FPGA IMPLEMENTATIONS OF ELLIPTIC CURVE CRYPTOGRAPHY AND TATE PAIRING OVER BINARY FIELD

Jian Huang

Thesis Prepared for the Degree of

MASTER OF SCIENCE

UNIVERSITY OF NORTH TEXAS

August 2007

APPROVED:

Hao Li, Major Professor
Phil Sweany, Committee Member
Krishna Kavi, Committee Member and Chair of
the Department of Computer Science and
Engineering
Oscar Garcia, Dean of the College of Engineering
Sandra L. Terrell, Dean of the Robert B. Toulouse
School of Graduate Studies

Huang, Jian. FPGA Implementations of Elliptic Curve Cryptography and Tate Pairing over Binary Field. Master of Science (Computer Engineering), August 2007, 70 pp., 12 tables, 21 figures, references, 52 titles.

Elliptic curve cryptography (ECC) is an alternative to traditional techniques for public key cryptography. It offers smaller key size without sacrificing security level. Tate pairing is a bilinear map used in identity based cryptography schemes. In a typical elliptic curve cryptosystem, elliptic curve point multiplication is the most computationally expensive component. Similarly, Tate pairing is also quite computationally expensive. Therefore, it is more attractive to implement the ECC and Tate pairing using hardware than using software. The bases of both ECC and Tate pairing are Galois field arithmetic units. In this thesis, I propose the FPGA implementations of the elliptic curve point multiplication in GF (2^{283}) as well as Tate pairing computation on supersingular elliptic curve in GF (2^{283}). I have designed and synthesized the elliptic curve point multiplication and Tate pairing module using Xilinx's FPGA, as well as synthesized all the Galois arithmetic units used in the designs. Experimental results demonstrate that the FPGA implementation can speedup the elliptic curve point multiplication by 31.6 times compared to software based implementation. The results also demonstrate that the FPGA implementation can speedup the Tate pairing computation by 152 times compared to software based implementation.

DEDICATION

To my parents.

ACKNOWLEDGEMENTS

I would like to express sincere gratitude to my major professor, Dr. Hao Li, for his guidance on my research and my course work. He has provided me with a lot of valuable advice and constructive criticism. I would also like to thank my other committee members, Dr. Kavi and Dr. Sweany for giving me helpful suggestions on my thesis. I owe special thanks to Dr. Tate for his kindly help on cryptography theories. I am thankful to my CSRL colleagues Cameron, Tommy, Suresh, Tim, Pete, Afrin, and Paul for their generous support. I am also grateful to Jerry, Chengyang, Ping, Yomi, and Toby for their encouragement during my graduate studies at UNT.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION AND PREVIOUS WORK

1.1. Motivation and Objective

Elliptic Curve Cryptography (ECC) and Tate pairing are two new types of public key cryptography. Using cryptography is an efficient way to protect confidential information to be sent over an insecure media, such as the Internet. Galois field arithmetic operations are the bases of both elliptic curve cryptography and Tate pairing. Elliptic curve cryptography and Tate pairing are both computationally expensive. FPGA implementation can improve the performance of ECC and Tate pairing compared with software implementation. Meanwhile FPGA have the advantage of flexibility compared to traditional ASIC implementation. We used resource sharing and parallel processing in our FPGA implementations of elliptic curve cryptography and Tate pairing. Binary field Galois field arithmetics are easier to be implemented in hardware. Therefore, our elliptic curve cryptography and Tate pairing are based on binary field.

1.1.1. Elliptic Curve Cryptography

E-commerce has become more and more popular in recent years. According to the Census Bureau of the US Department of Commerce, retail e-commerce sales for the fourth quarter of 2006 was $29.3 billion [3]. Hence, the security of web transactions is extremely important because a lot of sensitive information are transmitted over the Internet during these transactions, e.g., credit card numbers, social security numbers, etc.

Cryptography is the most standard and efficient way to protect the security of web transactions. It can be used to protect the confidentiality, integrity, authentication, and non-reputation of the web transactions. There are two major categories of cryptography schemes, i.e., public-key cryptography and symmetric-key cryptography. In public-key cryptography, the receiver and sender have their own private key and share a common public key. In symmetric-key cryptography, the receiver and sender must have the same private key, which makes it difficult to manage the private key. Public-key cryptography is easy for key distribution and key management. But it is not as efficient as symmetric-key cryptography [19, 39]. Thus, it is interesting to use dedicated hardware for public-key cryptography to improve the performance.

A well-known public-key cryptography algorithm is RSA, which was first proposed by Rivest, Shamir and Adleman in 1977 [40]. The security of RSA is based on hardness of integer factorization problem. It is commonly used in the secure sockets layer (SSL) protocol, which is the most popular way of protecting secure web transactions nowadays. SSL runs over transportation layer and it secures many application protocols such as HTTP, Telnet and FTP. However, due to the performance issue of RSA, using SSL usually slows down the web servers by three to nine times [9].

Elliptic Curve Cryptography (ECC) is an efficient substitution for RSA. It was originally proposed by Victor Miller of IBM and Neal Koblitz from the University of Washington [25, 32]. The security of ECC is based on the hardness of elliptic curve discrete logarithm problem (ECDLP). ECC can improve the performance of SSL because ECC has smaller key length yet still provides the same security level compared with RSA. Smaller key length results in faster computation, lower power consumption, and lower memory and bandwidth. Table 1.1 shows the equivalent key sizes of ECC and RSA [17]. Currently, 1024-bit RSA is standard, and it is projected that its size will increase to 2048 bits after 2010. The performance issues of RSA with such a large key size will then become a dominant force, which can severely affect the performance of RSA. So, we would like to use 283-bit ECC in place of the 2048-bit RSA since it can significantly reduce the key length and still provides the same security level.

Despite ECC's advantages over RSA, software based ECC implementations usually require long computation time, hence makes it difficult to be effectively utilized in real-time

Table 1.1. Equivalent Key Sizes between ECC and RSA

ECC	RSA	Protection lifetime
163	1024	until 2010
283	3072	until 2030
409	7680	beyond 2031

web-based transactions. To overcome this drawback, we propose an efficient FPGA implementation of ECC over GF(2^{283}), where GF stands for Galois Field, and 2^{283} means 283-bit binary operation. The key arithmetic operation in ECC is point multiplication. It determines the performance of the elliptic curve cryptosystem because it is the most computationally expensive unit.

The main contribution of our FPGA based design is the resources sharing and parallel processing optimization. The simulation results show that our implementation is significantly faster than the software implementation as well as previous FPGA implementations with the same security level [12, 27].

1.1.2. Tate Pairing

Identity based cryptography schemes have opened a new territory for public key cryptography [8, 38, 42]. Using identity based cryptography schemes, a sender can derive the public key of a receiver without receiving the certificate of the receiver issued by a certificate authority (CA). The public key can be derived from the identity of the receiver such as the email or IP address. Pairing over elliptic curve can be used to construct the identity based cryptography schemes. It is a map from two points on the elliptic curve to another multiplicative group. It has special properties of bilinearity. Currently, the most commonly used pairing methods are Tate pairing [13] and Weil pairing [31]. Originally, Weil pairing was used to attack public key cryptosystems. Later it was used for pairing based cryptosystems. It can be computed using either Miller algorithm [33] or modified Miller's algorithms [6, 7, 26].

Tate pairing is more efficient than Weil pairing because it requires one application of Miller's algorithm instead of two and it allows a host of optimizations [1]. Tate pairing is more than two times faster than Weil pairing [14]. Currently, Tate pairing computation is the most popular method used in many identity based cryptography schemes [8, 38, 42]. However, Tate pairing is computationally expensive, which makes it difficult to be efficiently implemented using software. We design and implement Tate pairing using field programmable gate array (FPGA) over binary field GF(2^m). It is more efficient for hardware implementation over binary field compared with other fields such as cubic field GF(3^m) [47]. The main arithmetic units needed for the Tate pairing computation include addition, Galois field multiplication, Galois field squaring, and Galois field inversion.

Our proposed FPGA implementation of the Tate pairing computation is in GF(2^{283}). We design the top level architecture for Tate pairing in order to optimize resources sharing. The simulation results show that our FPGA based implementation is faster than the software implementation as well as previous hardware implementations [24, 30].

1.1.3. Advantage of FPGA Implementation

We choose FPGA implementation because it can improve the performance of ECC and Tate pairing through resource sharing and parallel processing compared to software implementation. Meanwhile, FPGA implementation has the advantage of flexibility compared to ASIC implementation. For instance, using FPGA, the curve parameters for ECC and Tate pairing can be reconfigured. FPGA implementation is also suitable for hybrid private and public key scheme. For example, SSL may use ECC as public key cryptography, and AES as symmetric key cryptography. FPGA can configure the proper cryptographic schemes and sharing the resources.

1.2. Contribution

The contributions of this thesis are outlined as follows:

- Investigation of the applications for ECC and Tate pairing.
- Design a hardware architecture for elliptic curve cryptography.

- Design a hardware architecture for Tate pairing.
- FPGA implementation of elliptic curve cryptography.
- FPGA implementation of Tate pairing.

1.3. Previous Work

1.3.1. Elliptic Curve Cryptography

Hardware implementation of ECC has better performance than software implementation. Existing hardware implementations vary in the following aspects: GF(2^m), GF(p), key lengths (from 163 bits to 233 bits), platforms (FPGA, ASIC, sensor). In this section, we review some of the FPGA implementations of ECC over GF(2^m).

Orlando and Paar designed a reconfigurable elliptic curve processor (ECP) over GF(2^{167}) [35]. The ECP consists of main controller, arithmetic unit controller and arithmetic units. The point multiplication can be computed in 0.21 ms using the Montgomery algorithm. This work is generally considered as the benchmark of FPGA implementation of ECC. Its main advantages include scalable hardware architecture and reprogrammable processing units. Sandoval and Uribe proposed a hardware architecture that can perform three different ECC algorithms, i.e. elliptic curve Diffie-Hellman (ECDH), elliptic curve digital signature (ECDSA), elliptic curve integrated encryption scheme (ECIES) [34]. The main functional units in their cryptosystem are: coprocessor for scalar multiplication, random number generator, algorithms modules, and main controller. Its scalar multiplication can be completed in 4.7 ms for GF(2^{191}). Ernst et al. presented a generator based elliptic curve cryptosystem in [12]. The generator program can create customized VHDL netlists according to different key sizes and multiplier radix. Thus, this work is flexible in validating the correctness of the design. The authors chose Massey-Omura finite field multiplier, and Double-and-add algorithm for point multiplication. Their point multiplication can be computed in 6.85 ms for GF(2^{270}). Later, Leung et al. presented a microcoded FPGA based elliptic curve processor [27], which is similar to that presented in [12]. This design is parameterized for arbitrary key sizes and it allows for rapid development of different control flows. They used normal basis for the Galois field operations and the point multiplication can be computed in 14.3 ms for GF(2^{281}).

In addition to the hardware implementations discussed above, there exist other FPGA implementations for binary field in literature, such as [4, 11, 15, 21, 41, 43, 46]. A survey study conducted by Dormale and Quisquater is presented in [10], which summaries these FPGA based implementations.

1.3.2. Tate pairing

Tate pairing is relatively a new topic in cryptography. Thus, there are not many existing hardware implementations of it. The existing implementations also vary from characteristics (GF(2^m), GF(3^m), GF(p)), elliptic curve types (supersingular, non-supersingular, different embedded degrees). In this section, we briefly review some hardware implementations for supersingular elliptic curve over GF(2^{283}).

McCusker et al. designed Tate pairing over GF(2^{283}) [30]. The authors used the algorithm proposed by Kwon's [26]. They implemented Galois field multiplication, squaring, exponentiation and inversion. In addition, they used Karatsuba's algorithm [36] for the Galois field multiplication in GF(2^{4m}).

Keller et al. also used Karatsuba algorithm for the Galois field multiplier in GF(2^{4m}) [24]. Karatsuba algorithm uses GF(2^m) multiplier to realize the GF(2^{4m}) multiplier. They used a digit-serial architecture to design the GF(2^m) multiplier [48].

Shu et al. used two modified Kwon's algorithms in [45, 47]. They designed squaring, multiplication, inverter and exponentiation. In addition, they made use of a digit-serial multiplier in the design. Their implementation can compute the Tate pairing over GF(2^{283}) quickly. The algorithm used for Tate pairing in [47] is different from [24, 30]. In our work, we implemented the algorithms used in [30].

1.3.3. Other Implementations

In this section, we select some ASIC implementations and sensor implementations of elliptic curve cryptography.

1.3.3.1. *ASIC Implementations.* Akashi Satoh and Kohji Takano presented a processor that can support both prime and binary finite field for arbitrary prime numbers and irreducible polynomials [44]. This characteristic is achieved by introducing a dual field multiplier. A Montgomery multiplier with an optimized data bus and an on-the-fly redundant binary converter boost the throughput of the EC scalar multiplication. All popular cryptographic functions such as DSA, EC-DSA, RSA, CRT, and prime generation are also supported. Scalar multiplication can be performed in 0.19 ms in the binary field $F_{2^{160}}$.

Fabio Sozzani et al. presented a hardware implementation of ECC that includes some parallelism to maximize the usage of the field function units [49]. They combined the Double&ADD algorithm with the Montgomery algorithm to compute two different kP scalar multiplications in parallel. They have carried out modifications of the scheduling to synchronize the Control Units according to the availability of resources. The coprocessor is synthesized in 0.13 μm CMOS technology (VLSI CMOS9 library, STMicroelectronics). For 163-bit points, the scalar multiplication can be completed in 0.027 ms on average of two independent kP scalar multiplications.

1.3.3.2. *Sensor Implementations.* David J. Malan et al. presented the first known implementation of elliptic curve cryptography over F_{2^p} for sensor networks based on the 8-bit, 7.3828-MHz MICA2 mote [29]. Using instrumentation of UC Berkeley's TinySec module, the authors argued that there was a need for an efficient, secure mechanism for distribution of secret keys among nodes. And through their analysis of the implementation for TinyOS of multiplication of points on elliptic curves, public-key infrastructure is viable for TinySec key's distribution. They demonstrated that public keys can be generated within 34 seconds, and that shared secrets can be distributed among nodes in a sensor network within the same, using just over 1 kilobyte of SRAM and 34 kilobytes of ROM. Polynomial base is used for the implementation. The key size is 163-bit.

Haodong Wang et al. described a public-key implementation of access control in a sensor network [51]. The authors implemented elliptic curve cryptography over primary field, a public-key cryptography scheme. Their implementations are conducted on TelosB mote (TPR2400), which is the latest product in the Mote family. The experiment results show that the fixed point multiplication can be completed in 3.13 seconds and random point multiplication can be completed in 3.51 seconds. The digital signature can be completed in 3.35 seconds. And all these implementations are based on 160-bit elliptic curve.

1.4. Thesis outline

The rest of this thesis is organized as follows. In Chapter 2, we provide the background for ECC and Tate paring. In Chapter 3, we cover the main applications of ECC and Tate pairing. In Chapter 4, we discuss the detailed designs of the Galois field arithmetic units. In Chapter 5, we present the algorithms and architectures for ECC and Tate pairing, including the experimental results and comparisons of our implementations and previous implementations. In Section 6, We conclude our work.

CHAPTER 2

BACKGROUND

2.1. Cryptography

2.1.1. Symmetric Key Cryptography

There are two main categories of cryptography. One is symmetric key cryptography, the other is public key cryptography. The basic encryption/decryption scheme of symmetric key cryptography is shown in Figure 2.1 [50]. In Figure 2.1, plaintext is the original form of the message that sender wants to send to recipient. Ciphertext is the encrypted form of the original message which can be transmitted in an insecure channel such as Internet. The sender and recipient use the same secret key for the encryption and decryption function. Therefore, it is called symmetric key cryptography.

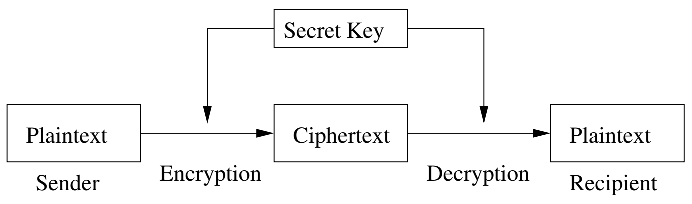

FIGURE 2.1. Symmetric Key Encryption/Decryption Scheme

2.1.2. Public Key Cryptography

The basic encryption/decryption scheme of public key cryptography is shown in Figure 2.2 [50]. The senders use recipient's public key for encryption. The recipient can decrypt the ciphertext using his own private key. In symmetric key cryptography, each pair of sender and recipient have a secret key. In public key cryptography, only the sender's public key is made publicitly, and multiple senders can use recipient's public key for the encryption.

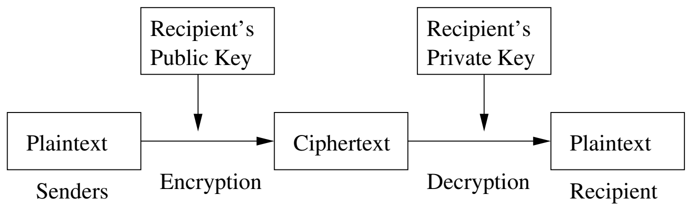

FIGURE 2.2. Public Key Encryption/Decryption Scheme

2.1.3. Digital Signature

The basic scheme of digital signature is shown in Figure 2.3 [50]. Just like the physical signature in the real life, digital signature is used to ensure the unforgeable identity of the singer. Digital signature is a reversal of the public key encryption/decryption scheme in Figure 2.2. Firstly, the sender can create a message identifying himself such as 'I am Harry'. Secondly, the sender can encrypt the identity message using his private key to create a digital signature. Then the digital signature can be transmitted to the recipient via Internet. After receiving the digital signature, the recipient can decrypt the digital signature using sender's public key. Finally, the recipient gets the identity message 'I am Harry'. The digital signature is unforgeable because only the person with access to the sender's private key can create the digital signature.

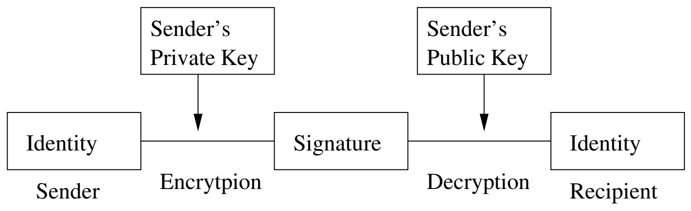

FIGURE 2.3. Digital Signature Scheme

2.1.4. Digital Envelopes

Digital envelope is a combination of symmetric key cryptography and public key cryptography technology. It improves the performance of the cryptosystem since the public key cryptography is relatively slow. The basic scheme of digital envelope is shown in Figure 2.4 [50]. In the digital envelope, the sender uses symmetric key cryptography to encrypt the plaintext with the session key. The session key is a generated secrete key usually used for one session. Then the sender uses public key cryptography to encrypt the session key with recipient's public key, which creates the digital envelop. Finally, the sender sends the ciphertext of the original messages and digital envelop to the recipient. In the recipient's side, the digital envelope is first decrypted to derive the session key. Then the recipient can use the session key to decrypt the ciphertext to get the plaintext.

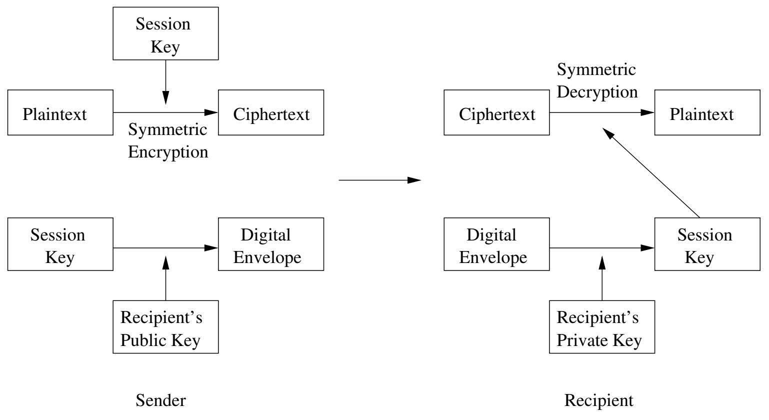

FIGURE 2.4. Digital Envelope

2.2. ECC

2.2.1. Group Law

In our design, we use points on the non-supersingular elliptic curve over GF(2^m) for point multiplication, point addition and point doubling. This is because non-supersingular elliptic curve provides the highest level of security [12]. The curve we choose has the following characteristic as defined in Equation (1).

$$E : y^2 + xy = x^3 + ax^2 + b \tag{1}$$

where $a, b \in$ GF(2^m). The points on E(GF(2^m)) include all the points satisfying Equation (1) and an identity point ∞. Let $P = (x_1, y_1), Q = (x_2, y_2) \in E(GF(2^m))$, where $P \neq \pm Q$ and $P \neq -P$. The group law of points in E(GF(2^m)) can be depicted as following [19]:

- Identity: $P + \infty = \infty + P = P$.

- Negatives: $-P = (x_1, x_1 + y_1)$.
- Point addition: $P + Q = (x_3, y_3)$, where $x_3 = \lambda^2 + \lambda + x_1 + x_2 + a$, and $y_3 = \lambda(x_1 + x_3) + x_3 + y_1$ with $\lambda = (y_1 + y_2)/(x_1 + x_2)$.
- Point doubling: $2P = (x_3, y_3)$, where $x_3 = \lambda^2 + \lambda + a = x_1^2 + b/x_1^2$, and $y_3 = x_1^2 + \lambda x_3 + x_3$ with $\lambda = x_1 + y_1/x_1$.

Figure 2.5 and Figure 2.6 show the geometric illustrations of point addition and point doubling on an elliptic curve respectively [45]. We can see that the results of addition and doubling of elliptic curve points are also on that curve.

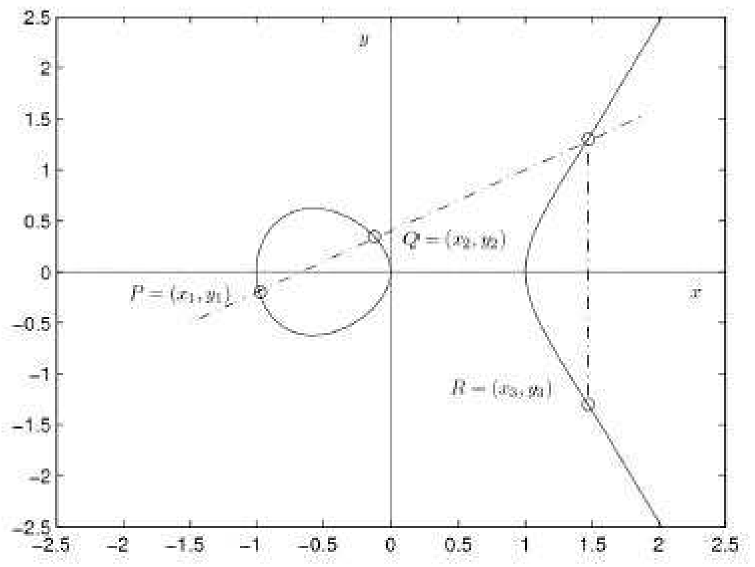

FIGURE 2.5. Point Addition: $P + Q = R$ [45]

2.2.2. Projective Coordinates

According to the group law of points on elliptic curve E, we can see that both point addition and point doubling need a Galois field inversion. Galois field inversion is much more expensive than Galois field multiplication. Using projective coordinates can eliminate the use of Galois field inversion in point addition and point doubling. The point addition and point doubling in projective coordinates can be computed as following [28]:

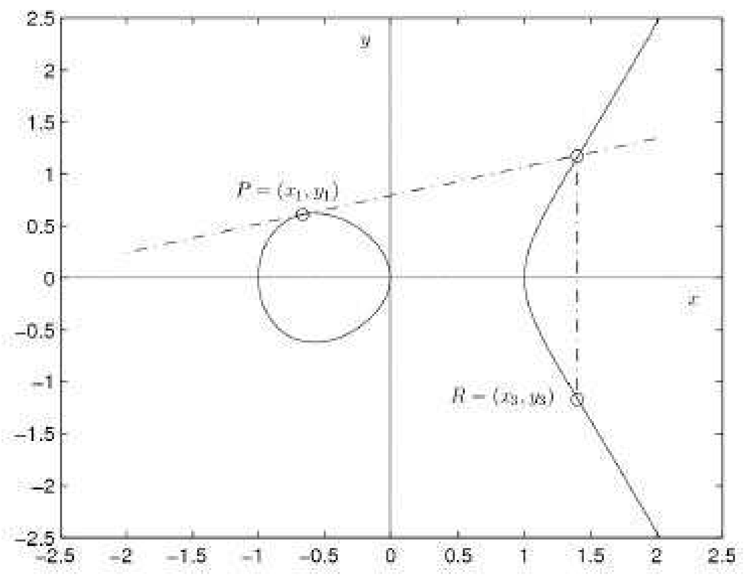

FIGURE 2.6. Point Doubling: $P + P = R$ [45]

- Point addition in projective coordinates:

$$Z_3 = (X_1 \cdot Z_2 + X_2 \cdot Z_1)^2 \tag{2}$$

$$X_3 = x \cdot Z_3 + (X_1 \cdot Z_2) \cdot (X_2 \cdot Z_1) \tag{3}$$

where (X_3, Z_3) is the result of the point addition in projective coordinate, and (X_1, Z_1) (X_2, Z_2) are the projective coordinates of P and Q, respectively.

- Point doubling in projective coordinates:

$$Z = X_1^4 + b \cdot Z_1^4 \tag{4}$$

$$X = Z_1^2 \cdot X_1^2 \tag{5}$$

where (X, Z) is the result of the point doubling in projective coordinates, and (X_1, Z_1) is the projective coordinates of P.

2.2.3. Security of ECC

The security of ECC is based on the hardness of elliptic curve discrete logarithm problem (ECDLP) [34]. ECDLP is to find d, given points P, Q on the elliptic curve, where $Q = dP$.

2.3. Tate pairing

2.3.1. Groups

We need to define some concepts in groups theory. First we define additive group. For any two points $P, Q \in$ group G, if $P + Q \in G$, then group G is an additive group. There is a negative point of P, i.e. $-P$. And $P + (-P) = 0$. 0 is zero element of the additive group. Additive group is also abelian group, that means the order of the addition doesn't matter, i.e. $P + Q = Q + P$. Multiplication $\underbrace{aP = P + P + \cdots + P}_{a}$, which means P adds to itself a times. An example for the additive group is the addition modulo.

Next we define multiplicative group. It's similar to additive group. For any two points $X, Y \in$ group H, if $X\dot{Y} \in H$, then group H is a multiplicative group. $X\dot{X}^{-1} = 1$, Here 1 is the identity element of the multiplicative group. And a multiplicative group is abelian group if $X \cdot Y = Y \cdot X$. Exponentiation $X^n = \underbrace{X \cdot X \cdots \cdot X}_{n}$, here multiplication repeats n times. An example for the multiplicative group is multiplication modulo.

The additive group and multiplicative group will be used in the pairing.

2.3.2. Pairing

Pairing can be regarded as a function to map points from an additive group to a multiplicative group. Let G_1 be an additive group with q elements. q is also called the order of the group. Let q be a large prime. Let $P \in G_1$ and P be a non zero point, then $qP = \text{inf}$. $P, 2P, 3P, ...(q-1)P, aP = \text{inf}$ is the set of all the q elements of group G_1. P is also called a generator of G_1. Let G_2 be a multiplicative group also with q elements. Pairing can be defined as a function e:

$$G_1 \times G_1 \rightarrow G_2 \tag{6}$$

Function e maps two points in G_1 to one point in G_2. And pairing has some very special properties, which are called bi-linearity. This means that, given any $Q, R, S \in G_1$, we have

$$e(Q, R + S) = e(Q, R) \cdot e(Q, S). \tag{7}$$

$$e(Q + R, S) = e(q, S) \cdot e(R, S). \tag{8}$$

Here, addition in the left part of the equation is addition in additive group G_1. Multiplication in the right part of the equation is multiplication in multiplicative group G_2. We can use the equation 7 and equation 8 to further derive new equations.

$$e(2P, P) = e(P + P, P) = e(p, p) \cdot e(p, p) = e(p, p)^2 = e(P, P + P) = e(P, 2P)$$

Similarly, we can get $e(3P, P) = e(P, P)^3 = e(P, 3P)$. Therefore, we have

$$e(aP, bP) = e(P, P)^{ab} = e(abP, P) = e(P, abP) \tag{9}$$

This is a very important equation for use in the pairing based cryptography. We will see how it is used in the applications later.

2.3.3. Elliptic Curves

We can't pick any kinds of additive group for pairing use. Elliptic curves are a good candidate for the additive group. Elliptic curves have been used for public key cryptography. It is believed that elliptic curve cryptography is more efficient than other public key cryptography such as RSA and El Gamal in term of the key length. A 160 bits key length in elliptic curve cryptography is comparable to a 1024 bits in RSA in terms of the security. A general form of elliptic curve over a finite field F_q can be expressed as Weierstrass form:

$$Y^2 + a_1XY + a_3Y = X^3 + a_2X^2 + a_4X + a_6 \tag{10}$$

Here all the coordinates a_1, a_2, a_3, a_4, a_6 is in the finite field F_q. Finite field is also called Galois field, it means a field that contains finitely numbers of elements. Usually people use binary field F_{2^m} or prime field F_p in the elliptic curve cryptography. Binary field is more efficient for the hardware implementation. Therefore, we use binary field in this thesis. For security reason, people want to choose m as odd or prime.

There are different kinds of elliptic curves. One is called supersingular elliptic curve, the other is called non-supersigular elliptic curve. Supersingular elliptic curve is usually considered as dangerous for cryptography use. But recent research shows with careful use, supersingular elliptic curves can be used in the pairing based cryptography. To our best understanding, most of the papers in pairing based cryptosystem use supersingular elliptic curves.

There are more mathematical definitions we need to know. First we define the embedding degree k. Embedding degree can be also called as security multiplier. It is an integer, and it is the minimal degree of the extension satisfying $E[l] \subset E(F_{q^k})$. Here l is positive integer. $E[l]$ means the set of points on elliptic curve E, and $lP = \infty$, $P \in E(F_q)$. There are two commonly used supersingular elliptic curves in pairing based cryptography. One is $Y^2 + Y = X^3 + X + b, b \in 0, 1$. The other is $Y^2 + Y = X^3$. The first one has embedded degree of 4, and the later one has embedded degree of 2. In this thesis, we use supersingular elliptic curve:

$$E_b : Y^2 + Y = X^3 + X + 1 \tag{11}$$

So points on E_b is the additive group G_1 in the equation 6. Therefore, we will need elliptic curve addition, elliptic curve double, point multiplication in G_1. Also finite field multiplication for G_2. We will use projective coordinate for the elliptic curve computations. Detailed arithmetic of elliptic curve will be introduced later.

2.3.4. Security of Pairing

The security of many pairing based cryptosystems is based on the hardness of Bi-linear Diffie-Hellman Problem (BDHP) [37]. BDHP is defined as: Given four points in group G_1, say $P, Q = aP, R = bP, S = cP$, compute $e(P, P)^{abc}$. Here we only know the points P, Q, R, S, but we don't know the value of a, b, c. This makes BDHP a hard problem. Many pairing based cryptography schemes based on the BDHP. If someone can solve BDHP, then pairing based cryptosystem will be cracked.

In addition to BDHP, there are some other problems related to it. First is Computational Diffie-Hellman Problem (CDHP). CDHP is defined as: Given points $P, Q = aP, R = bP \in G_1$, compute $T = abP$. Here a, b are also unknown. This is also a hard problem. It is shown by mathematicians that if G_1 is well chosen and the size of G_1 is the order of 2^{160}, then CDHP is computational infeasible. And we can also prove that BDHP is no harder than CDHP. This means if CDHP can be solved, then BDHP can be solved. Next we will prove this. If CDHP is solved, that means given $Q = aP, R = bP$ in group G_1, we can compute $T = abP$. Then using equation 9, we can get:

$$e(T, S) \Rightarrow e(P, P)^{abc}$$

This means that BDHP is also solved. Therefore, we proved that BDHP is no harder than CDHP.

In addition to CDHP, Discrete Logarithm Problem (DLP) is another hard problem related to BDHP. DLP is defined as: Given points $P, Q = aP \in G_1$, compute a. Actually, the security of elliptic curve cryptography is based on DLP. Similar to CDHP, we can also prove that BDHP is no harder than DLP. If DLP is solved, BDHP is also solved. Next we prove this. If given $P, Q = aP$ in group G_1, we can compute a. Similarly, given $P, R = bP \in G_1$, we can also compute b. If we know both a and b, we can easily compute $T = abP$ using point multiplication. This means that CDHP is solved. In turn, BDHP is solved. Therefore, we proved that BDHP is no harder than DLP.

CHAPTER 3

APPLICATIONS OF ECC AND TATE PAIRING IN CRYPTOGRAPHY

3.1. Applications of ECC

3.1.1. Elliptic Curve Diffie-Hellman

The goal of Elliptic Curve Diffie-Hellman (ECDH) key exchange protocol is to establish a secret session key between two parties over an insecure channel [19]. The two parties, Alice and Bob, want to establish a secret key without Oscar (the adversary) being able to compute this key. Alice and Bob first agree on a (non-secret) elliptic curve and a (non-secret) fixed curve point F. Alice chooses a secret random integer Ak which is her secret key, and publishes the curve point AP = Ak * F as her public key. Bob does the same: BP = Bk * F. Then Alice send her public key AP to Bob, Bob can compute the session key using his secret key: session key = Bk * AP = Bk * (Ak * F) = Ak * (Bk * F) = Ak * BP. This means Alice can also compute the session key using her secret key and the public key of Bob. And the security of this scheme relies on the discrete logarithm problem for elliptic curve, i.e. it is difficult to compute k given F and k*F.

3.1.2. Elliptic Curve Digital Signature Algorithm

Elliptic Curve Digital Signature Algorithm (ECDSA) is used to generate a digital signature for a message. The ECDSA signature generation scheme is shown in Algorithm 1 [19]. The ECDSA signature verification scheme is shown in Algorithm 2 [19].

In Algorithm 1 and Algorithm 2, the domain parameters $D = (q, FR, S, a, b, P, n, h)$ are composed of:

- q: field order
- FR: field representation
- S: seed to randomly generate the coefficients of the elliptic curve

Algorithm 1 ECDSA signature generation

```
INPUT: Domain parameters $D = (q, FR, S, a, b, P, n, h)$, private key $d$, message $m$.
OUTPUT: Signature $(r, s)$.
1. Select a random number k from $[1, n-1]$
2. Compute $kP = (x, y)$ and $r = x \bmod n$. If $r = 0$, go to step 1.
3. Compute $s = k^{-1}(H(m) + dr) \bmod n$. If $s = 0$ then go to step 1.
return $(r, s)$.
```

Algorithm 2 ECDSA signature verification

```
INPUT: Domain parameters $D = (q, FR, S, a, b, P, n, h)$, public key Q, message m, signature $(r, s)$.
OUTPUT: Signature Accepted or Rejected.
1. Verify $r$ and $s$ are integers in $[1, n-1]$. If fails, finish and reject the signature.
2. Compute $w = s^{-1} \bmod n$ and $H(m)$.
3. Compute $u_1 = H(m)w \bmod n$ and $u_2 = rw \bmod n$.
4. Compute $X = u_1P + u_2Q = (x_2, y_2)$.
5. Compute $v = x_2 \bmod n$, accept the signature if and only if $v = r$.
```

- a, b: coefficients of the elliptic curve
- P: base point
- n: order of P
- h: cofactor $h = \#E(F_q)/n$

We can prove the correctness of the ECDSA using mathematic inductions. If the signature is made by a legal user, then $s \equiv k^{-1}(e + dr) \pmod{n}$. This means

$$k \equiv s^{-1}(e + dr) \equiv s^{-1}e + s^{-1}dr \equiv we + wrd = u_1 + u_2d \pmod{n}.$$

Thus,

$$\begin{aligned} X &= u_1P + u_2Q \\ &= (u_1 + u_2)P \end{aligned}$$

$$
\begin{aligned}
&= kP \\
&= (x, y)
\end{aligned}
$$

Where, $X = (x_2, y_2)$. $r = x \bmod n$ and $v = x_2 \bmod n$. Hence, we get $r = v$.

3.1.3. ECC based SSL

Secure socket level (SSL) is the most widely used protocol for secure web transactions nowadays. SSL can provide authentication, encryption of the sensitive data. In addition, SSL is very flexible. It can adopt different protocols for key exchange, encryption, and hashing. A combination of the protocols used in SSL is called cipher suite. For example, RSA-RC4-SHA is a cipher suite, which uses RSA as the public key cryptography, RC4 as symmetric key cryptography, and SHA as hashing function. Public key cryptography is used for key exchange because it's easy for key management and key distribution. But public key cryptography is very computationally expensive. Therefore, people usually uses symmetric key for data encryption.

There are two components in SSL, i.e. handshake protocol and record layer protocol. Handshake protocol uses public key cryptography to agree on a cipher suite, to do key exchange between client and server. A master secret (session key) is established after the handshake protocol. It can also be used to authenticate the client and server. Record layer protocol uses symmetric key cryptography to encrypt the data. Therefore, public key cryptography is mainly used in handshake protocol of the SSL. In the next subsection, we introduce the RSA-based handshake and the ECC-based handshake.

3.1.4. RSA-based Handshake and ECC-based Handshake

RSA-based handshake is the most commonly used handshake for SSL now. The operations of RSA-based handshake is shown in Figure 3.1. The client and sever negotiate the cipher suite using ClientHello message and ServerHello message. Then server sends its RSA public key to the client. Client verifies the server's public key and encrypt a premaster key with server's public key and sends the premaster key back to the server. Server can decrypt the premaster key using its private key. Therefore, both client and server can use the premaster key to create a master key for the record layer protocol.

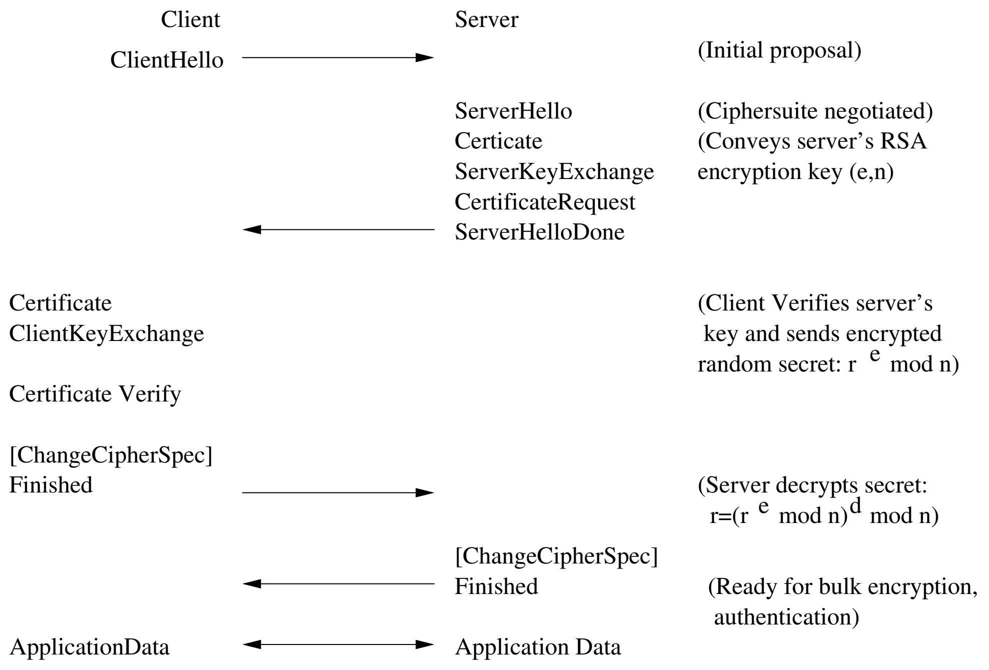

FIGURE 3.1. RSA-based SSL Handshake [18].

The ECC-based handshake is shown in Figure 3.2. The procedure of ECC-based handshake is similar to that of RSA-based handshake. First the client and server negotiate a cipher suite, e.g., ECDH-ECDSA-RC4-128-SHA. Then the server sends its public key to the client. The public key can be signed using ECDSA by a certificate authority. Then the client verifies the server's public key and sends its own public key to the server. Both client and server can use their own private key to create the shared premaster key now.

3.1.4.1. *Comparisons.* The public-key cryptographic operations in the two modes described above are shown in Table 3.1. For RSA-based handshake, the client performs one RSA verification operation to verify the authentication of the server, and one RSA encryption to encrypt the premaster key. The server performs one RSA decryption to get the premaster key. For ECC-based handshake, the client performs one ECDSA to verify the authentication of the server, and one ECDH to create the shared premaster key. The server just performs one ECDH operation to generate the same secret.

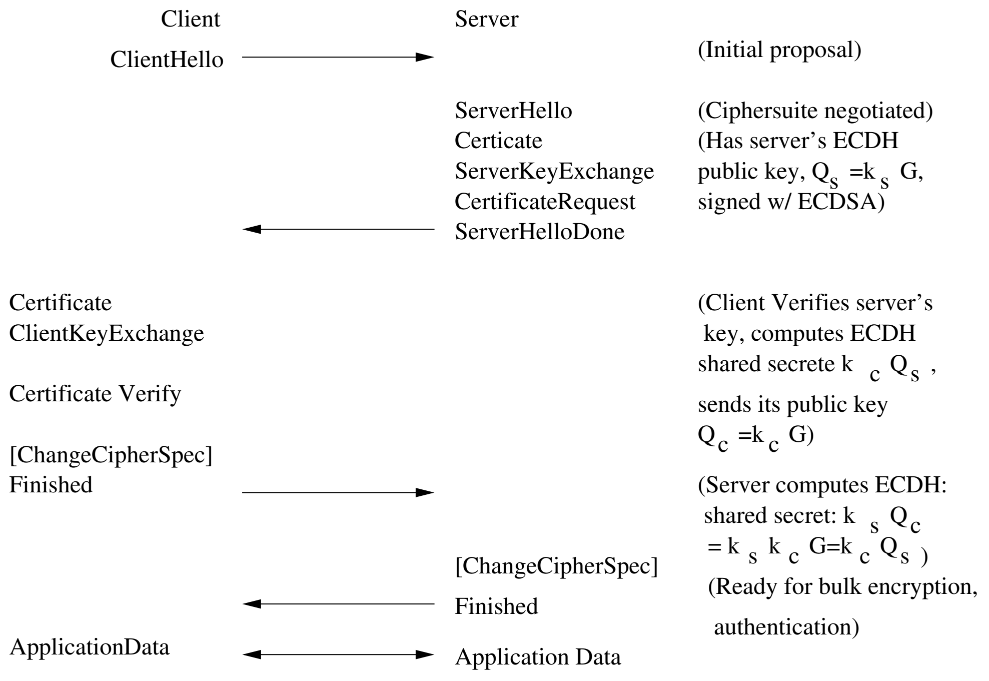

FIGURE 3.2. ECC-based SSL Handshake [18].

TABLE 3.1. Public-key Cryptographic Operations in an SSL Handshake [18]

	RSA	ECDH-ECDSA
Client	RSA_{verfiy}	$ECDSA_{verify} + ECDH_{op}$
Server	$RSA_{decrypt}$	$ECDH_{op}$

The performance evaluation of the RSA-based SSL handshake and ECC-based SSL handshake is shown in Table 3.2. Gupta et. al performed two different cipher suites for the evaluation, i.e., TLS-RSA-RC4-128-SHA and TLS-ECDH-ECDSA-RC4-128-SHA [18]. They tested different security levels for RSA and ECC operations, including 1024, 1536, 2048 bits RSA, and 160,192,224 bits ECC. The authors used http-load to send multiple HTTPS requests simultaneously. Then they used OpenSSL speed command to measure the RSA and ECC operations.

The time in Table 3.2 means first response time. It is the delay between starting and SSL handshake and receiving the first packet in the HTTPS response. It can be used to estimate the latency for the user to get the first response after clicking the URL. The ops/sec in the performance evaluation is the rate that the server can satisfy web page requests. We can see from the Table 3.2 that the performance of ECC-based SSL is better than RSA-based SSL in all three different security level. The higher the security level, the better performance advantage of ECC-based SSL over RSA-based SSL. Therefore, it's beneficial to replace RSA with ECC in SSL protocol.

TABLE 3.2. Performance Evaluation [18]

	ECC-160	RSA-1024	ECC-192	RSA-1536	ECC-224	RSA-2048
Time (ms)	3.69	8.75	3.87	27.47	5.12	56.18
Ops/sec	271.3	114.3	258.1	36.4	195.5	17.8
Performance ratio	2.4:1		7.1:1		11:1	
Key-size ration	1:6.4		1:8		1:9.1	

3.2. Applications of Tate pairing

3.2.1. Tripartite Key Agreement

Tripartite key agreement is also known as three parties Diffie-Hellman key agreement. It is an extension of two parties Diffie-Hellman key agreement. The goal of two parties Diffie-Hellman key agreement protocol is to establish a secret session key between two parties over an insecure channel. The two parties, Alice and Bob, want to establish a secret key without Oscar (the adversary) being able to compute this key. Alice and Bob first agree on a (non-secret) elliptic curve and a (non-secret) fixed curve point P. Alice chooses a secret random integer a which is her secret key, and publishes the curve point $AP = a \cdot P$ as her public key. Bob does the same: $BP = b \cdot P$. Then Alice send her public key AP to Bob, Bob can compute the session key using his secret key: session key $= b \cdot AP = b \cdot (a \cdot P) = a \cdot (b \cdot P) = a \cdot BP$. This means Alice can also compute the session key using her secret key and the public key of Bob. And the security of this scheme relies on the discrete logarithm problem for elliptic curve, i.e. it is difficult to compute k given P and $k \cdot P$. Here $\cdot$ is point multiplication over the elliptic curve.

Tripartite Diffie-Hellman key agreement is just an extension of the two parties Diffie-Hellman key agreement. There are three parties A, B, C. We need to find the minimal communications among these three parties to establish a common session key among them. For a naive tripartite key agreement protocol, we will need six steps:

(i) $A \rightarrow B, C : aP (\text{mod} \quad r)$

(ii) $B \rightarrow A, C : bP (\text{mod} \quad r)$

(iii) $C \rightarrow A, B : cP (\text{mod} \quad r)$

(iv) $A \rightarrow B, C : abP, acP (\text{mod} \quad r)$

(v) $B \rightarrow A, C : abP, bcP (\text{mod} \quad r)$

(vi) $C \rightarrow A, B : acP, bcP (\text{mod} \quad r)$

Here, P is a point on elliptic curve that three parties agreed. a is the secrete key of A, b is the secrete key of B, and c is the secrete key of C. r is a reduction polynomial for the modulo computation. The rightarrow $\rightarrow$ means broadcast messages sequentially. For example, $A \rightarrow B, C : abP, acP (\text{mod} \quad r)$ means that party A sends abP and acP to both B and C. After these six steps in naive three parties key agreement, party A, B, and C can compute $abcP$. Therefore $abcP$ is the common session key for these three parties. The security of the naive tripartite Diffie-Hellman key agreement is still based on the hardness of discrete logarithm problem.

A more advanced tripartite Diffie-Hellman key agreement based on pairing is proposed by Joux [22]. It only requires the first three steps of the naive tripartite DH key agreement. And it is the minimal steps of the tripartite DH key agreement because each party needs to send their public key at least once. The tripartite DH key agreement by Joux is shown below.

(i) $A \rightarrow B, C : aP (\text{mod} \quad r)$

(ii) $B \rightarrow A, C : bP (\mathrm{mod} \quad r)$

(iii) $C \rightarrow A, B : cP (\mathrm{mod} \quad r)$

A, B, C just broadcast once to the other parties. And using the bilinearity of pairing, i.e. equation 9, A, B, and C can all compute the common key. The processes are shown below.

$$A : e(bP, cP)^a = e(P, P)^{abc}$$

$$B : e(aP, cP)^b = e(P, P)^{abc}$$

$$C : e(aP, bP)^c = e(P, P)^{abc}$$

Therefore, the session key for party A, B, C is $e(P, P)^{abc}$ in Joux's tripartite DH key agreement. The security of it is based on the hardness of BDHP.

3.2.2. Identity-Based Key Agreement

The second example of pairing application is identity-based key agreement proposed by Sakai [42]. The difference between the identity-based key agreement and Diffie-Hellman based key agreement is that in identity-based key agreement, part A doesn't need to communicate with part B directly. Part A just needs to know the identity of part B, and vice versa. But in the identity-based key agreement, both A and B need to preregister with a trusted authority (TA) in ahead. There is a harsh function H being used here. H is a public function which can convert an arbitrary string to an element in the additive group G_1 shown as below.

$$Q_A = H(ID_A) \in G_1$$

$$Q_B = H(ID_B) \in G_1$$

Here Q_A and Q_B are all public information derived from the identity of A and B. The identity of A and B can be their email addresses or IP addresses. In our cases, Q_A and Q_B are the points on the supersingular elliptic curve. TA will select a secret integer s. Then TA computes sQ_A and sends it to A. Similarly, TA computes sQ_B and sends it to B. Then using the bilinear property shown as below, both A and B can compute the common session key.

$$e(SQ_A, Q_B) = e(Q_A, Q_B)^s = e(Q_A, sQ_B) \tag{12}$$

A knows sQ_A from TA and $Q_B = H(ID_B)$ from the identity of B. Therefore, A can compute the common key $e(Q_A, Q_B)^s$. Similarly, B knows sQ_B and $Q_A = H(ID_A)$, so B can also compute the common key $e(Q_A, Q_B)^s$.

Similar to the tripartite key agreement, the security of the identity-based key agreement is also based on the hardness of BDHP. But an extra vulnerability here is TA. Since TA knows the secret integer s, Q_A and Q_B, TA can compute the common session key. So A, B must trust TA. TA can't disclose the secrets to the adversary or being eavesdropped by the adversary. Otherwise, the security of the identity-based key agreement is gone.

3.2.3. Identity-Based Encryption

The third example is identity-based encryption (IBE) proposed by Boneh and Franklin [8] in 2001. This opens a new area for the public key cryptography, which is called identity-based public key cryptography (ID-PKC). For traditional public key infrastructure (PKI), a certificate authority (CA) is needed to verify the authentication of the public keys. For IBE, CA is not needed.

For the identity-based encryption, we need a trusted authority to choose the system parameters for IBE. The system parameters are shown in the following list.

- G_1: additive group
- G_2: multiplicative group
- e: pairing function
- s: system-wide master secret
- P: one point in group G_1
- P_{pub}: $P_{pub} = sP \in G_1$, it's a point multiplication on elliptic curve
- H: harsh function that can convert an arbitrary string to an element in G_1

- H_1: harsh function that can convert an element in G_2 to a string of length k. Here k is the length of the plaintext.

G_1, G_2, e, P, P_{pub} are all public information. In addition to the system parameters, the IDs of users are also public. IDs can be email, name, IP address and so on. Next we will show how the IBE works.

Suppose user A needs to send some encrypted message to user B. The secret key of user A is sQ_A, which is generated by TA. Here $Q_A = H(ID_A)$. First user B encrypts the plaintext. Let M be a k bit plaintext. B chooses a secret key r first. Then user B computes:

$$U = rP \in G_1$$

$$V = M \oplus H_1(e(Q_A, P_{pub})^r)$$

The encryption process needs point multiplication in group G_1, xor operation, and pairing computation. The ciphertext that user B sends to user A is $< U, V >$.

A decrypts the ciphertext using the bilinear property shown as blow.

$$e(Q_A, P_{pub})^r = e(Q_A, sP)^r = e(Q_A, P)^r s = e(sQ_A, rP) \tag{13}$$

The process of decryption is shown below.

(i) A gets the secret key sQ_A from TA
(ii) A gets the first part of the ciphertext $U = rP$
(iii) A computes $W = H_1(e(Q_A, P_{pub})^r)$
(iv) A gets the plaintext $M = W \oplus V$

Here, since $e(Q_A, P_{pub})^r = e(sQ_A, rP) = e(sQ_A, U)$, W can be computed using pairing. We can see from the above encryption steps, user B only needs the identity of user A, which is ID_A, as well as the system parameters. Similar to the previous applications, the security of the identity-based encryption is also based on the hardness of BDHP.

3.2.4. Identity-based Signature schemes

Another application example of pairing in identity-based cryptography is identity-based signature scheme proposed by Paterson [38] in 2002. It is similar to the identity-based encryption scheme. We still need a TA, and TA will choose a set of system parameters just like that in identity-based encryption. The system parameters include additive group G_1, multiplicative group G_2, pairing function e, one point P in G_1, and public key $P_{pub} = sP$, where s is the system master secret. In addition to these parameters, we still need three harsh functions. Harsh function H can convert arbitrary string to an element in G_1. Harsh function H_2 can convert arbitrary strings into integers. Harsh function H_3 can convert elements in G_1 into integers.

Suppose user A want to sign a message M. A first register with TA, and TA will send the private key sQ_A to A, where $Q_A = sH(ID_A)$. Then A picks a random integer r. The signature is the pair shown below.

$$< U = rP, V = r^{-1}(H_2(M)P + H_3(U)sQ_A) > \tag{14}$$

Here, r^{-1} is the inverse modulo, which means $r^{-1} \cdot r = 1 (\mathrm{mod}\ q)$. q is the reduction polynomial. The signature is just the pair $< U, V >$ computed by A. We can see that there is no pairing computation in the signing procedure. Therefore, the signing procedure can be very fast.

The verify procedure is to compute the pairing of $< U, V >$, and compare the results with $e(P,P)^{H_2(M)} \cdot e(P_{pub}, Q_A)^{H_3(U)}$. This is also due to the bilinear property of the pairing computation. The proof is shown below.

$$\begin{aligned} e(U,V) &= e(rP, r^{-1}(H_2(M)P + H_3(U)sQ_A)) \\ &= e(P, H_2(M)P + H_3(U)sQ_A) \\ &= e(P, H_2(M)P) \cdot e(sP, H_3(U)Q_A) \\ &= e(P,P)^{H_2(M)} \cdot e(P_{pub}, Q_A)^{H_3(U)} \end{aligned}$$

For the verifying procedure, we will need pairing computations. The security of the identity-based signature is also based on the hardness of BDHP.

CHAPTER 4

IMPLEMENTATIONS OF GALOIS FIELD ARITHMETIC UNITS

4.1. Adder in Galois field

The addition unit in Galois field is straightforward to implement over binary field. It can be designed using an array of XOR gates. This is one of advantages of implementing the Tate pairing computation over binary field.

4.2. Squarer in GF(2^{283})

We have designed a bit parallel squarer which is much faster than multiplying two binary polynomials [52]. Assume the binary polynomial is $a(x) = \sum_{i=0}^{282} a_i x^i$, then the squaring formula can be calculated using equation (15) [19]:

$$a(x)^2 = \sum_{i=0}^{282} a_i x^{2i} \tag{15}$$

Because we use $f(x) = x^{283} + x^{12} + x^7 + x^5 + 1$ as the reduction polynomial, we can obtain the formula of the coefficients of $a(x)^2$ by replacing x^{283} by $x^{12} + x^7 + x^5 + 1$. Therefore, the squarer is simply a set of XOR arrays to recombine the coefficients of $a(x)$. And the gate count is proportional to the polynomial bit [47], which is 283 in our case.

4.3. Multiplier in GF(2^{283})

Multiplication is a basic computation for Tate pairing computation. There are many algorithms for computing it. We introduce the most significant bit first (MSB) multiplier and the digital serial multiplier here.

4.3.1. MSB Multiplier

The MSB multiplier algorithm [19] for F_{2^m} is shown in Algorithm 3. It is suitable for hardware implementation and it's area efficient. It consists of shift operations and xor operations, which can both be easily implemented in hardware.

Algorithm 3 MSB Algorithm

```
INPUT: $a = (a_{m-1}, ..., a_1, a_0), b = (b_{m-1}, ..., b_1, b_0) \in F_{2^m}$, $r = (r_{m-1}, ..., r_1, r_0) \in F_{2^m}$ is a part of reduction polynomial $f(z) = z^m + r(z)$
OUTPUT: $c = a \cdot b, c = (c_{m-1}, ..., c_1, c_0) \in F_{2^m}$
INITIAL: $c = 0$
for $i = m - 1$ downto 0 do
  $c = c_{m-1}r + leftshift(c)$.
  $c = c + b_i a$.
end for
return $c$
```

4.3.2. Digit Serial Multiplier

We use the digit serial multiplier introduced in [19]. The advantage of this digit serial multiplier over MSB multiplier is that it can increase the speed of the multiplication operation. The digit serial multiplier requires to use a reduction module. The algorithm to design the digit serial multiplier is shown in Algorithm 4.

In Algorithm 4, $l = \lceil m/k \rceil$, k is the digit size, and l is the number of digits. In our implementation, we set $m = 283$, $k = 32$, $l = 9$. Using the digit serial multiplier can improve the performance of the Galois field multiplier compared to bit serial multiplier.

4.4. Reduction in GF(2^{283})

The reduction function is used in designing multiplier. We adopt the fast reduction modulo algorithm with digit size of 32 in our implementation [19]. The pseudocode is shown in Algorithm 5.

Algorithm 4 Digit serial multiplier for GF(2^m)

```
INPUT: $a = \sum_{i=0}^{m-1} a_i z^i \in \mathrm{GF}(2^m)$, $b = \sum_{i=0}^{l-1} B_i z^{ki} \in \mathrm{GF}(2^m)$, reduction polynomial $f(z)$.
OUTPUT: $c = a \cdot b$.
Set $c = 0$
for $i = 0$ to $l - 1$ do
    $c = c + B_i a$.
    $a = a \cdot z^k \bmod f(z)$.
end for
return $c \bmod f(z)$.
```

Algorithm 5 Fast reduction modulo $f(x) = x^{283} + x^{12} + x^7 + x^5 + 1$ in GF(2^{283}) (with $W = 32$)

```
INPUT: A binary polynomial $c(z)$ of degree at most 564.
OUTPUT: $c(z) \bmod f(z)$.
for $i = 17$ downto 9 do
    $T = C[i]$.
    $C[i-9] = c[i-9] + (T << 5) + (T << 10) + (T << 12) + (T << 17)$.
    $C[i-8] = c[i-8] + (T >> 27) + (T >> 10) + (T >> 12) + (T >> 17)$.
end for{Reduce $C[i]z^{32i}$ modulo $f(x)$}
$T = C[8] >> 27$. {Extract bits $27 - 31$ of $C[8]$}
$C[0] = C[0] + T + (T << 5) + (T << 7) + (T << 12)$.
$C[8] = C[8] \& 0x7FFFFFF$. {Clear the reduced bits of $C[8]$}
return $(C[8], C[7], \cdots C[1], C[0])$.
```

Note, $C[i]$ is a 32-bit word of $c(z)$, i.e. $c(z) = (C[17], C[16], \cdots C[0])$, which is at most 564-bit long. And the reduction result only consist of $(C[8], C[7], \cdots C[0])$, which has bit width of 283. The reduction module is composed of shift registers, XORs, and AND gates. Our reduction module can finish the computation in 4 clock cycles.

4.5. Multiplier in GF(2^{1132})

We use the Karatsuba multiplier presented by Keller et al. to implement the multiplier in GF(2^{1132}) [23, 24]. It uses 9 283-bit multipliers and 22 XORs to implement one 1132-bit multiplier. The architecture of Karatsuba multiplier is shown in Fig. 4.1. Using Karatsuba multiplier can improve the performance of the Tate pairing computation. But it also increases the chip area as it uses nine multipliers in GF(2^{283}).

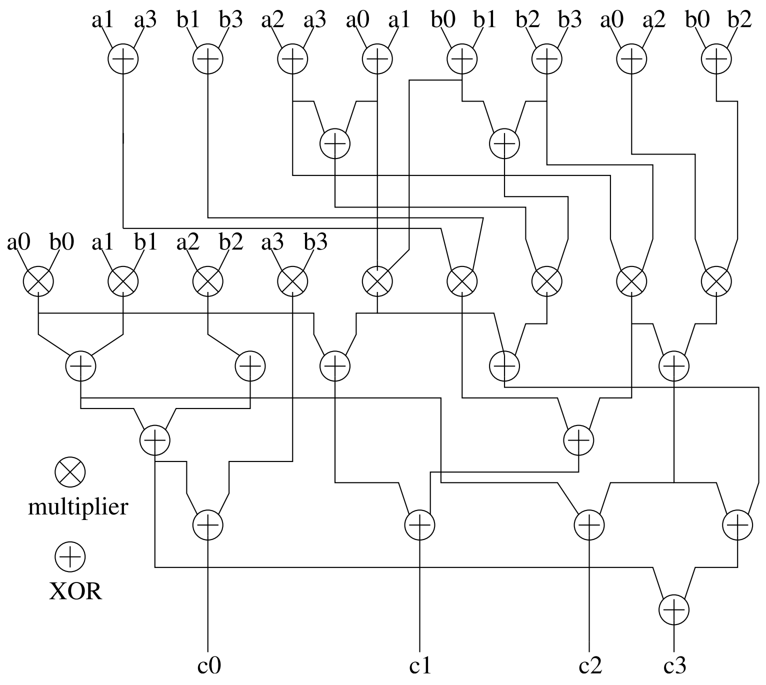

FIGURE 4.1. Karatsuba Multiplier Architecture for GF(2^{1132})

4.6. Exponentiation in GF(2^{1132})

The final exponentiation of the Tate pairing computation is to compute $c(x)^{2^{2\times 283}-1}$, where $c(x) \in GF(2^{1132})$. We divide it into two steps. The first step is to compute the exponentiation of $c(x)^{2^{2\times 283}}$, the second step is to compute the inversion of $c(x)^{-1}$ and then multiply it with the exponentiation computed in the first step.

The exponentiation can be computed using Frobenius map [30]. Let $c(x) = c_0 + c_1x + c_2x^2 + c_3x^3$. Note, $c(x) \in GF(2^{1132})$, and coefficients $c_0, c_1, c_2, c_3 \in GF(2^{283})$. Then,

$$c(x)^{2^{283}} = (c_0 + c_1) + (c_2 + c_3)x + (c_1)x^2 + c_3x^3$$

The exponentiation operation only requires addition and reordering of the coefficients. Therefore, the exponentiation in GF(2^{1132}) can be implemented using the inverter in GF(2^{1132}) and a few XORs.

4.7. Inverter in GF(2^{283})

Inversion is the most complex operation in Galois field arithmetic. It is based on Fermat's little theorem [16]. Let α be a nonzero element in GF(2^{283}), then $\alpha^{-1} = \alpha^{2^{283}-2}$. We can see that $2^{283} - 2 = \sum_{i=1}^{282} 2^i$. Thus,

$$\alpha^{-1} = \alpha^{\sum_{i=1}^{282} 2^i} = \prod_{i=1}^{282} \alpha^{2^i} \tag{16}$$

According to equation (16), the inversion can be implemented using 282 squarings and 281 multiplications. Actually, the number of multiplications can be reduced due to the following features [19, 20]. When m is odd:

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

When m is even: $\alpha^{2^{m-1}-1} = (\alpha^{2^{m-2}-1})^2\alpha$

Based on above, we derive the following formula to compute the inversion in GF(2^{283}).

$$
\begin{aligned}
tmp1 &= \alpha^{2^2-1} &=& \quad \alpha\cdot\alpha^2 \\
tmp2 &= \alpha^{2^4-1} &=& \quad tmp1\cdot(tmp1)^{2^2} \\
tmp3 &= \alpha^{2^8-1} &=& \quad tmp2\cdot(tmp2)^{2^4} \\
tmp4 &= \alpha^{2^{16}-1} &=& \quad tmp3\cdot(tmp3)^{2^8} \\
tmp5 &= \alpha^{2^{17}-1} &=& \quad \alpha\cdot(tmp4)^2 \\
tmp6 &= \alpha^{2^{34}-1} &=& \quad tmp5\cdot(tmp5)^{2^{17}} \\
tmp7 &= \alpha^{2^{35}-1} &=& \quad \alpha\cdot(tmp6)^2 \\
tmp8 &= \alpha^{2^{70}-1} &=& \quad tmp7\cdot(tmp7)^{2^{35}} \\
tmp9 &= \alpha^{2^{140}-1} &=& \quad tmp8\cdot(tmp8)^{2^{70}} \\
tmp10 &= \alpha^{2^{141}-1} &=& \quad \alpha\cdot(tmp9)^2 \\
tmp11 &= \alpha^{2^{282}-1} &=& \quad tmp10\cdot(tmp10)^{2^{141}} \\
\alpha^{-1} &= \alpha^{2^{283}-2} &=& \quad (tmp11)^2
\end{aligned}
\tag{17}
$$

where $tmp1$ to $tmp11$ are 283-bit registers used to store temporary data for the inversion operations. We can see from equation (17) that the inversion only needs 11 multiplications and 282 squarings.

4.8. Inverter in GF(2^{1132})

The inverter in GF(2^{1132}) is the most complex unit among all the arithmetic units. It can be implemented with one inverter in GF(2^{283}), one multiplier in GF(2^{1132}), and one exponentiation in GF(2^{1132}). The algorithm to compute inversion in GF(2^{1132}) is shown in Algorithm 6.

Algorithm 6 is derived from equation (18) which was proposed in [30].

$$\alpha^{-1} = \alpha^{2^{4m}-2} = (\alpha^r)^{-1}\alpha^{r-1} \tag{18}$$

where $m = 283$, $r = \frac{2^{4m}-2}{2^m-1}$ and

$$\alpha^r \in \mathrm{GF}(2^m), \quad \alpha \in \mathrm{GF}(2^{4m}).$$

Algorithm 6 Inversion in GF(2^{4m})

```
1: INPUT: $\alpha \in \mathrm{GF}(2^{4m})$.
2: OUTPUT: $\beta = \alpha^{-1}$, $\beta \in \mathrm{GF}(2^{4m})$.
3: Compute $\alpha^{r-1}$, here $r = \frac{2^{4m}-2}{2^m-1}$
4: Compute $\alpha^r = \alpha^{r-1}\alpha$.
5: Compute $(\alpha^r)^{-1}$.
6: Compute $\beta = (\alpha^r)^{-1}\alpha^{r-1}$.
7: **return** $\beta$.
```

Line 3 in Algorithm 6 can be computed as follows:

$$\alpha^{r-1} = ((\alpha^{2^{283}}\alpha)^{2^{283}}\alpha)^{2^{283}} \tag{19}$$

Thus, it can be implemented with three exponentiations in GF(2^{1132}) and two multiplications in GF(2^{1132}).

Line 4 in Algorithm 6 is a multiplication in GF(2^{1132}). Line 5 is implemented with an inverter in GF(2^{283}) because $\alpha^r \in \mathrm{GF}(2^{283})$. Line 6 is also a multiplication in GF(2^{1132}). So the entire inversion algorithm in GF(2^{1132}) can be implemented with one inverter in GF(2^{283}), one multiplier in GF(2^{1132}), and exponentiations in GF(2^{1132}).

4.9. Experimental Results

We first synthesize all the arithmetic units. The device utilization summary of the individual arithmetic units is shown in Table 4.1. The inverse in GF(2^{1132}) is the most complicated unit. It uses 33594 slices which accounts for 52% of the whole chip area. This is because the inverse in GF(2^{1132}) consists of inverse in GF(2^{283}) and multiplication in GF(2^{1132}). Squaring uses the least area among all the arithmetic units.

From Table 4.1, we observe that the synthesis will not meet area constraints if we instantiate inverter in GF(2^{1132}) as well as multiplier in GF(2^{1132}). Therefore, we share the multiplier in GF(2^{1132}) and inverter in GF(2^{283}) to implement the inverter in GF(2^{1132}). By doing so, the area constraints are met.

TABLE 4.1. Device utilization summary of Galois Field Arithmetic Units

Operation	Max Freq. (MHz)	# CLB Slices	# FF	# LUT
Mult. GF(2^{283})	246.670	1781 (2%)	2156 (1%)	3367 (2%)
Mult. GF(2^{1132})	248.447	25955 (41%)	32578 (25%)	48591 (38%)
Squaring GF(2^{283})	675.676	306 (0.48%)	317 (0.25%)	567 (0.45%)
Reduction GF(2^{283})	346.981	787 (1%)	825 (0.65%)	1439 (1%)
Inverse GF(2^{283})	160.817	7354 (11%)	6999 (5%)	13944 (11%)
Inverse GF(2^{1132})	122.592	33594 (52%)	43746 (34%)	54988 (43%)

CHAPTER 5

ALGORITHMS, ARCHITECTURES AND IMPLEMENTATIONS OF ECC AND TATE PAIRING

5.1. Algorithm for ECC

Point multiplication is to compute kP, where k is an integer and P is an point on an elliptic curve E defined over a field F_q. Point multiplication is also called scalar multiplication, and it dominates the execution time of elliptic curve cryptographic schemes. There are several algorithms for point multiplication over elliptic curve. We will introduce two most commonly used algorithms, i.e. right-to-left multiplication algorithm and Montgomery point multiplication algorithm.

5.1.1. Right-to-left algorithm

The algorithm for point multiplication using right-to-left binary method is shown in Algorithm 7.

Algorithm 7 Right-to-left Point Multiplication Algorithm.

```
INPUT: $k = (k_{t-1}, ...., k_1, k_0)_2, P \in E(F_q)$.
OUTPUT: $kP$.
$Q \leftarrow \infty$.
for $i$ from $n-2$ to 0 do
    if $k_i = 1$ then
        $Q \leftarrow Q + P$.
    end if
    $P \leftarrow 2P$.
end for
return $Q$
```

5.1.2. Montgomery Algorithm

In our design, we use Montgomery point multiplication algorithm for the implementation of point multiplication [19, 28, 43]. The pseudocode is shown in Algorithm 8.

Algorithm 8 Montgomery Point Multiplication Algorithm.

```
INPUT: An integer $k = (k_{n-1}, k_{n-2}, \cdots k_1, k_0,\ k_{n-1} = 1)$, a point $P(x, y) \in E(GF(2^m))$
OUTPUT: $Q = kP$.
Set $X_1 = x, Z_1 = 1, X_2 = x^4 + b, Z_2 = x^2$
for $i = n - 2$ downto 0 do
  if $k_i = 1$ then
    Pointadder($X_1, Z_1, X_2, Z_2$), Pointdouble($X_2, Z_2$)
  else
    Pointadder($X_2, Z_2, X_1, Z_1$), Pointdouble($X_1, Z_1$)
  end if
end for
return $Q = M_{xy}(X_1, Z_1, X_2, Z_2)$.
```

Note, "Pointadder" and "Pointdouble" in Algorithm 8 are computed using Equations (2) - (5). M_{xy} is the function to convert the projective coordinates to affine coordinates [4]. Its output, i.e., the coordinate of point Q, x_k and y_k can be computed as:

$$x_k = X_1/Z_1 \tag{20}$$

$$y_k = (x + x_k)[(y + x^2) + (X_2/Z_2 + x)(X_1/Z_1 + x)] \times (1/x) + y \tag{21}$$

5.2. Architecture for ECC

5.2.1. Design Hierarchy

The design hierarchy of a typical elliptic curve cryptosystem is shown in Fig. 5.1. The top level of the system contains cryptographic protocols. In an ECC based SSL connection, the ECC based cipher suite uses ECDH for key exchange, and ECDSA for authentication of the public key. Point multiplication is utilized in both of the ECDH and ECDSA protocol. The secondary level in the design hierarchy is point multiplication. Point multiplication is composed of point doubling and point addition. Point multiplication, point doubling and point addition are operations involving with the points on the elliptic curve. The bottom level of the ECC system is Galois field arithmetic including Galois field multiplication, Galois field inversion and Galois field squaring. Our design focuses on all but the protocol level of the elliptic curve cryptosystem.

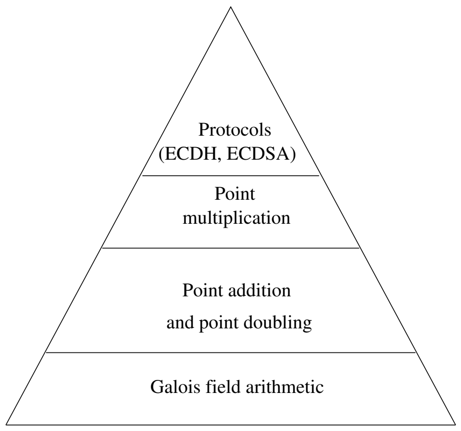

Figure 5.1. Hierarchy of Typical Elliptic Curve Cryptosystem.

5.2.2. Top level architecture and point multiplication

The top level architecture of a typical elliptic curve cryptosystem is illustrated in Figure 5.2. It is composed of main controller, register files, and point multiplier. The main controller is used to realize specific cryptographic protocols, such as ECDSA or ECDH. Point multiplier consists of point adder, point doubler and conversion module. And its implementation is our focus in this work. Details of the implementation of point multiplier is described in the next section.

The diagram of the point multiplier is shown in Figure 5.3. Based on the Montgomery point multiplication algorithm, the point multiplier is composed of point adder, point doubler, coordinates converter, squarer and XORs.

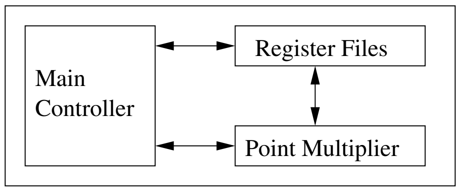

FIGURE 5.2. Top Level View of The Elliptic Curve Cryptosystem.

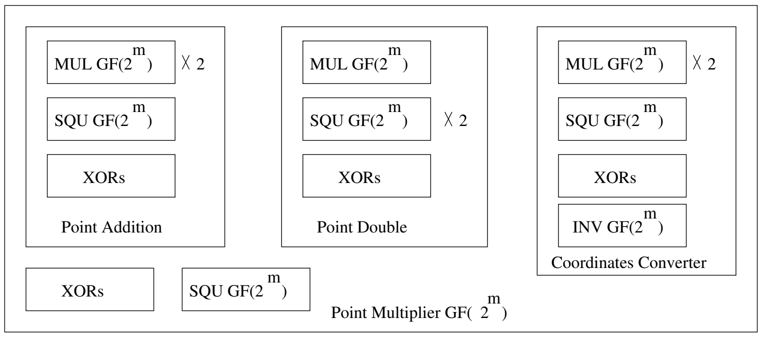

FIGURE 5.3. Architecture of Point Multiplier.

We use two Galois field multipliers, one Galois field squarer and XORs to implement point adder. Point doubler is composed of two Galois field squarers, one Galois field multiplier and XORs. The coordinates converter is more complicated than point adder and point doubler. It consists of two Galois field multipliers, one Galois field squarer, one Galois field inverter, and XORs. In our work, all the arithmetic units are designed in GF(2^{283}). The dataflows of the point adder, point doubler, and coordinates converter are shown in Figure 5.4, 5.5, and 5.6, respectively, where $t1, t2, t3, t4$ are 283-bit registers. The goal of our design is to optimize the parallel processing of the Montgomery point multiplication. Meanwhile, our design shares the arithmetic units in order to reduce chip area.

The block diagram of point multiplier is shown in Figure 5.7. One point double, one point adder, one finite field inversion and two finite field multipliers are instantiated in the point multiplier module. And the control unit controls the flow of the computation. The point definition is shown in Table 5.1. The state diagram of the point multiplier is shown in Figure 5.8.

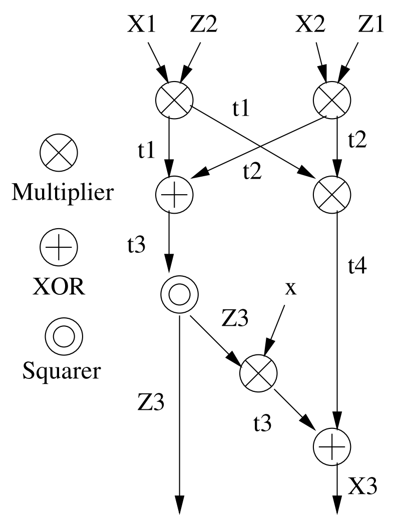

FIGURE 5.4. Dataflow of Point Adder

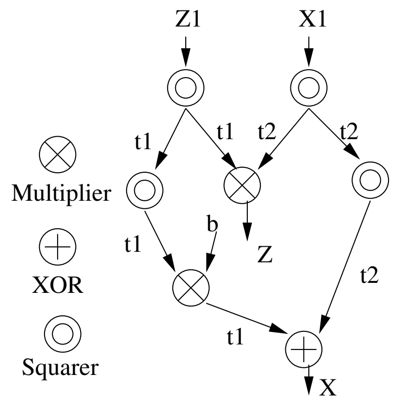

FIGURE 5.5. Dataflow of Point Doubler

5.3. Experimental Results of ECC

We have implemented and simulated the elliptic curve point multiplication with Xilinx's FPGA device. In order to show the effectiveness of hardware implementation over software based approach, we have also realized the design in software. We first provide the setups used in our work, then compare our FPGA based design with several previous works, and then show the difference between hardware and software implementations.

5.3.0.1. *Software Implementation.* The software implementation of the elliptic curve point multiplication is done using C++ and LiDIA. LiDIA is a C++ library of computational number theory [2]. The simulation of the point multiplication in GF(2^{283}) is based on Algorithm 8

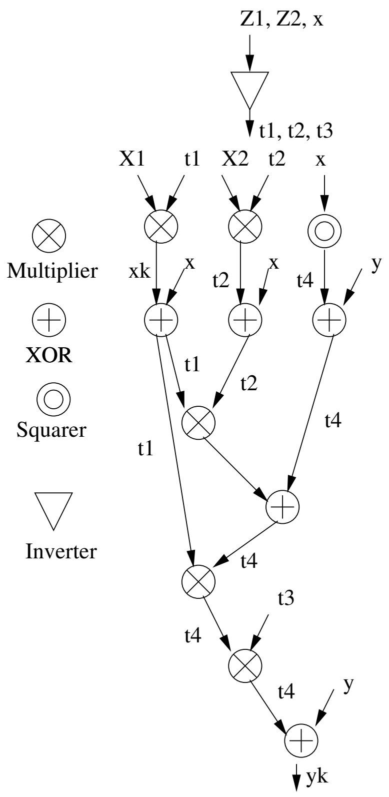

FIGURE 5.6. Dataflow of Coordinate Converter

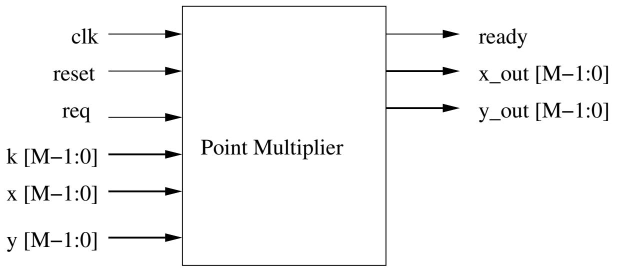

FIGURE 5.7. Block Diagram of Point Multiplier for $GF(2^m)$

and carried out on a Pentium4 2.8 GHz desktop with 1G memory. The source codes are compiled by GCC 4.1.1. The running time to perform a single Tate pairing operation is 9.6 ms.

TABLE 5.1. Port Definition of Point Multiplier

Name	Direction	Definition
clk	input	system clock
reset	input	system reset, low active
req	input	request signal for multiplication
k[M-1:0]	input	the integer for multiplication
x[M-1:0]	input	x coordinate of point on E in affine coordinates
y[M-1:0]	input	y coordinate of point on E in affine coordinates
ready	output	ready signal
x_out[M-1:0]	output	output result in affine coordinates
y_out[M-1:0]	output	output result in affine coordinates

5.3.1. FPGA Implementation

The hardware implementation is simulated by ModelSim XE and synthesized with Xilinx ISE 8.2i. The target device is Xilinx Virtex 4 XC4VFX140-FF1517-11. The optimization goal during synthesis is set as "speed", and the optimization effort is set to "normal".

5.3.1.1. *Synthesis Results.* We have synthesized all arithmetic units including point multiplication, point addition, point doubling. The device utilization summary of the individual arithmetic units is shown in Table 5.2. Column "Max Freq." lists the maximum frequency to run each unit individually. The rest columns show the number of CLB slices, Flip-flops (FF) and lookup-tables (LUT) used. The corresponding utility percentage is also given in Table 5.2. The coordinates converter is the most complicated unit which accounts for 35% of the total chip area.

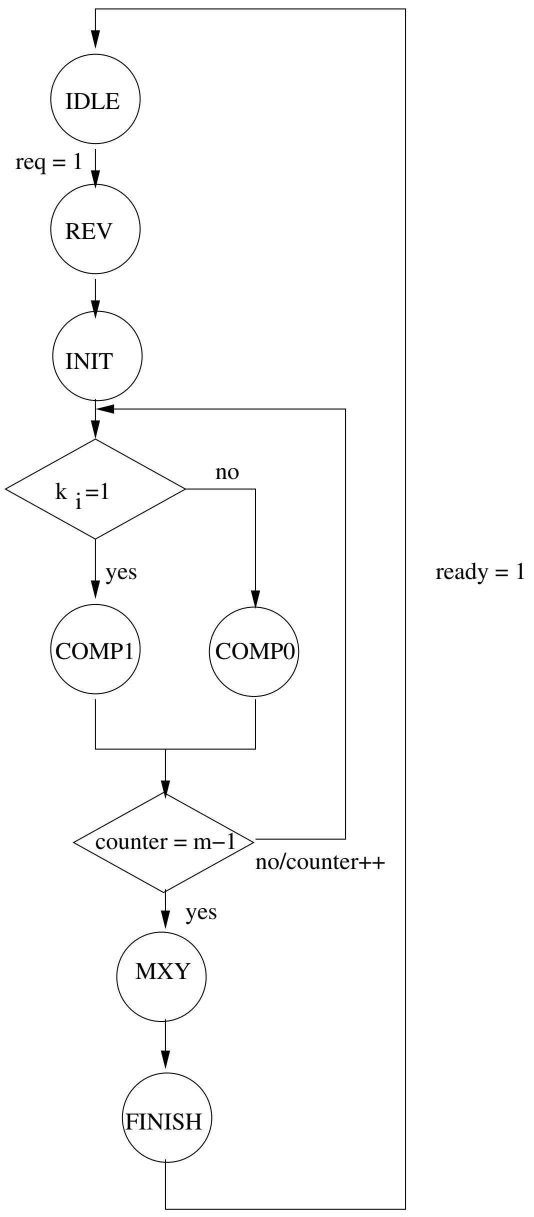

FIGURE 5.8. State Diagram of Point Multiplier for $GF(2^m)$

5.3.1.2. *Latency Comparison.* We have simulated the elliptic curve point addition, point doubling, coordinates converter and point multiplication in both software and hardware. The simulated latencies for these operations are shown in Table 5.3. Here, latency is the time to perform one specific arithmetic operation. The k values in our simulation have the same number of 1's and 0's in the binary representation.

Table 5.2. Device Utilization Summary of ECC.

Operation	Max Freq. (MHz)	# CLB Slices	# FF	# LUT
Point Addition GF(2^{283})	283.728	7412 (17%)	9016 (10%)	13826 (16%)
Point Doubling GF(2^{283})	281.861	5378 (12%)	6031 (7%)	10341 (12%)
Coordinates Converter	183.968	15009 (35%)	16129 (19%)	27753 (32%)
Point Multiplication GF(2^{283})	171.247	30001 (71%)	36142 (42%)	51094 (60%)

Table 5.3. Speedup of Hardware over Software.

Operation	FPGA Freq.	FPGA Latency	Software Latency	Speedup
Point Add GF(2^{283})	283.728 MHz	0.6 us	29 us	48
Point Double GF(2^{283})	281.861 MHz	0.41 us	21 us	51
Coord. Converter	183.968 MHz	24 us	58 us	2.4
Point Mult. GF(2^{283})	171.247 MHz	304 us	9600 us	31.6

According to Table 5.3, the FPGA implementation of the point multiplication is 31.6 times faster than the software implementation. We compare the simulated latency with Leung's [27] and Ernst's work [12] and show the results in Table 5.4. Our FPGA implementation of the point multiplication is 47 times faster than that in Leung's work (14.3 ms), and 22.5 times faster than that in Ernst's work (6.85 ms).

5.4. Algorithm for Tate pairing

Tate pairing can be computed using Miller's algorithm [33] or modified Miller's algorithms [5, 7, 26]. These algorithms are usually developed by mathematicians in number theory. The algorithm we choose to compute Tate pairing is shown in Algorithm 9 [30].

TABLE 5.4. Comparisons of Latency of Point Multiplication.

Design	Key Size	Latency
[27]	281	14.3 ms
[12]	270	6.85 ms
Our work	283	0.304 ms

This algorithm can be divided into four parts. The first part is initialization (line 1 to 3). The second part is accumulation (line 5 to 18). The third part is squaring (line 19 to 20). And the last part is exponentiation (line 22).

Using Algorithm 9, we compute Tate pairing for supersingular elliptic curve over GF(2^m). The inputs are two points P and Q on elliptic curve E_b. P, Q are m bits binary sequences. The output is $C(x)$. The coefficients of $C(x)$, i.e. c_3, c_2, c_1, c_0, are the pairing results, which are $4m$-bit binary sequences in the multiplicative group of G_2. And c_3, c_2, c_1, c_0 are m-bit sequences. Based on the above discussion, we can figure out that the Tate pairing computation requires at least Galois field multiplier in both GF(2^m) and GF(2^{4m}), and exclusive-or (XOR). Other operations such as squarer, exponentiation, and inverters are optional. However, they can be utilized to improve the performance. In our implementation, we use all of the above functional units.

5.5. Architecture for Tate pairing

5.5.1. Top Level

The top level architecture of Tate pairing is shown as Fig. 5.9. It is composed of the main controller, register files, and arithmetic units. Main controller can be designed using finite state machine (FSM) to control state transitions. Register files are implemented using flip-flops available on the FPGA chip. Let $m = 283$, the arithmetic units we have designed include multipliers in GF(2^m) and GF(2^{4m}), exponentiation in GF(2^{4m}), inverter in GF(2^m) and GF(2^{4m}), squarer in GF(2^m), reduction in GF(2^m).

Algorithm 9 Algorithm for Tate Pairing over GF(2^m)

```
1: INPUT: $P = (x_p, y_p), Q = (x_q, y_q)$
2: OUTPUT: $C(x)$
3: INITIAL $C(x) = c_3x^3 + c_2x^2 + c_1x^1 + x_1 = 1$
4: for $i = 1$ to $m$ do
5:     $xp = x_p^2, y_p = y_p^2$
6:     $z = x_p + x_q$
7:     $m_1 = x_p x_q$
8:     $w = z + m_1 + y_p + y_q + 1$
9:     $m_2 = c_0 w, m_3 = (c_2 + c_3)(z + 1)$
10:    $m_4 = (c_1 + c_2 + c_3)w$
11:    $m_5 = (c_0 + c_2 + c_3)(w + z + 1)$
12:    $m_6 = c_3(z + 1)$
13:    $m_7 = (c_1 + c_2)(w + z + 1)$
14:    $c'_0 = m_2 + m_3 + c_3$
15:    $c'_1 = m_2 + m_4 + m_5 + m_6 + c_0 + c_3$
16:    $c'_2 = m_2 + m_4 + m_5 + m_7 + c_1$
17:    $c'_3 = m_4 + m_7 + c_2$
18:    $c_0 = c'_0,\ c_1 = c'_1,\ c_2 = c'_2,\ c_3 = c'_3$
19:    $x_q = x_q^{2^{m-1}}$
20:    $y_q = y_q^{2^{m-1}}$
21: end for
22: $C(x) = C(x)^{2^{2\times m}-1}$
23: return $C(x)$
```

The port definitions are shown in Table 5.5. The ports can be categorized as control signals, data signals and system signals. Control signals include “req” and “ready”. Data signals include xp, yp, xq, yq, cx, and r. The reduction polynomial r used in our design equals

$x^{283}+x^{12}+x^7+x^5+1$, which is recommended by ANSI [19]. System signals include "clk" and "reset". The state diagram of Tate pairing computation is shown in Fig. 5.10. It consists of states such as IDLE, ACCU (accumulation), SQUA (squaring), EXP(exponentiation), and FINISH. These states correspond to the various parts described in Algorithm 9.

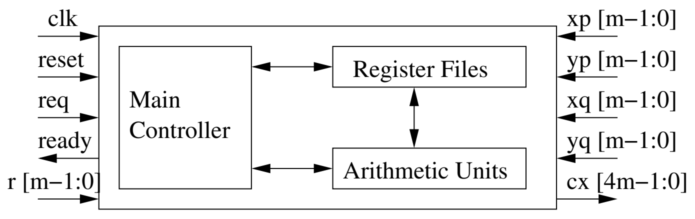

FIGURE 5.9. Top Level Architecture for Tate Pairing in $GF(2^m)$

TABLE 5.5. Port Definitions of Tate Pairing Module

Name	Direction	Definition
clk	input	system clock
reset	input	system reset, low active
req	input	request signal
xp[M-1:0]	input	x coordinate of point P
yp[M-1:0]	input	y coordinate of point P
xq[M-1:0]	input	x coordinate of point Q
yq[M-1:0]	input	y coordinate of point Q
r[M-1:0]	input	reduction polynomial
ready	output	ready signal
cx[4M-1:0]	output	computation results

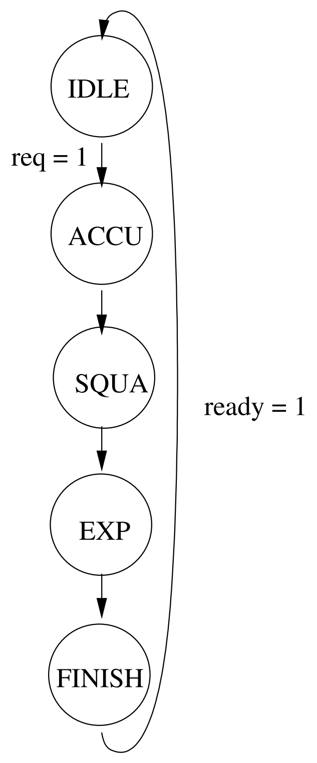

Figure 5.10. State Diagram of Tate Pairing Computation

5.5.2. Architecture of Arithmetic Units

The architecture of the arithmetic units in our design is shown in Fig. 5.11. This architecture is designed in such a way that we can share functional units to minimize chip area. The majority of operations are carried out in GF(2^{283}). So from this point on, an arithmetic unit will be in GF(2^{283}) by default unless specified otherwise. We use two multipliers (MUL), two squarers (SQU), an inverter (INV) in GF(2^{1132}), an inverter in GF(2^{283}), one multiplier in GF(2^{1132}), and XORs to implement the arithmetic units. Fig. 5.11 also illustrates the hierarchy of the arithmetic units. The inverter in GF(2^{1132}) is the most complex unit, and it consists of one inverter in GF(2^{283}) and one multiplier in GF(2^{1132}). The inverter in GF(2^{283}) is composed of one multiplier and one squarer. We adopt the Karatsuba multiplier for GF(2^{1132}), which is implemented with nine multipliers in GF(2^{283}) [24]. The multiplier is a digit serial multiplier [19], and it consists of a reduction module to reduce the degree to 283. Both the squarers and adders can be computed in one clock cycle in our implementation.

We use two multipliers and two squarers because they optimizes the parallel computation in the accumulation part and squaring part of the Tate pairing algorithm. The dataflow of the accumulation part of the Tate pairing algorithm is shown in Fig 5.12. For the squaring part, two squarers are also optimal because x_q and y_q can be squared simultaneously. In the finial exponentiation part, we use Frobenius map and one inverter in GF(2^{1132}). But we do not really instantiate the inverter in GF(2^{1132}). Instead, we share the inverter in GF(2^{283}) and multiplier in GF(2^{1132}) to implement the inverter in GF(2^{1132}). The detailed implementations of these arithmetic units will be presented in the following subsections.

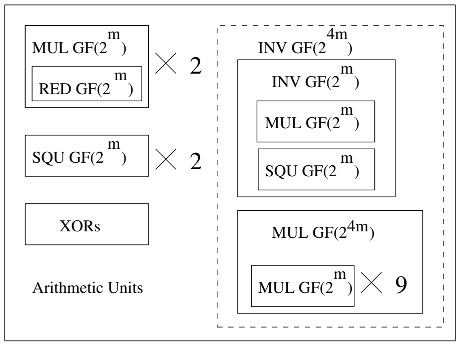

FIGURE 5.11. Architecture of Arithmetic Units

5.6. Experimental Results of Tate Pairing

5.6.1. Software Implementation

The software implementation of the Tate pairing computation is done using C++ and LiDIA. LiDIA is a C++ library for computational number theory [2]. We simulate the Tate pairing computation over GF(2^{283}) based on Algorithm 9 on a Pentium4 2.8GHz computer with 1G memory. The source codes are compiled by GCC 4.1.1. The running time to perform one Tate pairing operation is 90 ms.

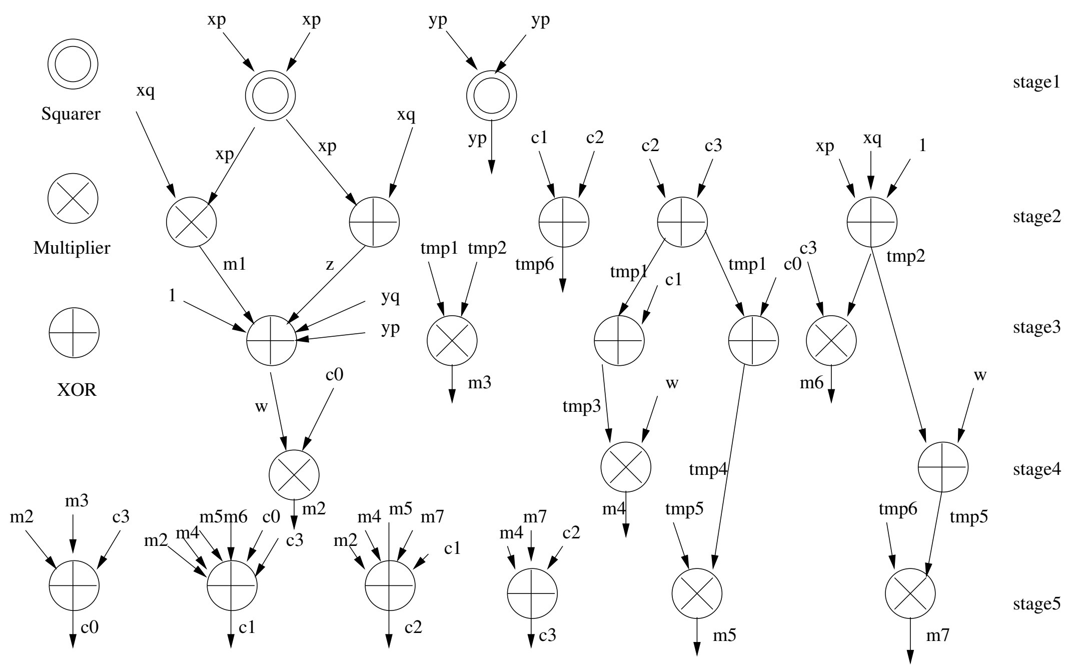

FIGURE 5.12. Dataflow for Accumulation of Tate Pairing

5.6.2. FPGA Implementation

The hardware implementation of Tate pairing is written in Verilog. The designs are simulated using Modelsim XE and synthesized using Xilinx ISE 8.2i. The target device is Xilinx Virtex 4 XC4VFX140-FF1517-11. The optimization goal of the synthesis is set as speed, and the optimization effort is set to normal.

5.6.2.1. *Synthesis Results.* We first synthesize all the arithmetic units. The device utilization summary of the individual arithmetic units is shown in Table 5.6. The inverse in GF(2^{1132}) is the most complicated unit. It uses 33594 slices which accounts for 52% of the whole chip area. This is because the inverse in GF(2^{1132}) consists of inverse in GF(2^{283}) and multiplication in GF(2^{1132}). Squaring uses the least area among all the arithmetic units.

TABLE 5.6. Device Utilization Summary of Tate Pairing

Operation	Max Freq. (MHz)	# CLB Slices	# FF	# LUT
Tate pairing GF(2^{283})	159.758	55844 (88%)	57753 (45%)	104860 (83%)

Then we synthesize the Tate pairing computation with these arithmetic units. The device summary for Tate pairing computation is also shown in Table 5.6.

5.6.2.2. *Latency Comparison.* We have simulated all arithmetic units to evaluate the latency for each operation. Here, latency is the time to perform one specific arithmetic operation. In order to compare the latency of our work with that of McCusker's work [30], we also set the simulation frequency as 250 MHz. The latency of each arithmetic unit is shown in Table 5.7.

TABLE 5.7. Latency Comparisons of Arithmetic Units

Operation	Latency of [30]	Latency of our work	Speedup
Mult. GF(2^{283})	897 ns	218 ns	4.11
Mult. GF(2^{1132})	940 ns	254 ns	3.7
Squaring GF(2^{283})	4 ns	4 ns	1
Inverse GF(2^{283})	10769 ns	5758 ns	1.87
Inverse GF(2^{1132})	14666 ns	6518 ns	2.25

According to Table 5.7, on average, our work is 2.6 times faster than McCusker's work [30]. We also estimated latency of our implementation based on the model proposed by Keller [24]. We compare the estimated latencies with Keller's [24] and McCusker's work [30] and show the results in Table 5.8. This is 2.5 times faster than that in Keller's work (1.48 ms), and 1.4 times faster than that in McCusker's work (0.84 ms). In addition, the latency of our FPGA based implementation is 152 times faster than that of software implementation (90 ms).

Table 5.8. Comparisons of Estimated Latency of Tate Pairing

	Latency	Speedup over software
[24]	1.48	60
[30]	0.84 ms	107
Our work	0.59 ms	152

CHAPTER 6

CONCLUSION AND FUTURE WORK

Elliptic curve cryptography and Tate pairing are new techniques in public key cryptography. ECC is an efficient substitution to RSA, which is one of the most commonly used public key cryptography schemes nowadays. ECC has shorter key sizes with the same security level compared to RSA. Shorter key sizes will save power, bandwidth, and improve performance. Tate pairing is used for identity based cryptosystem. Using Tate pairing, the public key can be derived from the identity information without issuing certificates.

Both ECC and Tate pairing are built upon Galois field arithmetics. Because they are very computationally expensive, we used FPGA to implement them. Through parallel processing and resource sharing, the performances are improved greatly compared to their software implementations. FPGA also has the advantage of flexibility compared to traditional ASIC implementations.

In this thesis, we study hardware implementation of elliptic curve point multiplication to speedup secure web transactions. We propose an FPGA based implementation in GF(2^{283}) optimizing the dataflow due to data dependency. Our implementation is significantly faster than those previous works presented in the literature. We also compared the FPGA implementation with its software implementation. The experimental results show that the hardware based implementation can improve the latency by a factor of 31.

In addition to ECC, we also explored hardware implementation of Tate pairing. We proposed an FPGA based implementation for Tate pairing computation in GF(2^{283}). Experimental results show that, on average, the arithmetic units in our work run 2.6 times faster than in previous work [30]. The estimated latency of our implementation is 1.4 times faster than that in [30], and 2.5 times faster than that in [24]. We also compared the

FPGA implementation with its software implementation. The results show that the FPGA implementation can speedup the latency by a factor of 152.

In this research, all of our designs are implemented using Verilog and synthesized using Xilinx ISE tool. Our target device is Xilinx Virtex 4 XC4VFX140-FF1517-11. We also performed all the simulations using Modelsim.

In our future research, we have several directions: Firstly, we will download our synthesized netlists onto FPGA chips and measure performance in real hardware environment. Secondly, we plan to integrate our designs into real applications such as VoIP, RFID and so on. Thirdly, we plan to extend our current cryptosystem into prime field and arbitrary key sizes. Finally, we plan to design a hybrid processor which combines both ECC and Tate pairing together. The hybrid processor can be reconfigured to realize specific cryptography schemes such as elliptic curve digital signature, identity based encryption. FPGA's flexibility makes it possible to design this kind of reconfigurable hybrid cryptosystem.

BIBLIOGRAPHY

[1] *ECC programming*, `http://rooster.stanford.edu/~ben/maths/ep/tate.html`.

[2] *LiDIA - a library for computational number theory*, `http://www.cdc.informatik.tu-darmstadt.de/TI/LiDIA/`.

[3] *Quarterly retail e-commerce sales*, `http://www.census.gov/mrts/www/data/html/06Q4.html`.

[4] B. Ansari and M. Anwar Hasan, *High performance architecture of elliptic curve scalar multiplication*, Tech. Report CACR, 2006.

[5] Paulo S. L. M. Barreto, Hae Yong Kim, Ben Lynn, and Michael Scott, *Efficient algorithms for pairing-based cryptosystems*, CRYPTO '02: Proceedings of the 22nd Annual International Cryptology Conference on Advances in Cryptology (London, UK), Springer-Verlag, 2002, pp. 354–368.

[6] Paulo S. L. M. Barreto, Ben Lynn, and Michael Scott, *Efficient implementation of pairing-based cryptosystems*, Journal of Cryptology 17 (2004), no. 4, 321–334.

[7] P.S.L.M. Barreto, S. Galbraith, and M. Scott, *Efficient pairing computation on supersingular abelian varieties*, Cryptology ePrint Archive, 2004.

[8] D. Boneh and M. Franklin, *Identity-based encryption from the /Weil pairing*, 21st Annual International Cryptology Conference, 2001, pp. 213–229.

[9] C. Coarfa, P. Druschel, and D. Wallach, *Performance analysis of tls web servers*, Network and Distributed Systems Security Symposium, 2002.

[10] Guerric Meurice de Dormale and Jean-Jacques Quisquater, *High-speed hardware implementations of elliptic curve cryptography: A survey*, J. Syst. Archit. 53 (2007), no. 2-3, 72–84.

[11] H. Eberle, N. Gura, and S. Chang-Shantz, *A cryptographic processor for arbitrary elliptic curves over GF(2^m)*, Application-Specific Systems, Architectures and Processors (ASAP) (2003), 98–10.

[12] M. Ernst, S. Klupsch, O. Hauck, and S.A. Huss, *Rapid prototyping for hardware accelerated elliptic curve public-key cryptosystems*, rsp 00 (2001), 0024.

[13] Gerhard Frey and Hans-Georg Ruck, *A remark concerning m-divisibility and the discrete logarithm in the divisor class group of curves*, Mathematics of Computation 62 (1994), no. 206, 865–874.

[14] Steven D. Galbraith, Keith Harrison, and David Soldera, *Implementing the tate pairing*, ANTS-V: Proceedings of the 5th International Symposium on Algorithmic Number Theory, Springer-Verlag, 2002, pp. 324–337.

[15] C. Grabbe, M. Bednara, J. von zur Gathen, J. Shokrollahi, and J. Teich, *A high performance VLIW processor for finite field arithmetic*, IPDPS '03: Proceedings of the 17th International Symposium on Parallel and Distributed Processing (Washington, DC, USA), IEEE Computer Society, 2003.

[16] J. Guajardo and C. Paar, *Itoh-Tsujii inversion in standard basis and its application in cryptography and codes*, Designs, Codes and Cryptography 25 (2002), no. 2, 207–219.

[17] V. Gupta, S. Gupta, S. Chang, and D. Stebila, *Performance analysis of elliptic curve cryptography for SSL*, 3rd ACM workshop on wireless security, 2002, pp. 87–94.

[18] Vipul Gupta, Douglas Stebila, Stephen Fung, Sheueling Chang Shantz, Nils Gura, and Hans Eberle, *Speeding up secure web transactions using elliptic curve cryptography.*, he 11th Annual Network and Distributed System Security (NDSS) Symposium, 2004, pp. 87–94.

[19] Darrel Hankerson, Alfred Menezes, and Scott Vanstone, *Guide to elliptic curve cryptography*, Springer, 175 Fifth Avenue, New York, NY, 2004.

[20] Toshiya Itoh and Shigeo Tsujii, *A fast algorithm for computing multiplicative inverses in GF(2^m) using normal bases*, Information and Computation 78 (1988), no. 3, 171–177.

[21] K. Jarvinen, M. Tommiska, and J. Skytta, *A scalable architecture for elliptic curve point multiplication*, IEEE Field Programmable Technology (FPT), 2004, pp. 303–306.

[22] A. Joux, *A one round protocol for tripartite Diffie-Hellman*, Algorithmic Number Theory Symposium, 2000, pp. 385–394.

[23] A. Karatsuba and Y. Ofman, *Multiplication on many-digital numbers by automatic computers*, Translation in Physics-Doklady 7 (1963), 595–596.

[24] M. Keller, T. Kerins, and W. Marnane, *FPGA implementation of a GF(2^{4m}) multiplier for use in pairing based cryptosystems*, Field Programmable Logic and Applications, 2005, pp. 594– 597.

[25] N. Koblitz, *Elliptic curve cryptosystems*, Mathematics of Computation 48 (1987), 203–209.

[26] S. Kwon, *Efficient tate pairing computation for elliptic curves over binary fields*, Australasian Conference on Information Security and Privacy, 2005, pp. 134–145.

[27] K.H. Leung, K.W. Ma, W.K. Wong, and P.H.W. Leong, *FPGA implementation of a microcoded elliptic curve cryptographic processor*, IEEE Symposium on Field-Programmable Custom Computing Machines, 2000.

[28] Julio Lopez and Ricardo Dahab, *Fast multiplication on elliptic curves over GF(2^m) without precomputation*, Cryptographic Hardware and Embedded Systems, no. Generators, 1999, pp. 316–327.

[29] D.J. Malan, M. Welsh, and M.D. Smith, *A public-key infrastructure for key distribution in tinyos based on elliptic curve cryptography*, First Annual IEEE Communications Society Conference on Sensor and Ad Hoc Communications and Networks, 2004, pp. 71–80.

[30] Kealan McCusker, Noel Connor, and D. Diamond, *Low-energy finite field arithmetic primitives for implementing security in wireless sensor networks*, International Conference on Communications, Circuits And Systems, 2006, pp. 1537–1541.

[31] Alfred Menezes, Scott Vanstone, and Tatsuaki Okamoto, *Reducing elliptic curve logarithms to logarithms in a finite field*, STOC '91: Proceedings of the twenty-third annual ACM symposium on Theory of computing, ACM Press, 1991, pp. 80–89.

[32] V. S. Miller, *Use of elliptic curves in cryptography*, Advances in Cryptology 218 (1985), 417–426.

[33] V.S. Miller, *Short programs for functions on curves*, Unpublished Manuscript, 1986.

[34] M. Morales-Sandoval and C. Feregrino-Uribe, *On the hardware design of an elliptic curve cryptosystem*, Proceedings of the Fifth Mexican International Conference in Computer Science, October 2004, pp. 60–70.

[35] Gerardo Orlando and Christof Paar, *A high-performance reconfigurable elliptic curve processor for GF(2^m)*, Second International Workshop on Cryptographic Hardware and Embedded Systems - CHES 2000 (Worcester, MA, USA), 2000, pp. 41–56.

[36] C. Paar, P. Fleischmann, and P. Roelse, *Efficient multiplier architectures for galois fields GF(2^{4n})*, IEEE Transaction on Computing 47 (1998), 162–170.

[37] K. G. Paterson, *Cryptography from pairings: A snapshot of current research*, Information Security Technical Report 7 (2002), 41–54.

[38] K.G. Paterson, *ID-based signatures from pairings on elliptic curves*, Electronic Letters, 2002, pp. 1025–1026.

[39] C. P. Pfleeger and S. L. Pfleeger, *Security in computing*, Prentice-Hall, Upper Saddle River, New Jersey, 2006.

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

[41] M. C. Rosner, *Elliptic Curve Cryptosystems on Reconfigurable Hardware*, Master's thesis, Dept. of Electrical and Computer Engineering, Worcester Polytechnic Institute, 1998.

[42] R. Sakai, K. Ohgishi, and M. Kasahara, *Cryptosystems based on pairing*, Symposium on Cryptography and Information Security, 2000.

[43] Nazar A. Saqib, Francisco Rodriguez-Henriquez, and Arturo Diaz-Perez, *A parallel architecture for fast computation of elliptic curve scalar multiplication over GF(2^m)*, ipdps 04 (2004), 144a.

[44] Akashi Satoh and Kohji Takano, *A scalable dual-field elliptic curve cryptographic processor*, IEEE Transactions on Computers 52 (2003), no. 4, 449–460.

[45] C. Shu, *Hardware Architectures of Elliptic Curve based Cryptosystems over Binary Fields. PhD Thesis*, Master's thesis, Dept. of Electrical and Computer Engineering, George Mason University, 1998.

[46] C. Shu, K. Gaj, and T. El-Ghazawi, *Low latency elliptic curve cryptography accelerators for NIST curves on binary fields*, IEEE Field-Programmable Technology (FPT), 2005, pp. 309–310.

[47] Chang Shu, Soonhak Kwon, and Kris Gaj, *FPGA accelerated Tate pairing based cryptosystems over binary fields*, IEEE International Conference on Field Programmable Technology, Dec. 2006.

[48] L. Song and K. Parhi, *Low energy digit-serial/parallel finite field multipliers*, Kluwer Journal of VLSI Signal Processing Systems 19 (1998), 149–166.

[49] Fabio Sozzani, Guido Bertoni, Stefano Turcato, and Luca Breveglieri, *A parallelized design for an elliptic curve cryptosystem coprocessor*, ITCC '05: Proceedings of the International Conference on Information Technology: Coding and Computing (ITCC'05) - Volume I (Washington, DC, USA), IEEE Computer Society, 2005, pp. 626–630.

[50] Lincoln D. Stein, *Web security*, Addison-Wesley, Reading, Massachusetts, 1997.

[51] Haodong Wang, Bo Sheng, and Qun Li, *Elliptic curve cryptography based access control in sensor networks*, International Journal of Sensor Networks 1 (2006), no. 2.

[52] Huapeng Wu, *Bit-parallel finite field multiplier and squarer using polynomial basis*, IEEE Trans. Comput. 51 (2002), no. 7, 750–758.