated again immediately until the simulation is correct. This can improve the efficiency of the design process. During the synthesis process, the synthesis tool takes the VHDL code as input and generates a netlist. Afterwards, implementation software reads the netlist and maps the logic to the components in the target FPGA. Then the "place and route" tools place the components in the FPGA and route the connections within the design. Finally, the implementation tool generates the bitstream file of the design for a specific FPGA. The bitstream can be downloaded to the FPGA to configure it, after which the FPGA will perform the function.

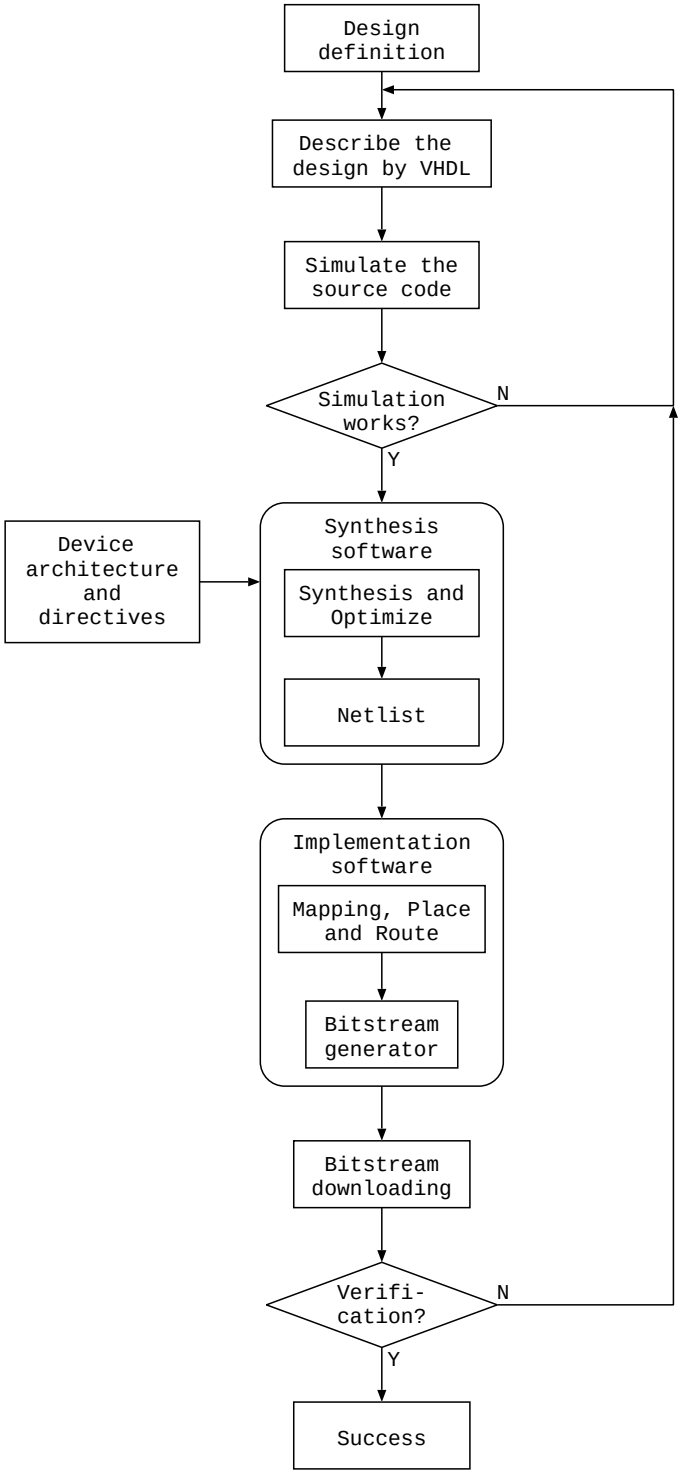


Figure 6.1: Development cycle.

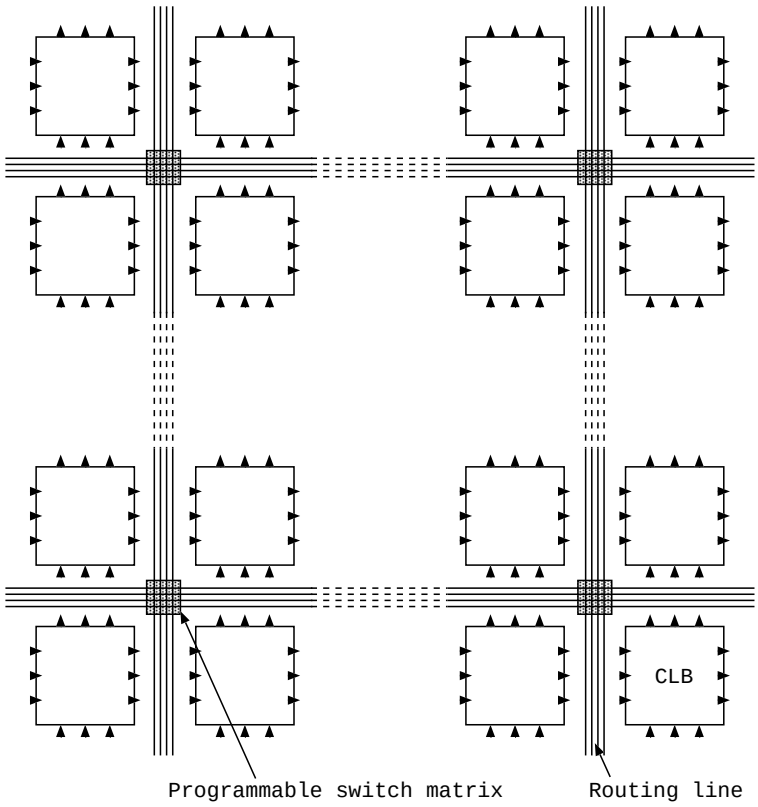


Figure 6.2: Xilinx SRAM-based FPGA structure.

6.3 Hardware Platform

6.3.1 FPGA

An FPGA is an integrated circuit (IC) that can be programmed after manufacture. Since it is re-programmable under field, it is a kind of reconfigurable hardware. Typical architecture of an FPGA comprises an regular array of *configurable logic blocks* (CLBs) with routing resources for interconnection and surrounded by programmable *input/output blocks* (IOBs). CLBs provide the functional elements for constructing logic while IOBs provide the interface between the pins of the package and the CLBs. FPGAs are widely used as a prototype before fabricating a VLSI design, or can be used directly in a product.

The elliptic curve processor is built on the Xilinx Virtex^{TM} FPGA. It is

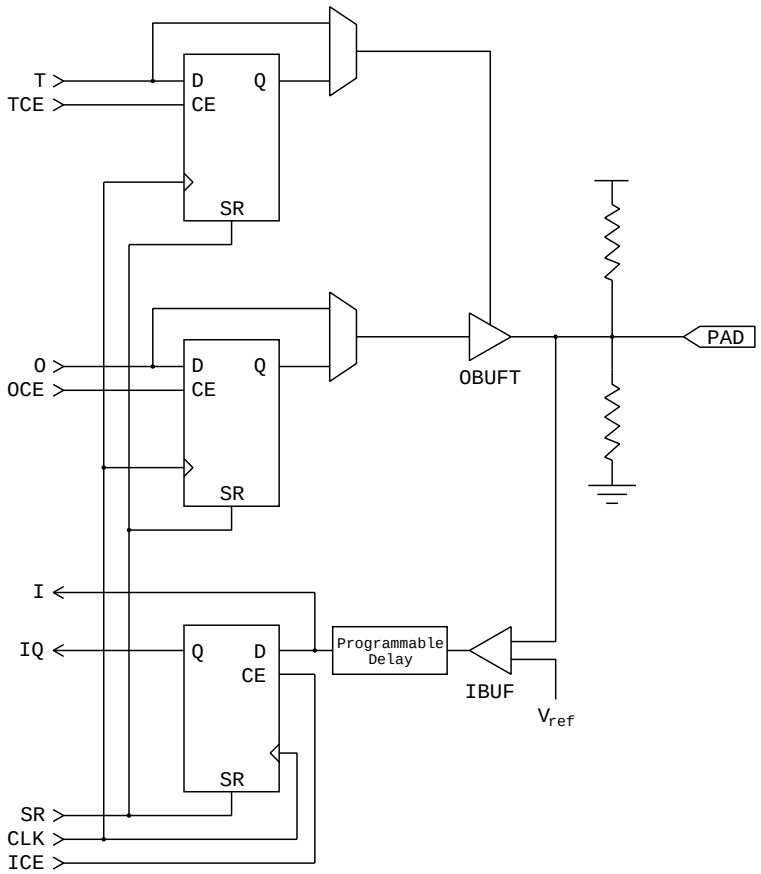


Figure 6.3: Structure of Xilinx Virtex IOB.

manufactured in a 5-layer-metal 0.22 μm CMOS process. Designs on Virtex can achieve synchronous system clock rates up to 200 MHz [Xil00]. Figure 6.2 shows the basic structure of Xilinx SRAM-based FPGAs. CLBs in the FPGA are arranged in rows and columns and between them are routing lines connected with programmable switch matrix. By programming the switch matrix, the I/O of CLBs can be routed and connected together.

The structure of Xilinx Virtex IOB is shown in Figure 6.3. The three D-type flip-flops are synchronized on the same clock while two are for input and output together with the other one is for the control to the output tri-state buffer. The input signal can be routed to internal logic either directly or through an input flip-flop. A programmable delay element at the D-input of the input flip-flop is to eliminate the pad-to-pad hold time. Moreover, by configuring

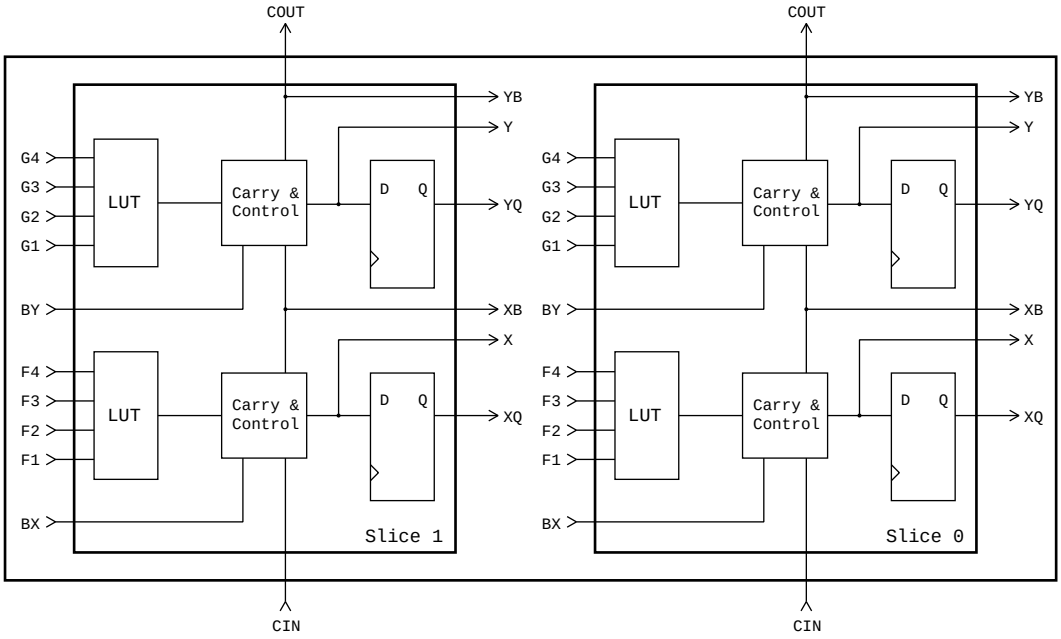


Figure 6.4: Simplified structure of Xilinx Virtex CLB.

the threshold voltage V_{ref} at the input buffer, the device can support designs with different voltage level. Similarly, the output from internal logic can be routed to the pad either directly or through the optional output flip-flop. All I/O pins not involved in configuration are set to high impedance state so that the internal logic is isolated.

The basic building block of the Virtex FPGA is the the logic cell (LC). A LC includes a 4–input function generator, carry logic and a storage element. Each Virtex CLB contains four LCs, organized in two slices (Figure 6.4). The 4-input function generator are implemented as 4-input look-up tables (LUTs). Each of them can provide the functions of one 4-input LUT or a 16×1-bit synchronous RAM (called "distributed RAM"). Furthermore, two LUTs in a slice can be combined to create a 16×2-bit or 32×1-bit synchronous RAM, or a 16×1-bit dual-port synchronous RAM [Xil00].

Also within the Virtex chip are dedicated several large blocks of special memories, called Block SelectRAMs or BlockRAMs. They are configurable as single-ported or dual-ported in widths from 1 to 16-bits [Xil00]. These were

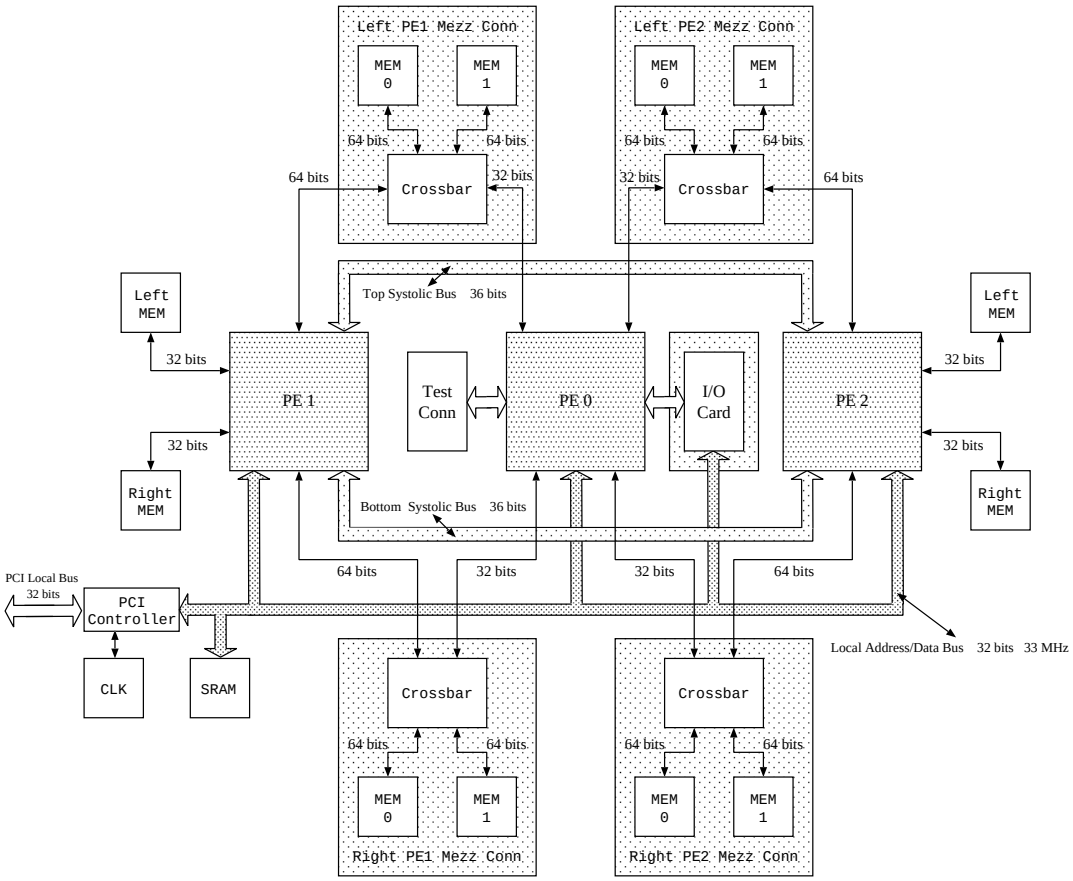


Figure 6.5: Wildstar block diagram.

used in the ECP for the storage of microcode and for communications with the host (discussed in Section 6.4.3).

6.3.2 Reconfigurable Hardware Computing

The hardware platform that used to implement the elliptic curve processor is Annapolis Micro Systems Wildstar^{TM} board [Ann99]. It is a PCI board consisting of 3 Processing Elements (Xilinx Virtex XCV1000–6 [Xil00]). The Virtex FPGA XCV1000–6 has 128 $k$bits of BlockRAM (arranged as 16×8–$k$bit blocks). In addition, there are 6144 configurable logic blocks (12288 slices). It is equivalent to more than one million system gates.

Figure 6.5 shows the block diagram of the Wildstar board [Ann99]. The

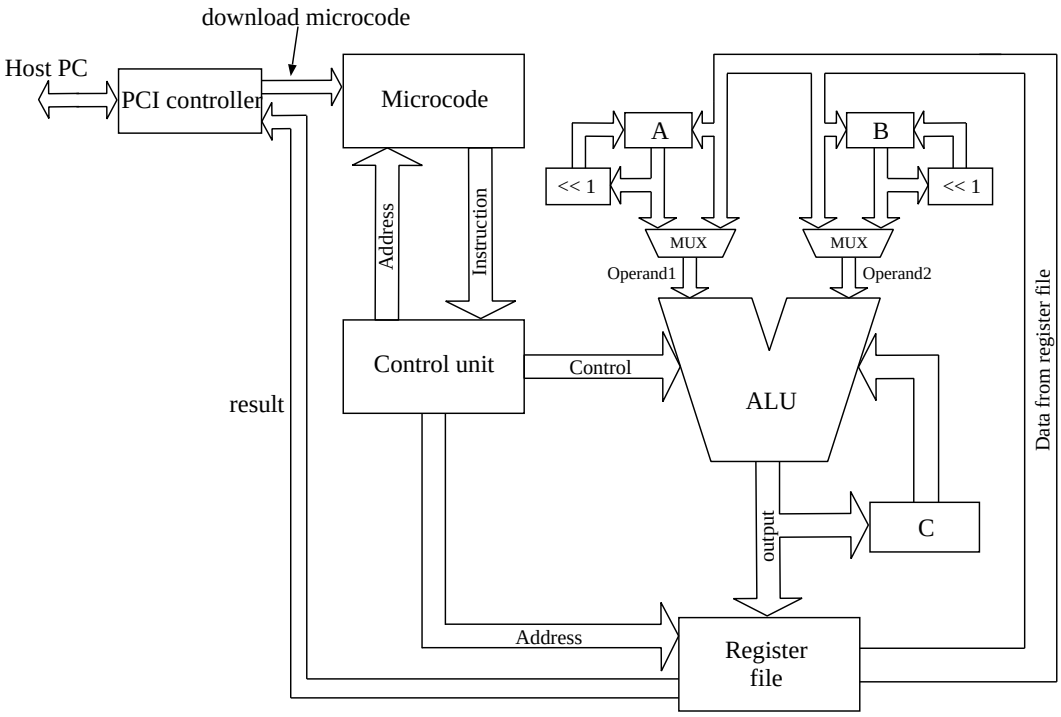


Figure 6.6: ECP architecture.

three Processing Elements (PEs) are connected to a Local Address/Data bus (LAD bus) which is used for communicating with the host computer. PE1 and PE2 are connected to on-board memories. Mezzanine memory card can be installed on the Wildstar board to provide extra memory which is shared between three PEs. The elliptic curve processor described in this thesis used PE0 only.

6.4 Elliptic Curve Processor Architecture

A block diagram of the elliptic curve processor is shown in Figure 6.6. The organization is similar to a traditional microcoded central processing unit (CPU) in that it consists of an arithmetic logic unit (ALU), a register file, a microcode sequencer and microcode storage. To compute a curve multiplication, the curve parameters are first initialized by bitstream modification (Section 6.7). The resulting bitstream is downloaded to the FPGA and the microcode is downloaded via the PCI interface (Figure 6.6). The ECP then computes a curve

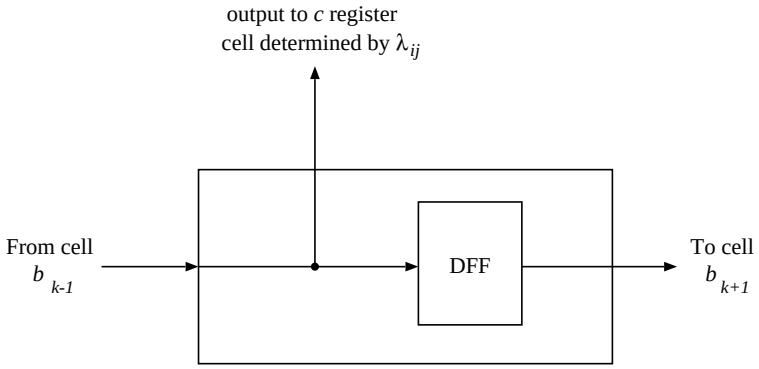


Figure 6.7: F_{2^n} multiplier element of b_k.

multiplication, while the PC polls a status signal. When the multiplication has completed, the PC can download the result from the register file (Figure 6.6). Major differences between this architecture and a conventional CPU are that the datapath is n-bits wide and the ALU performs operations based on F_{2^n} arithmetic instead of integer arithmetic (see Section 3.6). Three registers A, B and C are particularly for multiplication use and it will be discussed in next section.

6.4.1 Arithmetic Logic Unit (ALU)

As discussed in Chapter 3, there is no carry propagation in the arithmetics based on F_{2^n}. Therefore the ALU of elliptic curve processor is simpler and faster than an integer ALU. The complexity is determined by the F_{2^n} multiplier since multiplication is the only operation that involves registers.

The multiplication is defined by Equation 3.6 and is given below

$$c_k = \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} \lambda_{ij0} a_{i+k} b_{j+k}. \tag{6.1}$$

It can be rewritten as

$$c_k = \sum_{j=0}^{n-1} b_{j+k} \sum_{i=0}^{n-1} \lambda_{ij0} a_{i+k}$$

with all subscripts are modulo n.

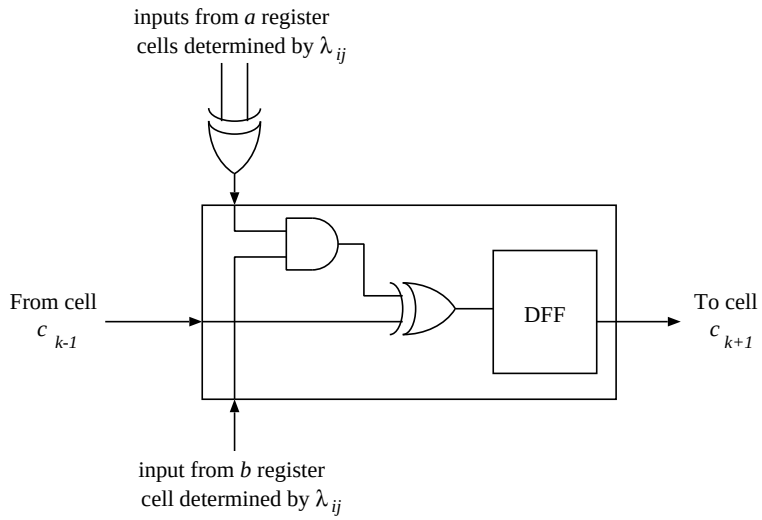


Figure 6.8: F_{2^n} multiplier element of c_k.

Define a function of the cyclic permutation

$$F_j(k) = b_{j+k} \sum_{i=0}^{n-1} \lambda_{ij0} a_{i+k}$$

again with all subscripts are taken to be modulo n. For cyclic relationship, j is set to be equal to k, therefore

$$F(k) = b_{2k} \sum_{i=0}^{n-1} \lambda_{ik0} a_{i+k}. \tag{6.2}$$

The function F in Equation 6.2 defines the connection between registers A, B and C.

Figure 6.7 shows the 1-bit register used for register B in the multiplier and a similar circuit is used for register A. It is just simply a D-type flop-flip in order to operate the cyclic shift on register A and B. Figure 6.8 shows the circuit used for calculating the c_k of Equation 6.2 [AMOV91].

In each cycle, the c_k'th cell computes sum of inputs from register A and then adds with the input from register B. The addition operation is implemented simply as an XOR function (Section 3.6.1) and the squaring function is implemented as a rotate left operation (Section 3.6.2). Therefore Equation 6.2 is calculated in the cell c_k every clock cycle.

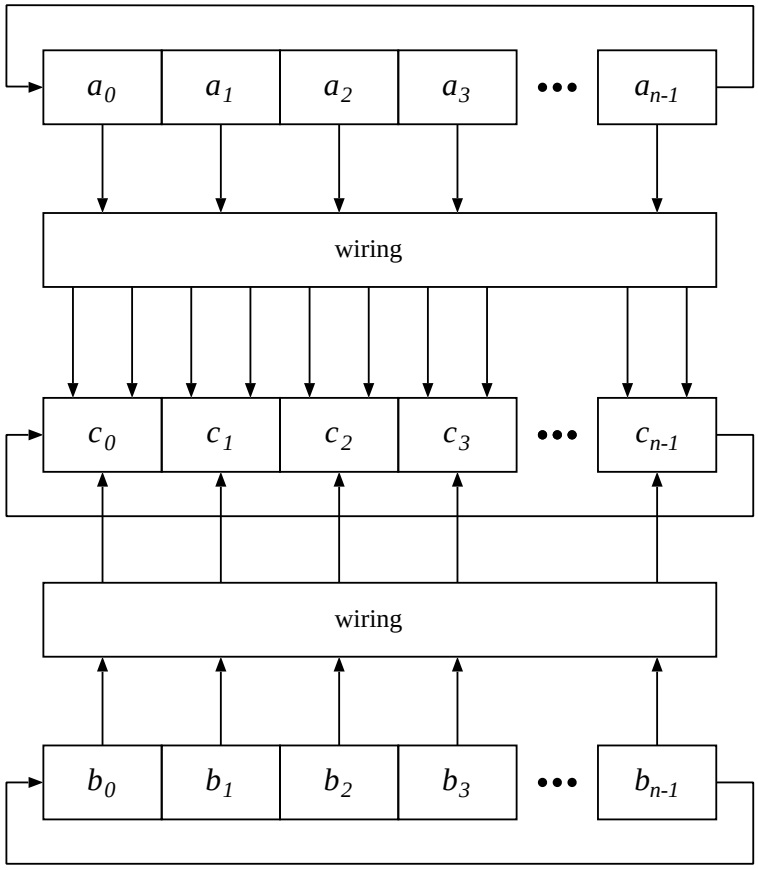


Figure 6.9: F_{2^n} multiplier circuit (wiring boxes are illustrated in Figure 6.10) for the case $n = 5$.

In each cycle, the A, B and C registers are rotated as shown in Figure 6.9. The result being that after n cycles, the contents of register C are the desired product of the A and B inputs [AMOV91]. It should be noted that an optimal normal basis reduces the number of interconnections and fanout of signals in the multiplier to the minimum possible, resulting in reduced area and increased speed over a non–optimal normal basis. In fact, the maximum fanout for a_i in Figure 6.9 is 4 [AMOV91].

Figure 6.10 shows an example of multiplier when $n = 5$. By Table 3.3 and

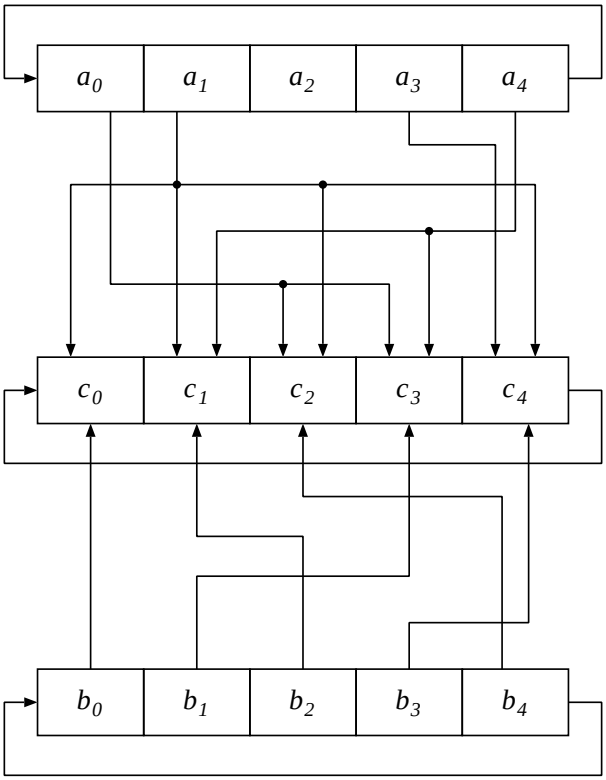


Figure 6.10: ONB Multiplier with $n = 5$.

Equation 6.2, the connections of cell c_k is defined by $F(k)$ below

$$\begin{aligned} F(0) &= b_0(a_1), \\ F(1) &= b_2(a_1 + a_4), \\ F(2) &= b_4(a_0 + a_1), \\ F(3) &= b_1(a_4 + a_0), \\ F(4) &= b_3(a_1 + a_3). \end{aligned}$$

Hence,

$$\begin{aligned} c_k &= b_k(a_{k+1}) + b_{k+1}(a_k + a_{k+3}) + b_{k+2}(a_{k+3} + a_{k+4}) \\ &\quad + b_{k+3}(a_{k+1} + a_{k+2}) + b_{k+4}(a_{k+2} + a_{k+4}) \end{aligned}$$

where $0 \leq k \leq 4$ and all subscripts are modulo 5. The result is the same as Equation 3.9.

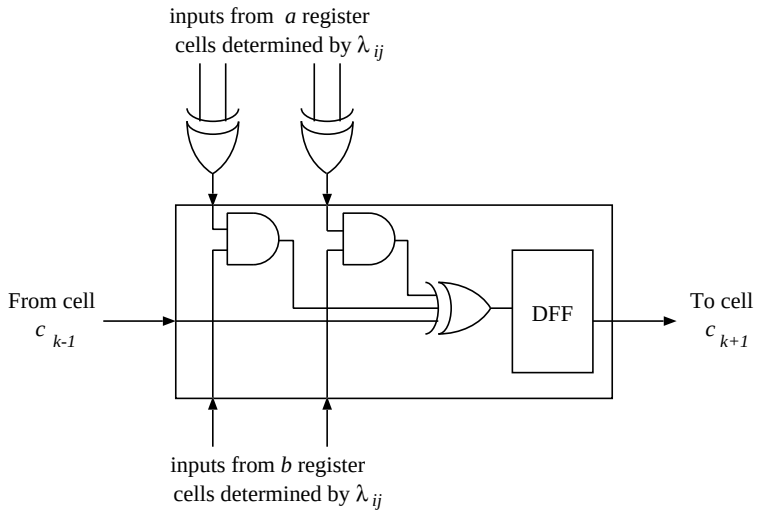


Figure 6.11: Multiplier element of a parallel multiplier.

The ALU can be easily parallelized. To increase the parallelism by a factor p, the multiplier logic can be duplicated p times. As an example, Figure 6.11 shows the case for $p = 2$. The multiplication requires $\lceil \frac{n}{p} \rceil$ cycles of parallel multiplication plus an extra cycle to form a final rotation. Therefore the total number of cycles required for a multiplication is $\lceil \frac{n}{p} \rceil + 1$.

6.4.2 Register File

A $16 \times n$-bit dual-port synchronous register file is constructed from the 16×1-bit distributed RAM feature of the Xilinx Virtex series logic cell (see Section 6.3). This gives an eight-fold reduction in resources over RAMs based on latches. It is used for storing the parameters, for example, c, P in Equation 5.8 and a_2, a_6 in Equation 5.4. These parameters can be downloaded to the register file by the bitstream reconfiguration discussed in Section 6.7.

Figure 6.12 shows the structure of the register file. In order to give out two n-bit data at the same time, the register file contains two sets of $16 \times n$-bit distributed RAMs. For data consistency, both of them share the same input data bus and write address bus. However, read address buses are different so that contents of different registers can be accessed simultaneously.

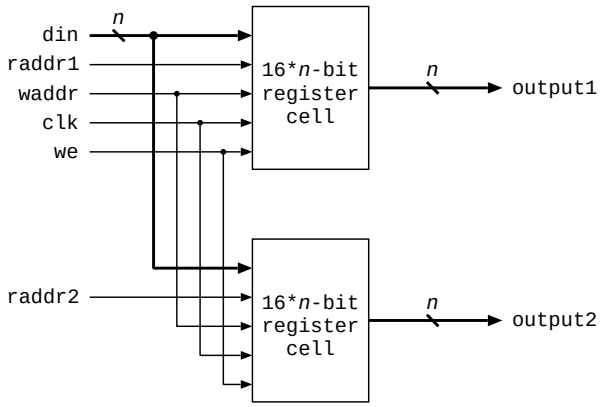


Figure 6.12: $16 \times n$-bit Register file.

6.4.3 Microcode

The ALU plus register file form a F_{2^n} processor similar to previous designs [AMV93]. However, for performing elliptic curve cryptography, higher level elliptic curve multiplications of Section 5.4.4 are required. This could be implemented as a finite state machine [Ros98b] or in *microcode*. The implementation described in this thesis opted for a microcoded approach which has the following advantages in a FPGA implementation:

1. the microcode is stored in Xilinx Virtex BlockRAMs and do not use logic resources of the FPGA (as explained in Section 6.3). The microcode sequencer in this design is very simple and has a small overhead. Since the critical path of the ECP is in the datapath, the microcode implementation of the control does not affect the overall performance

2. the microcode can be changed without requiring re-compilation of the elliptic curve processor. The microcode can be altered directly by the host PC program

3. algorithmic optimizations to the processor can be performed entirely in microcode

4. a microcoded description is a higher level abstraction than a finite state

Operation	Clock Cycles
NOP	1
XOR	1
Rotate left, ROTL	1
Shift right, SHFR	1
Field Multiplication, MUL	$\lceil \frac{n}{p} \rceil + 1$
Transfer register value, TFR	1
Jump Instructions	1
JKZ, JCZ, JMP	

Table 6.1: Clock cycles required for each instruction using a p-way parallel ALU.

machine and hence easier to develop and debug.

The instruction set of the processor is shown in Table 6.1. Apart from instructions which directly control the ALU, there are three types of jump instructions: JMP – jump unconditionally, JKZ – jump if the least significant bit of K register is zero and JCZ – jump if the $COUNT$ register is zero. The K register stores the parameter c of Equation 5.8. If the least significant bit of K is set, then an ESUM (Section 5.4.3) is performed. Otherwise, EDBL (Section 5.4.3) will be performed and K register will be rotated by 1 bit. In addition, the $COUNT$ register is initialized to all '1's. It is used to keep track of the multiplication process. It is rotated along with K register and is used to indicate the end of a multiplication. Note that the K and $COUNT$ registers are in the register file.

Each instruction is 16 bits in width and the format of instructions is shown in Figure 6.13. The instruction width was dictated by the choice of a 12-bit address (the current implementation has < 512 lines of microcode and hence fits into 9 bits). Since there are 9 different operations which are encoded into 4 bits, 16 bits are required. Most instructions accept a source register and a destination register in `operand1` and `operand2` respectively. The jump instructions have a 12-bit jump address.

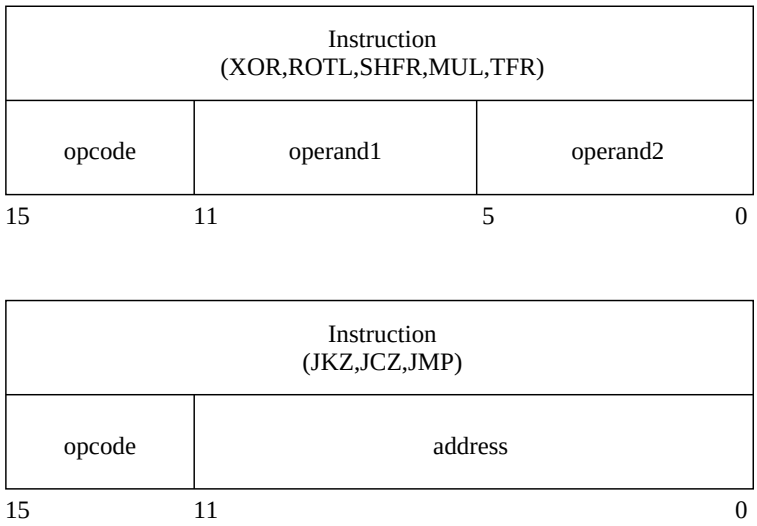


Figure 6.13: Instruction format.

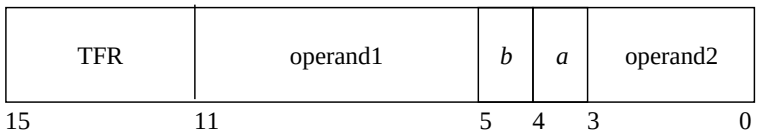


Figure 6.14: TFR instruction format.

There are three special registers A, B and C which are involved in multiplication of a field element (as shown in Figure 6.9). Before multiplication, the operands are transferred from the register file into A and B using a TFR instruction (Figure 6.1). When bit 5 of a TFR instruction set, `operand1` will be transferred to register B. Similarly, if bit 4 set, the contents of register `operand1` will be transferred to register A. Note that bits 0-3 of the TFR instruction to the multiplier are unused. The NOP instruction was used in earlier versions of the processor to fill a branch delay slot caused by pipelining of the fetch and execute parts of the microcode sequencer. The final implementation does not have pipelining and these NOP instructions could be removed. As will be shown in Table 7.4, the result of leaving the unnecessary NOP instructions in the microcode does not lead to a significant performance degradation.

The state transition diagram of the microcode sequencer is shown in Figure 6.15. It is implemented as a simple state machine.

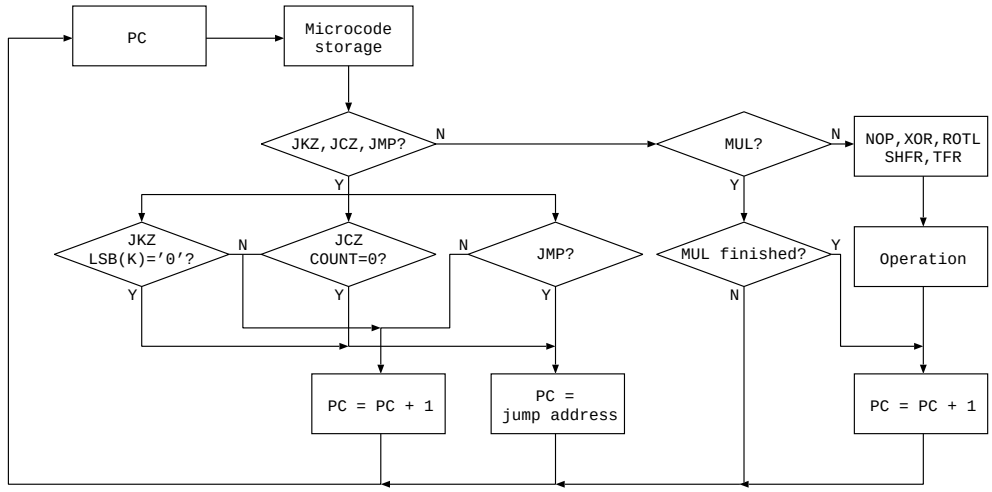


Figure 6.15: Microcode sequencer.

6.5 Parameterized Module Generator

The strength of a cryptosystems is determined by the size of the cryptographic key. In order to provide elliptic curve processors with different cryptographic strength to suit different requirements, a parameterized module generation program was used. The *parameterized module generator* can generate the VHDL description of elliptic curve processors for any n with an optimal normal basis. Hence this scheme advantageously uses the reconfigurable nature of the FPGA to add the flexibility of being able to choose arbitrary n (compared with fixed n of previous designs [AMV93]).

Besides generating the elliptic curve processors with different values of n, the module generator can produce processors with parallelized ALU as stated in Section 6.4.1. The module generator takes a parameter p which is the level of parallelism of the ALU. By varying p, different tradeoffs performance can be made. Therefore this can trade off between area and performance. The block diagram of parameterized module generator is shown in Figure 6.16.

The module generator is a program written in the Perl programming language [WCS96] which takes n and p as an input parameters and produces the VHDL code of the ECP as output. It first calculates the multiplication table λ

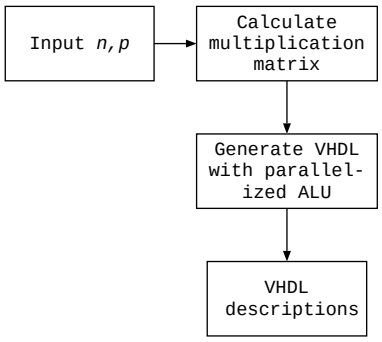


Figure 6.16: Parameterized module generator block diagram.

using the process described in Section 3.6.3. It then customises the VHDL code for the ALU based on the multiplication table and p, resulting in a complete VHDL description of the ECP. Perl is a language which supports long integer arithmetic which was helpful in performing the arithmetic required to generate the λ matrix of the field multiplier. Since the ECP design is synthesized from a behavioral VHDL description, it would trivial to port the design to other FPGA families (e.g. Altera) or an ASIC. Input of the curve parameters can be done via the bus interface instead of bitstream reconfiguration for better portability.

6.6 Microcode Toolkit

For the curve operations such as point multiplication on an elliptic curve are implemented as sequences of field operations which can be implemented in microcode. A toolkit was developed to facilitate the microcode development.

A microcode simulator and debugger was written to facilitate development of microcode programs. It supports all the instructions shown in Table 6.1. Moreover, breakpoints can be set so that the contents of the registers can be listed in the middle of the program for debugging. Single stepping is also supported.

A two pass symbolic assembler was also developed which takes symbolic

input and produces the binary microcode which can be downloaded to the processor's microcode store, BlockRAMs in Virtex. The assembler is written in the Perl programming language. Its strong string processing features helps in converting the symbolic assembly program into a binary form.

6.7 Initialization by Bitstream Reconfiguration

In order to perform an elliptic curve multiplication, the values of c and P in Equation 5.8 as well as the parameters of the curve must be downloaded to the processor. This is normally done by interfacing the register file to the host PC via a dual port memory. In this work, a bitstream modification technique was used to modify the contents of RAMs in which the parameters are stored. Since the register file consists of the distributed RAMs in Virtex (Section 6.3), they are placed in arbitrary locations.

In order to modify the content of the registers, the locations of those CLBs which form the register file are determined from the circuit description file. The NCD file is an circuit description representing the design mapped to components in the FPGA. This file is, originally, in binary format. However, it can be converted into a human readable format by the tools provided by Xilinx. Therefore the information about the locations of register file can be extracted.

A software program was written which takes as inputs the human readable format of NCD file of the design, the register's initial content file and the bitstream of the design. It modifies the register contents in the bitstream accordingly and recomputes the cyclic redundancy code (CRC) of the bitstream. The resulting bitstream can be downloaded to a Virtex FPGA as usual. The advantage of this technique is that circuitry to download the parameters from the host PC to the FPGA is avoided, hence reducing the overall area and increasing the speed of the processor.

Since only a small portion of the bitstream needs to be modified to change

the parameters, but the entire bitstream must be downloaded to affect the change, runtime reconfiguration techniques [BDH+97, XA00] may be a more efficient approach.

6.8 Summary

In this chapter began with introducing the CAD tools and the design flow of developing the elliptic curve processor. It was followed by an overview of hardware platform used in the work described in this thesis. The implementations of the components of the processor were also detailed.

The ECP was developed on a reconfigurable hardware computing platform, Wildstar, which consists of Xilinx Virtex FPGAs. The device can communicate with the host system so that data can be transferred between them through the device interface.

A parameterized module generator was developed which can generate elliptic curve processors with different values of n with different levels of parallelism. The design of ALU was based on the cyclic relationship between the operands and the product (see Chapter 3). The register file of the processor was constructed by the distributed RAM feature of Virtex which can reduce the demanded resources. The ECP is programmed by microcode which can be downloaded from the host computer and are stored in the BlockRAM feature of Virtex. A microcode toolkit including simulator and debugger, was developed to facilitate the microcode development. A script was also written to modify the initial contents of the registers therefore no recompiling of the processor is needed for different curve parameters.

Chapter 7

Results

7.1 Introduction

In this chapter, the performance of elliptic curve processor with serial multiplier is first presented. Then a comparison of the relative merits of affine and projective coordinates is made. Finally, the results using a parallel multiplier are given.

7.2 Elliptic Curve Processor with Serial Multiplier ($p = 1$)

VHDL code for the elliptic curve processor was generated using the parameterized module generator (discussed in Section 6.5) for different values of n with an optimal normal basis (security considerations were discussed in Section 2.4.4). Synthesis and implementation were performed using Synopsys FPGA Express 3.4 and Xilinx Foundation 3.2i respectively. Table 7.1 shows the resource utilization and maximum clock rate reported by the Xilinx tools for designs with different n. As can be seen in Figure 7.1, resource requirements are linear with n. Therefore by projecting the graph, the largest n that can be implemented on XCV1000–6 is 1100. The size of the microcode is less than 512 16-bit words and does not significantly change for different n.

n	# of slices	Reported freq (MHz)
113	1410	31
155	1868	30
173	2148	28
281	3315	26
371	4247	22
473	5264	18

Table 7.1: Resource utilization and maximum clock rate for different n on a Xilinx XCV1000–6. The Xilinx XCV1000–6 contains 12288 slices (6144 CLBs).

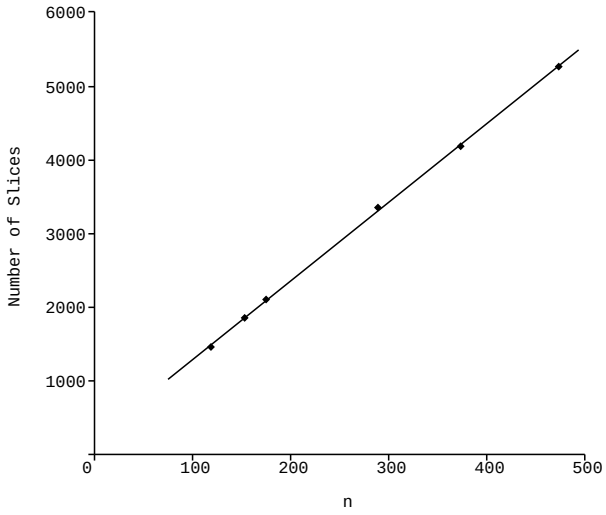


Figure 7.1: Number of slices used for different n.

The ECP was successfully tested on a Wildstar board (see Section 6.3), the microcode being downloaded to the BlockRAMs by the host PC and the parameters being downloaded to RAMs by bitstream reconfiguration.

Examples of curve multiplication on the work are shown in Appendix B. Curve multiplications with $n = 113$, 155, 173, 281, 371 and 473 were performed and calculation results (public key Q in Equation 5.8) were compared with the ECC software package from [Ros98a]. The results were also verified by the cryptographic algorithm discussed in Section 5.6. Messages were generated randomly, encrypted using the public keys and then decrypted using the private key to verify the processor.

n	SW time (ms)		HW time (ms)	Speed-up		Throughput kbits/s
	E4500	PIII-866		E4500	PIII-866	
113	27.6	16	4.3	6	4	13
155	63.2	40	8.3	8	5	9
173	86.6	54	11.1	8	5	8
281	281.8	196.2	29.9	9	7	5
371	746.3	442.6	63.1	12	7	3
473	1490.9	873.2	126.2	12	7	2

Table 7.2: Execution time for elliptic curve multiplication (projective coordinates) and comparison with a software implementation.

The execution time of the processor was compared with that of an optimized software implementation of an optimal normal basis elliptic curve package [Ros98a] running on a SUN Enterprise E4500 with UltraSPARC-II 400 MHz processors and a PC with Pentium-III 866 MHz processor. The results and the throughput are presented in Table 7.2. It can be seen that the elliptic processor is approximately 6 to 12 times faster than the software implementation on E4500 and 4 to 7 times faster on Pentium-III. PC to FPGA communications overhead is negligable.

7.3 Projective verses Affine Coordinates

The projective and affine implementations share the same hardware design with different microcodes. The total number of cycles required for an elliptic curve multiplication for various n are given in Table 7.3, where assuming that the c of Equation 5.8 is a n-bit binary number with half the number of bits set. The execution time required for an elliptic curve multiplication at the maximum frequency is shown in Table 7.3. Note that these figures include the time for host processor interfacing. The implementations using projective coordinates are always faster than using affine coordinates because field inversions are extremely expensive (Section 5.4.4).

n	# of cycles (affine)	# of cycles (proj.)	HW time affine (ms)	HW time proj.(ms)	$P:A$
113	148581	134484	4.8	4.3	0.9
155	324717	249879	10.8	8.3	0.77
173	402926	310043	14.4	11.1	0.77
281	1021814	784155	39.1	29.9	0.76
371	2261605	1407005	101.4	63.1	0.62
473	3657560	2267501	203.5	126.2	0.62

Table 7.3: Execution time for projective and affine coordinate implementations of elliptic curve multiplication.

7.4 Elliptic Curve Processor with Parallel Multiplier ($p > 1$)

The dynamic instruction frequencies for a curve multiplication using different n are shown in Table 7.4. From the table, it can be clearly seen that the bottleneck is in field multiplication (MUL) which accounts for approximately 90% of the execution time.

The time taken for a curve multiplication using a parallel multiplier is given in Table 7.5 ($n = 113$ and 473), showing that a parallel implementation has an 85% increase in area leading to 530% improvement in speed when $n = 113$ while an 94% increase in area gives 994% improvement in speed when $n = 473$. Figure 7.2 is a plot of normalized performance verses the degree of parallelism p. As can be seen from the figure, the execution time improves as parallelism is increased and tradeoffs between area and performance can be easily made. Improvement is approximately linear for $p \leq 6$ when $n = 113$ and $p \leq 10$ when $n = 473$.

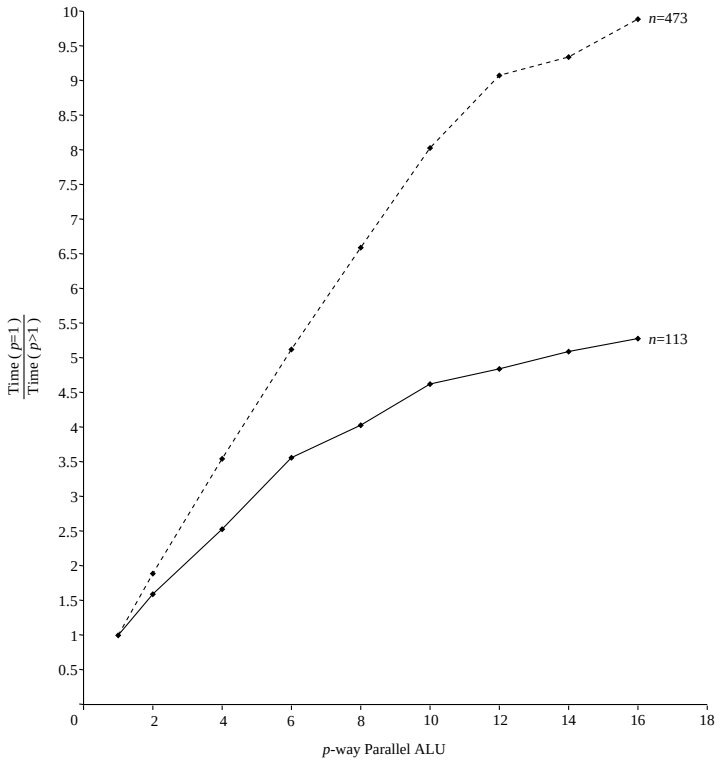


Figure 7.2: Normalized execution time for one curve multiplication using a p-way parallel ALU.

7.5 Summary

In this chapter, the performance of the elliptic curve processor in affine and projective coordinates and with serial and parallel multipliers were presented. The ECP had 4–12 times speed up over the software implementation. Projective coordinates was shown faster than affine coordinates.

A dynamic instruction frequency analysis showed that multiplication accounts for up to 97% of instructions in the ECP. A parallel multiplier improved the performance of the processor by reducing the number of cycles for multiplication. For $n = 113$ and 473, the parallel multiplier up to $p = 16$ was tested, the speedup being linear up to $p = 6$ and 10 respectively.

	Projective			Affine		
n	113	155	173	113	155	173
NOP	291 (0.22%)	391 (0.16%)	444 (0.14%)	291 (0.2%)	391 (0.12%)	444 (0.11%)
XOR	616 (0.46%)	847 (0.34%)	946 (0.31%)	784 (0.53%)	1078 (0.33%)	1204 (0.3%)
MUL	128820 (95.79%)	242112 (96.89%)	301368 (97.2%)	127680 (85.93%)	288288 (88.78%)	359136 (89.13%)
ROTL	673 (0.5%)	925 (0.37%)	1033 (0.33%)	12825 (8.63%)	24102 (7.42%)	30015 (7.45%)
TFR	3625 (2.7%)	4972 (1.99%)	5548 (1.79%)	5432 (3.66%)	7931 (2.44%)	8858 (2.2%)
JKZ	113 (0.08%)	155 (0.06%)	173 (0.06%)	113 (0.08%)	155 (0.05%)	173 (0.04%)
JCZ	111 (0.08%)	153 (0.06%)	171 (0.06%)	111 (0.07%)	153 (0.05%)	171 (0.04%)
JMP	111 (0.08%)	153 (0.06%)	171 (0.06%)	111 (0.07%)	153 (0.05%)	171 (0.04%)
SHFR	123 (0.09%)	170 (0.07%)	188 (0.06%)	1233 (0.83%)	2465 (0.76%)	2753 (0.68%)
Total	134484 (100%)	249879 (100%)	310043 (100%)	148581 (100%)	324717 (100%)	402926 (100%)

Table 7.4: Dynamic instruction counts (dynamic instruction frequencies in parentheses) for an elliptic curve multiplication using different n.

	$n = 113$			$n = 473$		
p-way	slices	cycles	Time (ms)	slices	cycles	Time (ms)
1	1410	134484	4.3	5264	2267501	126.2
2	1860	71204	2.6	6928	1150278	69.2
4	1970	39564	1.7	7396	591666	35.7
6	2076	28264	1.2	7872	402306	24.5
8	2182	23744	1.06	8340	312360	19.1
10	2300	20354	0.93	8799	253246	15.7
12	2434	18094	0.89	9288	217680	13.8
14	2515	16964	0.84	9752	192602	13.5
16	2614	15833	0.81	10229	170340	12.7

Table 7.5: p-way parallel ALU resource utilization and performance for projective coordinates ($n = 113$ and 473).

Chapter 8

Conclusion

In this work, a novel architecture for an elliptic curve processor over F_{2^n} using an optimal normal basis was developed. The processor can perform a scalar multiplication on elliptic curve, the central operation of elliptic curve cryptography. The main results that were obtained are as follows:

- the curve multiplication over F_{2^n} with n = 113, 155, 173 , 281, 371 and 473 was successfully tested on the hardware with reported frequencies 31, 30, 28, 26, 22 and 18 MHz respectively
- curve multiplication in affine and projective coordinates were implemented and tested. Experiments showed that projective implementation had 10% (n = 113), 23–24% (n = 155, 173, 281) and 38% (n = 473) improvement over affine.
- projective implementation with serial multiplier gave an 4–12 times improvement over an optimized software implementation
- the work with 16-way parallel multiplier (n = 113 and 473) gave 5 and 10 times improvement over the serial multiplier and it gave linear improvement when degree of parallelism $\leq$ 6 and 10 respectively.

The processor used the reconfigurable nature of FPGA devices to achieve flexibility not attainable in a traditional ASIC. In particular, it was generated

by a parameterized module generator which can generate field processors using an optimal normal basis for arbitrary n using a multiplier with different speed/area tradeoffs. A microcoded approach was used to implement a curve processor on top of the field processor. Different algorithms can be implemented by changing the microcode and the design has lower I/O requirements than the basic field processor. The microcoded processor has the advantage that a high bandwidth interface is not required compared with previous elliptic curve processors which only implemented field operations.

With the advantages of smaller key sizes, lower memory and computational requirements than other public key cryptosystems, ECC lends to sending information securely over the Internet where bandwidth and processing capabilities are limited.

8.1 Recommendations for Future Research

The thesis described an elliptic curve cryptographic processor performing elliptic curve scalar multiplication. This is an active area of this research and new algorithms have been developed with improved efficiency.

An improved method for computing curve multiplication (cP) using a *non–adjacent form* (NAF) [BSS99] has been reported. The expected length of the NAF is about one–third of the original binary form [BSS99], therefore the multiplication process can be made three times faster on average although conversion to NAF form is required.

Another improved algorithm for computing repeated doubling (2^iP) can further improve the performance [GP97]. It can compute 2^iP for an elliptic curve defined over F_{2^n} with only one inversion. It provides a faster computation for 2^iP than consecutive doublings if the cost of inversion is 2.5 times or more than multiplication (which is the case for the ECP described in the work).

The elliptic curve processor described in this thesis requires the parameters to be loaded into the registers before the calculation started. More functions could be handled by the processor. For example, generating random elliptic curves and elliptic curve point counting can be implemented on chip. This approach would improve the resistance of the processor to probing attacks [And01] and reduces the software support required by the processor.

From the design point of view, FPGA devices are very suitable platforms for implementing cryptographic hardware. With low cost, high speed and feasibility of upgrading in the field, FPGAs will be used more and more for cryptographic systems on embedded devices. The work described in this thesis is just one example of such an application.

Bibliography

[ABV89] D. Ash, I. Blake, and S. Vanstone. Low complexity normal bases, 1989.

[AMOV91] G. B. Agnew, R. C. Mullin, I. M. Onyszchuk, and S. A. Vanstone. An implementation for a fast public-key cryptosystem. *Journal of Cryptology*, 3:63–79, 1991.

[AMV93] G. B. Agnew, R. C. Mullin, and S. A. Vanstone. An implementation of elliptic curve cryptosystems over $F_{2^{155}}$. *IEEE Transactions on Selected Areas in Communications*, 11:804–813, 1993.

[And01] Ross J. Anderson. *Security Engineering: A Guide to Building Dependable Distributed Systems*. John Wiley & Sons, 2001.

[Ann99] Annapolis Micro Systems, Inc. *Wildstar Reference Manual*, 1999. Revision 3.3.

[BDH+97] J. Burns, A. Donlin, J. Hogg, S. Singh, and M. Wit. A dynamic reconfiguration run-time system. In *Proceedings of Field-Programmable Custom Computing Machines (FCCM'00)*, pages 66–75, 1997.

[Bih97] Eli Biham. A fast new DES implementation in software. In *Fast Software Encryption*, pages 260–272, 1997.

[BSS99] I. Blake, G. Seroussi, and N. Smart. *Elliptic Curves in Cryptography*. Cambridge University Press, 1999.

[CCI88] CCITT. Recommendation $X.509$: The directory authentication framework, 1988.

[cer] Certicom corporation. `http://www.certicom.com/`.

[DH76] W. Diffie and M. E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, IT-22:644–654, 1976.

[GP97] J. Guajardo and C. Paar. Efficient algorithms for elliptic curve cryptosystems. In *Advances in Cryptology - Crypto '97*, pages 342–356. Springer-Verlag, 1997. Lecture Notes in Computer Science Volume 1294.

[Gro01] J. Großschädl. A bit-serial unified multiplier architecture for finite fields $GF(p)$ and $GF(2^m)$. In *Workshop on Cryptographic Hardware and Embedded Systems (CHES '01)*, 2001.

[Hel71] Y. Hellegouarch. Points d'ordre fini sur les courbes elliptiques. In *Acad. Sci. Paris S'er. A-B 273*, pages A540–A543, 1971.

[HR82] M. Hellman and J. Reyneri. Fast computation of discrete logarithms in $GF(q)$. In *Advances in Cryptology: Proceedings of Crypto '82*, pages 3–14. Plenum Publishing, 1982.

[ieee] *IEEE Standard 1363, IEEE Standard Specifications For Public-Key Cryptography.*

[IT88] T. Itoh and S. Tsujii. A fast algorithm for computing multiplicative inverses in $GF(2^m)$ using normal bases. *Info. and Comput.*, 78(3):171–177, 1988.

[Ken93] S. Kent. *RFC 1422: Privacy Enhancement for Internet Electronic Mail: Part II: Certificate-Based Key Management.* Internet Activities Board, February 1993.

[Kob87a] N. Koblitz. *A Course in Number Theory and Cryptography.* Springer-Verlag, 1987.

[Kob87b] N. Koblitz. Elliptic curve cryptosystems. *Mathematics of Computation*, 48:203–209, 1987.

[Kob91] N. Koblitz. CM-curves with good cryptographic properties. In *Advances in Cryptology - Crypto '91*, pages 279–287. Springer-Verlag, 1991. Lecture Notes in Computer Science Volume 576.

[Kob93] N. Koblitz. *Introduction to Elliptic Curves and Modular Forms.* Springer-Verlag, 1993.

[LN94] R. Lidl and H. Niederreiter. *Introduction to Finite Fields and Their Applications.* Cambridge University Press, 1994.

[LV00] Arjen K. Lenstra and Eric R. Verheul. Selecting cryptographic key sizes. In *Public Key Cryptography*, pages 446–465, 2000.

[Men93] A. J. Menezes. *Elliptic Curve Public Key Cryptosystems.* Kluwer Academic Publishers, 1993.

[Mil86] V. S. Miller. Use of elliptic curves in cryptography. In *Advances in Cryptology - Crypto '85*, pages 417–428. Springer-Verlag, 1986. Lecture Notes in Computer Science Volume 218.

[MOV99] A. J. Menezes, P.C. van Oorschot, and S. A. Vanstone. *Handbook of Applied Cryptography.* CRC Press, 1999.

[MOVW89] R. C. Mullin, I. M. Onyszchuk, S. A. Vanstone, and R. M. Wilson. Optimal normal bases in $GF(p^n)$. *Discrete Applied Mathematics*, 22:149–161, 1988/89.

[nua] NUA Internet Surveys, Year in review. `http://www.nua.ie/surveys/`.

[Odl84] A. M. Odlyzko. Discrete logarithms in finite fields and their cryptographic significance. In *Advances in Cryptology: Proceedings of EuroCrypt '84*, pages 224–316. Springer-Verlag, 1984. Lecture Notes in Computer Science Volume 209.

[OP99] G. Orlando and C. Paar. A super-serial Galois field multiplier for FPGAs and its application to public key algorithms. In *Proceedings of the IEEE Symposium on Field–programmable custom computing machines (FCCM'99)*, pages 232–239, 1999.

[OP01] G. Orlando and C. Paar. A scalable $GF(p)$ elliptic curve processor architecture for programmable hardware. In *Workshop on Cryptographic Hardware and Embedded Systems (CHES '01)*, 2001.

[OW99] P. van Oorschot and M. Wiener. Parallel collision search with cryptanalytic applications. *Journal of Cryptology*, 12:1–28, 1999.

[Pat00] C. Patterson. High performance DES encryption in Virtex™ FPGAs using JBits™. In *Proceedings of Field-Programmable Custom Computing Machines (FCCM'00)*, pages 113–121, 2000.

[PH78] S. Pohlig and M. Hellman. An improved algorithm for computing logarithms over $GF(p)$ and its cryptographic significance. *IEEE Transactions on Information Theory*, 24(1):106–110, 1978.

[PID92] J. Stern P. Ivey, S. Walker and S. Davidson. An ultra–high speed public key encryption processor. In *Proceedings of the IEEE Custom Integrated Circuits Conference*, pages 19.6.1–19.6.4, 1992.

[Pol74] J. Pollard. Theorems on factorization and primality testing. In *Proceedings of the Cambridge Philosophical Society*, volume 76, pages 521–528, 1974.

[Pol75] J. M. Pollard. A Monte Carlo method for factorization. *BIT*, 15(3):331–334, 1975.

[Rie87] H. Riesel. *Prime Numbers and Computer Methods for Factorization, 2nd Ed.* Birkhauser, 1987.

[Ros98a] M. Rosing. *Implementing Elliptic Curve Cryptography.* Manning, 1998.

[Ros98b] M. Rosner. *Elliptic Curve Cryptosystems on Reconfigurable Hardware.* Master's Thesis, Worcester Polytechnic Institute, Worcester, USA, 1998.

[RS97] K. Rubin and A. Silverberg. Wiles' proof of fermat's last theorem, 1997.

[rsaa] RSA Laboratories. `http://www.rsa.com/`.

[rsab] RSA Laboratories' Frequently Asked Questions About Today's Cryptography, Version 4.1.

[RSA78] R. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21:120–126, 1978.

[Sil85] J. H. Silverman. *The Arithmetic of Elliptic Curves.* Springer-Verlag, 1985.

[Ska96] K. Skahill. *VHDL for Programmable Logic*. Addison-Wesley, 1996.

[SOOS95] R. Schroeppel, H. Orman, S. O'Malley, and O. Spatschek. Fast key exchange with elliptic curve systems. In *Advances in Cryptology - Crypto '95*, pages 43–56. Springer-Verlag, 1995. Lecture Notes in Computer Science Volume 963.

[ST92] J. H. Silverman and J. Tate. *Rational Points on Elliptic Curves*. Springer-Verlag, 1992.

[SV93] M. Shand and J. E. Vuillemin. Fast implementations of RSA cryptography. In *Proceedings of the 11th IEEE Symposium on Computer Arithmetic*, pages 252–259. IEEE Computer Society Press, 1993.

[WCS96] L. Wall, T. Christiansen, and R. L. Schwartz. *Programming Perl*. O'Reilly, 2nd edition, 1996.

[XA00] B. Xu and D. H. Albonesi. Runtime reconfiguration techniques for efficient general purpose computation. *IEEE Design & Test of Computers, Special Issue on Configurable Computing*, pages 42–52, January-March 2000.

[Xil00] Xilinx. *Virtex 2.5V field programmable gate arrays*, 2000.

Appendix A

Elliptic Curves in Characteristics 2 and 3

A.1 Introduction

Elliptic curve cryptography is based on calculation of elliptic curves over finite fields. The equations of elliptic curves over different fields are different. In this chapter, it shows the derivations of the elliptic curves of finite fields with characteristic 2 and 3.

A.2 Derivations

Elliptic curve defined over finite field K, $E(K)$, is given by *Weierstrass equation*,

$$y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6 \tag{A.1}$$

where $a_1, a_2, a_3, a_4, a_6 \in F_{2^n}$.

Two elliptic curves E_1/K and E_2/K

$$E_1 \quad : \quad y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6 \tag{A.2}$$

$$E_2 \quad : \quad y^2 + a'_1xy + a'_3y = x^3 + a'_2x^2 + a'_4x + a'_6 \tag{A.3}$$

are said to be isomorphic over K, denoted $E_1/K \cong E_2/K$, if and only if there exists $u, r, s, t \in K$, $u \neq 0$, such that (x, y) on E_1/K is mapped to $(u^2x + r, u^3y + u^2sx + t)$ on E_2/K [Men93]. Therefore the change of variables

$$(x, y) \longrightarrow (u^2x + r, u^3y + u^2sx + t) \tag{A.4}$$

transforms Equation A.2 to Equation A.3. The transformation A.4 is known as an admissible change of variables.

Since $E_1 \cong E_2$ over K, then the admissible change of variables A.4 yields the set of equations [Men93]:

$$\left.\begin{aligned} ua_1' &= a_1 + 2s \\ u^2a_2' &= a_2 - sa_1 + 3r - s^2 \\ u^3a_3' &= a_3 + ra_1 + 2t \\ u^4a_4' &= a_4 - sa_3 + 2ra_2 - (t + rs)a_1 + 3r^2 - 2st \\ u^6a_6' &= a_6 + ra_4 + r^2a_2 + r^3 - ta_3 - t^2 - rta_1. \end{aligned}\right\} \tag{A.5}$$

Therefore E_1/K and E_2/K are isomorphic over K if and only if there exists $u, r, s, t \in K$, $u \neq 0$, such that satisfy Equations A.5.

A.3 Elliptic Curves over Finite Fields of Characteristic $\neq$ 2,3

For a field F_p, p (a prime) is the characteristic of the field. If the elliptic curve defined over field K of characteristic $\neq 2$, then Equation A.1 can be simplified by completing the square and thus replace y by $\frac{1}{2}(y - a_1x - a_3)$ [Sil85]. That is applying the admissible change of variables

$$(x, y) \longrightarrow \left(x, \frac{1}{2}(y - a_1x - a_3)\right)$$

transforming E/K to

$$E'/K : y^2 = 4x^3 + b_2x^2 + 2b_4x + b_6$$

where

$$\begin{aligned} b_2 &= a_1^2 + 4a_2 \\ b_4 &= 2a_4 + a_1a_3 \\ b_6 &= a_3^2 + 4a_6. \end{aligned}$$

Moreover define

$$\begin{aligned} b_8 &= a_1^2a_6 + 4a_2a_6 - a_1a_3a_4 + a_2a_3^2 - a_4^2 \\ c_4 &= b_2^2 - 24b_4 \\ c_6 &= -b_2^3 + 36b_2b_4 - 216b_6 \\ \Delta &= -b_2^2b_8 - 8b_4^3 - 27b_6^2 + 9b_2b_4b_6 \qquad \text{(A.6)} \\ j &= \frac{c_4^3}{\Delta}. \qquad \text{(A.7)} \end{aligned}$$

Δ of Equation A.6 and j of Equation A.7 are called the discriminant and j-invariant, $j(E)$, of the elliptic curve. For elliptic curves in the same isomorphism class, their values of j-invariant would be the same. Since j-invariant is defined by inverse of Δ, it is real if $\Delta \neq 0$. Therefore, an elliptic curve is said to be non–singular (there exists isomorphic curve) if $\Delta \neq 0$.

Further if elliptic curve over K of characteristic $\neq$ 2,3, then the admissible change of variables [Men93]

$$(x, y) \longrightarrow \left(\frac{x - 3b_2}{36}, \frac{y}{108}\right)$$

further transforms E' to

$$E''/K : y^2 = x^3 - 27c_4x - 54c_6.$$

Therefore for field K of characteristic $\neq$ 2,3, an elliptic curve over K is the set of points (x, y) satisfying the equation

$$y^2 = x^3 + a_4x + a_6 \qquad \text{(A.8)}$$

together with $\mathcal{O}$, where $x, y \in K$ [Sil85]. Moreover the associated quantities are

$$\Delta = -16\left(4a_4^3 + 27a_6^2\right) \tag{A.9}$$

$$j(E) = \frac{-1728(4a_4)^3}{\Delta}. \tag{A.10}$$

A.4 Elliptic Curves over Finite Fields of Characteristic = 2

If K of characteristic 2, then j-invariant of E/K is [Men93]

$$\frac{a_1}{\Delta}.$$

If $j(E) \neq 0$, then the admissible change of variables [Men93]

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

transforms E to the curve

$$E_1/K : y^2 + xy = x^3 + a_2 x^2 + a_6. \tag{A.11}$$

For E_1, $\Delta = a_6$ and $j(E_1) = \frac{1}{a_6}$.

If $j(E) = 0$, then the admissible change of variables

$$(x, y) \longrightarrow (x + a_2, y)$$

transforms E to the curve

$$E_2/K : y^2 + a_3 y = x^3 + a_4 x + a_6. \tag{A.12}$$

For E_2, $\Delta = a_3^4$ and $j(E_2) = 0$.

Appendix B

Examples of Curve Multiplication

B.1 Introduction

In this chapter, examples of curve multiplication performed on the work are shown. For each example, the parameters (a_2, a_6 in Equation 5.4) and the private key (c in Equation 5.8) were randomly generated. The base point (P in Equation 5.8) was found according to the method described in Section 5.5. Numbers are presented in hexadecimal form with high order digits on the left hand side.

B.2 Numerical Results

Example 1: n=113 bits

```
a2 :   1ffff ffffffff ffffffff ffffffff
a6 :   1ffff ffffffff ffffffff ffffffff
Base point P :
x :    16c80 3abc107c 188da158 67fa62d4
y :     de1e 607f7da5 8e4a1858 a8ba7c60
private key c :
c :     698a 82651d65 5d7a9d36 7d395bac
public key Q:
x :     4f7b 1ef82cb4 30f49db9 eaaaa5c9
y :    1070c 4970bfb7 d5249488 09e2de55
```

Example 2: n=113 bits

```
a2 :    ffff ffffffff fffdbf91 af6dea73
a6 :    400e 1a3faf38 66484043 011647ac
Base point P :
x :     cc73 99a652ba df30a35f 2adb772c
y :     9d08 3a0cf2d2 a3e11965 510c92f5
private key c :
c :     6f18 9f251db9 04892ca7 18480fd5
public key Q:
x :      985 0e2ca6ce 1f30a1ab 49b8d839
y :     bdfc ae9a446c 93a9c55a 4126f632
```

Example 3: n=113 bits

```
a2 :   1e8b5 5472af02 c70d8ce0 fcc01e63
a6 :    400e 1a3faf38 66484043 011647ac
Base point P :
x :    1c817 0844ae3a fe571aa8 fed48497
y :    1b87b fa893d86 c50395d1 ffa15a9d
private key c :
c :     8e1e 55dcdf6f 0e94e98b eca355e2
public key Q :
x :    1847e b0a88f63 22a214f6 f3476329
y :    1fe8a b79dd95e 658e0ca2 c00f2670
```

Example 4: n=155 bits

```
a2 : 7ffffff ffffffff ffffffff ffffffff ffffffff
a6 : 7ffffff ffffffff ffffffff ffffffff ffffffff
Base point P :
x :  4b2c670 a207432f 050cba69 ef6f260d 7403b9bd
y :  1c06f8e a6fb07f1 5d8ed4eb ab829d8a 230e5bb3
private key c :
c :  4b22557 e7a21ca7 c2569ce0 36c38148 6246725a
public key Q :
x :  3f601e7 3f983025 7e6bb129 98cb4606 7e2bd780
y :  2cc9915 7a2c5f40 45ed81f1 67f2cb1a 74d9e54c
```

Example 5: n=155 bits

a2 : 1999999 9999998e dbbbc76b 2af2dfd7 cd192e17

a6 : fffe202 97eeade8 751a888b dbb1daec c4bfca37

Base point P :

x : 4fceb80 c9954b5f cb803e0b 2c730d92 545af31c

y : 7fab1a1 536648a1 1778553a bd504650 9a3e3017

private key c :

c : 7251db9 0486ec38 c7b5fa48 01509251 e1c1baa1

public key Q :

x : 104931e 28440337 e1d9596a baab0a5e 6edbd668

y : fed596 17875d96 f822efe3 15c3b793 eae8515d

Example 6: n=155 bits

a2 : 3ad89d a8278400 6a468427 6eb52e36 75192e17

a6 : fffe202 97eeade8 7535b64e ef46fd00 771f8507

Base point P :

x : 736b98f 0af0e23d 98645286 9ec12595 3bb523ab

y : 119bf37 e3486733 75a1a759 955d08d4 f2d44454

private key c :

c : 74bd6c5 b11eb9fb aa2008ab 1421858a 58c5b599

public key Q :

x : 67c3b2d 76af4c68 45e8576b d4cee873 e644b23f

y : 2d1b76b 9ee7d52f 5a3154a7 49bf82c6 7fa69ec5

Example 7: n=173 bits

```
a2 :    1fff ffffffff ffffffff ffffffff ffffffff ffffffff
a6 :    1fff ffffffff ffffffff ffffffff ffffffff ffffffff
Base point P :
x :     13c5 44136e20 dacb0eea 358dcdf6 46ba8547 0fda0aaf
y :      69f 0419abb4 d4f94960 e4a5b01f 5feaefa8 7d40ae76
private key c :
c :       1b e10bec0d 953acd8c a37577a3 0f1c73bc 2c8e8fd5
public key Q :
x :      39d 312e6fda 6c5507af b5e4b2c5 312a7930 9e6477a7
y :     192c 74cb81b5 a4c959f0 dd1df342 0ceeaac8 898303c3
```

Example 8: n=173 bits

```
a2 :       1 5b3cb9a1 8fe52a3d 4cc05920 caa4f2b1 7c682af2
a6 :   15b04 1d8a121c 74c84653 97b6c30f 4fadff64 bce86df2
Base point P :
x :     1f67 f43f7b75 fce12bce 46fd7331 4c04248f 8671de5d
y :     17af d85d7132 673d3a91 b14df784 74b80d84 e40d836c
private key c :
c :      1d3 d91376fc d7bb1c59 c7057e80 24c96f1c 0c550126
public key Q :
x :     1113 3d013f5e cbe4bcb5 644de4fe c7d949c5 19993f97
y :      707 254df99f ee191c36 9c79e822 303f6d30 879aec8b
```

Example 9: n=173 bits

```
a2 :       8 6ca9f209 17f12a53 06d1c990 9b9e9102 7a5136f2
a6 :   113f6 3af371d5 28cd33f7 a94d255c ecbaaca7 57186df2
Base point P :
x :      47b 7f4c2633 3f84b65b 82bef409 c9dd884a 751b764f
y :      cb0 70e53ae6 c6abda6d 0fa555c2 4cb9ed7a 60288e8d
private key c :
c:        54 dbaaacf9 a467e7c1 f416e71a 8aff78e0 2e2346c1
public key Q :
x :      8f5 e0dd3934 03432022 9d7cb4dd 45b6eda7 9674c557
y :      ce5 6faa7d70 46a67747 02190ecf ce1ac463 e2f6bd01
```