A Java Based Software Solution For Efficient Pairing Cryptography

Yejun Zou

NUI MAYNOOTH

Ollscoil na hÉireann Má Nuad

Department of Computer Science

National University of Ireland, Maynooth

Co. Kildare

Ireland

This thesis submitted in partial fulfilment of the requirements for the M.Sc Degree in Software Engineering

Supervisor: Dr. Joe Timoney

October 2010

Declaration

I hereby certify that this material, which I now submit for assessment on the program of study leading to the award of Master of Science in Computer Science by Research, is entirely my own work and has not been taken from the work of others save and to the extent that such work has been cited and acknowledged within the text of my work.

Signed:_____ Date: _____

Abstract

This thesis is concerned with applying software Engineering techniques to pairing based cryptographic systems. In particular we evolve our existing cryptographic system to incorporate new cryptographic concepts that strengthen the system. We discuss the design approach taken to allow an advanced mathematically based cryptographic concept to be developed in a maintainable manner. We present the integration process and evolutionary impacts on the existing system. We provide some testing data on the resulting system and give an indication of its performance. The language chosen is Java and the objective is that the Java developer can easily use the resulting system with minimal knowledge of the underlying machinery. Specifically, we present, implement, and evaluate alternative approaches to the use of a standard implementation of Tate pairing in a Java-based biometric identity verification tool.

Acknowledgement

With my deepest affections and appreciations I would like to thank my supervisor, Dr. Joe Timoney, for his guidance, support and encouragement throughout this project. I also would like to thank IRCSET, Claude Shannon Institute and the computer science in NUI Maynooth for their financial and technical support in the past three years. Finally many thanks to my family and my friends for their support and encouragement.

Contents

List of Figures **8**

List of Tables **10**

1 Introduction **11**

1.1 Overview 11

1.2 What is Cryptography 12

1.2.1 Terminology in Cryptography [39] 13

1.2.2 Well known Ciphers 13

1.2.3 Cryptographic Attacks 14

1.3 Pairing Based Cryptography on Elliptic Curves 15

1.4 Security of Pairing Based Cryptography 18

1.5 Software & Cryptography 19

1.6 Motivation of this project 21

1.7 Outline of The Dissertation 21

2 Mathematical Background **23**

2.1 Preliminaries 23

2.1.1 Tate Pairing 24

2.1.2 Miller's algorithm for Tate Pairing 25

2.2 Elliptic Nets Theory 26

2.2.1 Elliptic Divisibility Sequence 26
2.2.2 Stange's Elliptic Net 27
2.2.3 Tate Pairing using Elliptic Net 30
2.2.4 Existing Approach 32
2.3 Suitable Curves for Tate Pairing 33
2.3.1 Supersingular Curves 33
2.3.2 Barreto-Naehrig Curves 34
2.4 Summary 37

3 Design **38**

3.1 Current State of IBS 38
3.2 Designing a New IBS 40
3.3 Design for new curves 41
3.4 Logical view of the Design 41
3.4.1 Logical View of existing APIs 41
3.4.2 Logical View of New APIs 43
3.5 Software Development Strategy 45
3.5.1 Waterfall Model 45
3.5.2 V-Model 46
3.5.3 Iterative and Incremental Development Model 47
3.5.4 Our approach of Development process 49
3.6 Summary 51

4 Development **52**

4.1 Analysis of existing Elliptic Curve and Pairing libraries 52
4.2 Development of the Elliptic Net System 53
4.2.1 Class EDS 56
4.2.2 Class Block 56

4.2.3 Class EllipticNet 56
4.2.4 Class TatePairingViaENet 58
4.3 Evolving the IBS system 60
4.3.1 Class blitz.curve.EllipticCurve 60
4.3.2 Class blitz.Field.Extension.Fp2 61
4.3.3 *nonResidue* in class csi.crypto.EllipticNets.EllipticNet 63
4.4 Adding BN-Curves 64
4.4.1 Field extension 64
4.4.2 Curve Generation 66
4.4.3 Tate Pairing over BN-Curves 76
4.5 Complete System 80
4.6 Summary 81

5 Testing and Results **82**
5.1 Portability and Pre-settings 82
5.2 User Test Cases 83
5.2.1 Test cases of EDS and EllipticNet 83
5.2.2 Test Cases of Tate Pairing Via Elliptic Nets algorithm 87
5.2.3 Test Cases with a Random Input Value 92
5.2.4 Condition Testing (White box testing) 96
5.3 Performance Test 99
5.3.1 Comparison over supersingular curves 100
5.3.2 Comparison over BN-Curves 105
5.4 Summary 106

6 Conclusion **107**
6.1 Summary 107

6.2 Future Work 108

List of Figures

1.1 Diagram of Biometric Identity Based Signature Scheme [13] . 20

2.1 Example of Elliptic Net in [58] 29
2.2 Doubling and Double-adding a Block Centered on $W(k, 0)$ . . 30

3.1 Tate Pairing inside IBS 39
3.2 Desired New Tate Pairing inside IBS 40
3.3 Package View of Existing Blitz in [12] 42
3.4 Logical View of the Existing APIs 43
3.5 Logical View of New APIs 44
3.6 Waterfall Model of Software Development in [56] 45
3.7 V-Model of Software Development in [56] 47
3.8 Iterative and Incremental Model of Software Development in [24] 48
3.9 Iterations of the Project 49

4.1 Package Diagram of Elliptic Nets System 54
4.2 Class Diagram of EllipticNets package 55
4.3 Flow of Tate Pairing Computation 59
4.4 Class Diagram of Fp12 and Fp12Element 65
4.5 Class Diagram of CurveGen, BNCurve and TwistedCurve . . 67

4.6 Flow of Generation of Random Point R on $E' : y^2 = x^3 + B$. 73
4.7 Comparison of the Two TatePairing Classes 77
4.8 System View 80

5.1 Control Flow Graph of TatePairing(int bits, int curveType) Generated by Visustin [1] 98
5.2 Comparison of Computation Time for Miller's Algorithm with the Elliptic Nets Algorithm on Supersingular Curves as the Length of p is increasing 103
5.3 The Relationship between the Length of P and the Difference of the Time Cost between the Two Algorithms 104
5.4 Comparison of Computation Time for Miller's Algorithm with the Elliptic Nets Algorithm on BN-Curves as the Length of p is increasing 105

List of Tables

1.1 Key-size Equivalence in [27] 17

1.2 Bit sizes of curve parameters and corresponding embdding degree to obtain commonly desired levels of security [41] . . . 18

5.1 Test case of class EDS 85

5.2 Test case of class EllipticNet 86

5.3 Test case of Elliptic Nets Algorithm 1 87

5.4 Test case of Elliptic Nets Algorithm 2 88

5.5 Test case of Elliptic Nets Algorithm 3 89

5.6 Test case of Elliptic Nets Algorithm 4 90

5.7 Test case of Elliptic Nets Algorithm 5 91

5.8 Test case of Random Value for Supersingular Curves 93

5.9 Test Result for TestCase ID: 08 94

5.10 Test case of Random Values for BN-Curves 95

5.11 Test Result for TestCase ID: 09 95

5.12 Test cases for *TatePairing(int bitLength, int curveType)* . . . 99

5.13 Test Result for Test Cases in Table 5.12 99

5.14 Raw Benchmark for Supersingular Curve 103

5.15 Raw Benchmark for BN-Curves 105

Chapter 1

Introduction

1.1 Overview

In 2007, Burnett, Byrne, Dowling and Duffy [13] introduced a Java based IBS application with Tate pairing as the core encryption technique. They described an identity based signature scheme that uses biometric information, i.e. a user's fingerprint, to construct the public key. Their scheme is beneficial in many repudiation situations, for example a legal dispute over whether a contract had been signed or not by a user. A biometric reading provided by the alleged signer would be enough to verify the signature. Thus, their schemes involves biometric data extractions and an identity based signature scheme which employs the biometric data for user keys. The Tate pairing in their system is calculated with Miller's algorithm [38]. This is accepted to be the primary algorithm for pairing computation. However, Stange introduced Elliptic Net theory with its application in the Tate pairing computation [58]. Thus, Tate pairing can be obtained from either algorithm. Moreover, Barreto and Naehrig introduced new pairing friendly curves, known as BN-Curves ([3]), that were intended to enhance the security of pairing based cryptogra-

phy.

Thus, the goal of this thesis is to apply a software engineering approach to the design, implementation and testing of a practical paring based cryptosystem that is founded on the theory of Elliptic nets. Its contribution is to implement a Java Elliptic Nets API, to modify the existing BIO-IBS system [13], to compute the Tate pairing through both Miller's algorithm and the Elliptic Nets algorithm, to implement a more secure type of curves, and to compare the two Tate pairing computation algorithms for performance at different security levels with the two types of curves. This is the first Java structured implementation of Elliptic Nets and the first system to offer developers a choice of algorithm in the Tate pairing calculation.

1.2 What is Cryptography

The word **Cryptography** comes from Greek "$Krytó$"(hidden) and "$grápho$"(to write) [39]. It is the science of hiding the meaning of information. Generally speaking, it can be synonymous with the conversion of information. It is usually applied to avoid unwanted people reading the information. Prior to the early 20^{th} century, cryptography was chiefly concerned with linguistic and lexicographic patterns. Since then cryptography intersects the disciplines of mathematics, computer science and engineering, derived using mathematical algorithms and implemented using software that runs on computers or embedded processors. These new forms of cryptography are strongly driven by rapid advances in computer communications technologies. Cryptography is becoming necessary when sensitive data is being transacted over any untrusted medium. It provides the services such as keeping secrets from an unexpected audience, authentication with a signature, verification of data integrity, and security certificates for the communications.

1.2.1 Terminology in Cryptography [39]

- Cipher: procedure to render messages unintelligible except to an authorized recipient;
- Encryption: process to convert original message to unintelligible message;
- Decryption: process to recover the original message;
- Plaintext: original readable message;
- Ciphertext: encrypted message;

1.2.2 Well known Ciphers

Modern cryptography can be categorized into symmetric ciphers, asymmetric ciphers and hash functions according to the number of keys. The symmetric cipher only has one private key and this key is used for both encryption and decryption. The examples of symmetric ciphers include DES(Data Encryption Standard), triple-DES, AES(Advanced Encryption Standard), CAST-128, CAST-256, One-time Pad, RC4, DES-X, IDEA(International Data Encryption Algorithm) [39]. The asymmetric cipher, also known as public key cryptography (PKC), involves two keys: a private key for decryption and a public key for encryption. The well-known asymmetric ciphers are El Gamal, RSA, Elliptic Curve Cryptography(ECC), McEliece and NTRUEncypt [39]. The cryptographic hash functions, also called message digests, are often used to encrypt passwords and provide a measure of the data integrity. The hash functions in common use today include MD5, SHA1, SHA-256, SHA-512 and

RIPEMD [39]. Applications in the real world could use all the three cryptographic techniques for secure communication, or depending just one or two together.

1.2.3 Cryptographic Attacks

Cryptography has been applied for secure communications through the various techniques as mentioned in Section 1.2.2. However, there are some people that try to extract the information by attacking the cryptographic system. These cryptographic attacks circumvent the security of the cryptographic systems by finding weaknesses of the ciphers or the cryptographic schemes. Such a process is also called "cryptanalysis". In [39], cryptographic attacks can be classified into six related types including three plaintext based attacks and three ciphertext based attacks. They are: **K**nown **P**laintext Attacks, **C**hosen **P**laintext Attacks, **A**daptive **C**hosen **P**laintext Attacks, **C**iphertext **O**nly Attack, **C**hosen **C**iphertext Attacks, and **A**daptive **C**hosen **C**iphertext Attacks. For example, a Brute Force Attack, which systematically attempts every possible key to unlock the message, is used in a **K**nown **P**laintext or **C**iphertext **O**nly **a**ttack; The Meet-In-The-Middle Attack [39] is also a **K**nown **P**laintext **a**ttack. It is a passive attack in that the attacker can read the message without authorization but cannot alter the message or replace the message entirely. It can be used successfully against the DES, and this is why Triple DES is sometimes used [14]. The **B**irthday **A**ttack is a **C**hosen **C**iphertext Attack that can discover collisions in hashing algorithms such as MD5 and SHA1 ([37, 44]). A 256-bit Hash is needed to give a 2^{128} resistance to the Birthday attack [44]. Recently, the **S**ide **C**hannel Attack has become popular as it leverages additional information based on the physical implementation of a cryptographic algorithm, including the hardware used,

to encrypt or decrypt data([55, 33, 60]).

In the 1990's, many cryptographic schemes were based on the Discrete Logarithm Problem (DLP)([62]) which is presumed to be a hard mathematical problem, and thus it is the basis of new cryptography schemes such as El Gama and ECC mentioned in Section 1.2.2. Pairing was shown to attack such schemes successfully [36, 23]. In [36], Menezes, Vanstone, and Okamoto proved and used the Weil pairing to reveal the weakness of supersingular curves (see Section 2.3.1). Later on, Frey and Ruck published their attack (FR attack) with Tate pairing in 1994 [23] to break the DLP-based cryptography. This drove a new need for more complex cryptographic schemes. However, for implementation purposes, they need to be efficient. Otherwise, a trade-off between efficiency and security is required.

1.3 Pairing Based Cryptography on Elliptic Curves

In reverse to pairing based attacks, pairing is also useful for designing complex cryptographic schemes, particularly in pairing-based elliptic curve cryptography [34]. This is a new asymmetric cipher technique and it has exploded over the past six years [20]. The central idea is the construction of a mapping between two useful cryptographic groups: G_1 and G_2 which allows for cryptographic schemes based on the reduction of one problem in one group to a different, usually easier problem in the other group. Such a mapping e is described below:

$$e : G_1 \times G_1 \rightarrow G_2$$

where e is supposed to be a bilinear mapping, which means

$$\forall P, Q \in G_1 \ \ and \ \ \forall a, b \in \mathbb{Z}_q^*, e(aP, bQ) = e(P, Q)^{ab}.$$

The bilinearity allows pairings such as the Weil Pairing and Tate Pairing to be useful because it enables new identity-based cryptographic primitives. Identity-based (also known as ID-based) crypto schemes have the advantage that there is an explicit connection between a user's unique identification, such as an e-mail address or biometric measurement, and their private key. This eliminates the need for a public key distribution infrastructure. The authenticity of the public keys is guaranteed implicitly as soon as the transport of the private keys to the corresponding user is kept secure. It also allows extra embedding data, such as an expiration date for a message, coded as part of a user ID in the system. Joux firstly introduced a pairing based one-round three-party key exchange in 2000 [2]. In 2001, Boneh and Franklin published the first ID-based encryption (IBE) scheme [7]. Since then there have been many approaches to ID-based cryptography such as [9, 8, 63, 43]. Particularly in 2004, the Java based approaches of IBE and IBS were introduced in [47, 19, 13].

The security of the pairing based cryptography is based on the assumption that the Decision Diffie-Hellman (DDH) problem [10] is easily solved with a pairing function but the Computational Diffie-Hellman (CDH) problem remains infeasible.

In short, the DDH can be described as:

Given $\langle P, aP, bP, cP \rangle$ with $a, b, c \in_R \mathbb{Z}_q^*$, and P is affine point on elliptic curve, then determine whether $c = ab$. This can be solved easily by defining pairing functions: $e_1 = (aP, bP), e_2 = (P, cP)$ and if $e_1 = e_2$, then $c = ab$

due to the bilinearity.

The CDH can be expressed as:

Given $\langle P, aP, bP \rangle$ with $a, b \in_R \mathbb{Z}_q^*$, and P is affine point on elliptic curve, then find $c \in_R \mathbb{Z}_q^*$ such that $c = ab$. This is hard to achieve as it is equivalent to solving the DLP.

In the pairing-based cryptography, Tate pairing is particularly useful due to its properties (see Section 2.1.1) and the efficient manner in which it can be evaluated. This was mainly due to the algorithm by Miller [38] that rapidly computes multiples of points on elliptic curves.

The rich mathematical structure of pairing also ensures strong security for relatively small key sizes compared to more traditional systems like RSA [50]. In [27], the ECRYPT II (European Network of Excellence in Cryptology) provides the latest key size equivalences shown in Table 1.1. As the pairing is defined with elliptic curves, the key size of pairing based cryptography can be considered as the same as the key size of ECC. One of the significant benefits of ECC is that ECC saves memory space as it can provide equivalent crypto strengths with smaller number of key sizes (in bits) compared with other cryptographic techniques such as RSA.

Security Level (bits)	**RSA (bits)**	**DLP**		**ECC (bits)**	**Hash (bits)**
		key	group		
64	816	128	816	128	128
72	1008	144	1008	144	144
80	1248	160	1248	160	160
96	1776	192	1776	192	192
112	2432	224	2432	224	224
128	3248	256	3248	256	256
160	5312	320	5312	320	320
192	7936	384	7936	384	384
256	15424	512	15424	512	512

Table 1.1: Key-size Equivalence in [27]

1.4 Security of Pairing Based Cryptography

The security of a pairing based cryptosystem relies on two parameters: the bit length, r and the bit size of the extension field $k \log_2 n$, where k is the embedding degree and p is the number of elements in the finite field. The embedding degree is the degree of the extension field that the pairing maps into. The parameters need to be chosen high enough so that the discrete logarithm problem is hard in both the subgroup of the curve and the finite field [41]. An Elliptic curve with a small embedding degree and a large prime-order subgroup is said to be pairing friendly. According to [22], much work has been done trying to match the bit sizes of curve parameters to obtain commonly desired levels of security. Table 1.2 from [22] shows the size of bit curve parameters and corresponding embedding degrees to obtain commonly desired levels of security, noting that

$$\rho = \log p / \log r$$

Security Level (in bits)	Subgroup size r (in bits)	Extension field size q^k (in bits)	Emgedding degree k	
			$\rho \approx 1$	$\rho \approx 2$
80	160	960-1280	6-8	2*,3-4
112	224	2200-3600	10-16	5-8
128	256	3000-5000	12-20	6-10
192	384	800-10000	20-26	10-13
256	512	14000-18000	28-36	14-18

Table 1.2: Bit sizes of curve parameters and corresponding embdding degree to obtain commonly desired levels of security [41]

In general, for efficient pairing computation we need curves with embedding degree rather small. However, to improve security it is more efficient to have a greater value of k than p [41].

1.5 Software & Cryptography

Most cryptographic schemes are implemented as software programs. Well-known examples include PGP[17] and NTRUEncrypt[25]. Biometrics cryptography processing normally consists of a hardware interface but the processing is either done on a computer or embedded processor. There are some existing Java based software solutions to the cryptography. *Sun* provides security services and utilities since J2SE 1.4.2, which includes the most common hash functions, symmetric and asymmetric ciphers. Up to their latest JDK 1.6.21, the java.security package with its sub-packages and together with the javax.security.* packages could provide most popular security services including digital certificates, digital signatures, public key cryptography, and authentication [46]. [26] is another well-known Java based security provider. Their products are also free of charge for educational and research purposes. They provide ECC including the Elliptic Curve Diffie-Hellman protocol and Elliptic Curve Digital Signature protocol. These are not suitable for our system specification as they are not suitable for identity based cryptographic scheme. We applied pairing to allow identity based cryptographic scheme. Since the pairing computation is still very timing consuming compared with other ciphers, there is no official or commercially released library in this area.

The cryptographic scheme in [13] was the first java approach for pairing based cryptography as mentioned in Section 1.1. The following Figure 1.1 shows the system structure, which includes four main stages named Biometric Extraction, Fuzzy Extraction, Parameter Selection, and IBS system [13].

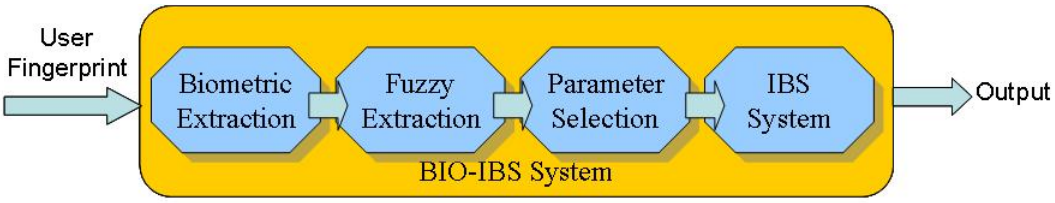

Figure 1.1: Diagram of Biometric Identity Based Signature Scheme [13]

The Biometric Extraction block provides the biometric measurement. It reads the user's biometric information, such as their fingerprint, and translates it to a byte array. However, it must take into account the facts that either a biometric identity can vary over time or that the reading taken may not be perfect [29]. Thus, checking for an exact match with a previous measurement may not always work, and to overcome this some further processing is needed. The fuzzy extraction block is intended rectify this. The fuzzy extractor is based on the Hamming distance metric. It is built using Error Correcting Codes (ECC). The ECC chosen is a Bose-Chaudhuri-Hocquenghem (BCH) code [13]. These can be designed to correct errors to about half the code's block length. Applying the fuzzy extractor means that variable biometrics can still reproduce unique keys [28]. Following the fuzzy extraction, the Parameter Selection and IBS stages takes place. Elliptic curve cryptography with Tate pairing is used for the encoding. The Tate pairing in this system is calculated using the original Miller's algorithm [38]. There are two possible configurations here for these blocks: (1) Signing or (2) Verification. In the case of signing two parameters are produced: ID and PAR, where ID is the identity used for the key generation and PAR is the publicly available reproduction parameter. This PAR will be used in conjunction with another biometric input in the verification process to recover ID [29]. In the case of verification, a combination of the new biomet-

ric measurement and the PAR is used to generate an identification value, ID'. If this is equal to the stored ID, within some threshold, then the user is verified, otherwise they are rejected [13].

1.6 Motivation of this project

It was noticed that Tate pairing was very valuable to the IBS system. Here it was evaluated with Miller's algorithm (see Section 2.1.2). However, since Stange introduced the Elliptic Net theory in 2006 [58], Tate pairing can now be evaluated with an alternative algorithm. This new technique brings a fresh perspective to the java-based BIO-IBS system of Section 1.5. In general, these advances have given rise to new requirements for the system. The pertinent questions to ascertain these are:

1. How can the Elliptic Net algorithm be integrated with the BIO-IBS system?

2. Can the performance of the whole system be at least maintained or can it be improved?

1.7 Outline of The Dissertation

The structure of this dissertation is as follows: In Chapter 2 the mathematical concepts involved in this thesis are outlined. It will cover the areas of finite field and its extension fields, elliptic curve arithmetic, Tate pairing, Elliptic Nets theory, and discuss two types of pairing friendly elliptic curves. Chapter 3 provides the initial design of the project. Chapter 4 covers the development procedure of the Elliptic Nets algorithm, which includes

the issues involved in integrating the Elliptic Nets system with our existing pairing based cryptosystem and adding BN-Curves for Tate Pairing in both algorithms. Chapter 5 will address all the questions of the Section 1.6. It will give some test cases with results and provides a comparison of the two approaches to Tate calculation to assess performance. Finally, we discuss conclusions and future work in Chapter 6.

Chapter 2

Mathematical Background

This chapter gives a flavour of the mathematical theory behind the system. Section 2.1 explains the fundamental arithmetic of Elliptic curves and defines Tate Pairing. Section 2.2 introduces Elliptic Divisibility Sequence(EDS), Elliptic Net and Tate pairing calculation via Elliptic Nets. Section 2.3 introduces two types of elliptic curves which can be candidates for pairing computation. These two curves are chosen as they benefit from efficiency.

2.1 Preliminaries

We represent the finite field with p elements as

$$\mathbb{F}_p = \{0, 1, 2, 3, \ldots, p-2, p-1\}.$$

An integer r is called a quadratic residue modulo p if it is congruent to a perfect square (mod p); i.e., if there exists an integer x such that: $x^2 \equiv r$ (mod p). Otherwise, r is called a quadratic nonresidue (mod p). Then, the extension field $\mathbb{F}_{p^2}$ is defined as $\{a + bi\}$ with $a, b \in \mathbb{F}_p$ and i^2 is a quadratic nonresidue modulo p([62]).

The basic units for elliptic curve arithmetic are points (x, y) on an elliptic curve, E, over a finite field, $\mathbb{F}_p$, denoted $E(\mathbb{F}_p)$, of the form

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

with $a_1, a_2, a_3, a_4, a_6 \in \mathbb{F}_p$. The form above is general. It can be transformed into a Weierstrass form as

$$y^2 = x^3 + Ax + B \tag{2.2}$$

with $x, y, A, B \in \mathbb{F}_p$ by a suitable transformation. We can define the abstract concepts of addition, $P + Q$, and scalar multiplication by an integer, $[n]P$, on the points of $E(\mathbb{F}_p)$.

We can also define a special point at infinity, ∞, which is not a solution to the equation given above. These operations combine to make $E(\mathbb{F}_p)$ a finite abelian group with ∞ behaving as the identity element. Details of how these concepts are implemented appear in [5, 31]. The *order* of a point P is defined to be the smallest integer n such that $[n]P = \infty$. We let $E(\mathbb{F}_p)[n]$ be the subgroup of $E(\mathbb{F}_p)$ consisting of points of order n. This is also called the group of n-torsion points on $E(\mathbb{F}_p)$. Let $\#E$ be the order of the curve(i.e. the number of points on the curve), then for any point $P \in E$, $[\#E]P = \infty$.

2.1.1 Tate Pairing

As mentioned in Section 1.2.3, the Tate Pairing was introduced to cryptography by Frey and Ruck in [23].

Definition: Consider the elliptic curve $E(\mathbb{F}_p)[n]$, let k be a positive integer such that $n \mid p^k - 1$ with k minimal, if this is satisfied then k is called the

embedding degree. Let $\mu(n) = \{a \in \mathbb{F}_{p^k} \mid a^n = 1\}$ be the n^{th} roots of unity. Then according to Washington [62], the Tate pairing τ'_n can be defined as:

$$\tau'_n : E(\mathbb{F}_p)[n] \times E(\mathbb{F}_{p^k})/nE(\mathbb{F}_{p^k}) \rightarrow \mathbb{F}_{p^k}/(\mathbb{F}_{p^k})^n \tag{2.3}$$

and the modified Tate pairing τ_n is:

$$\tau_n : E(\mathbb{F}_p)[n] \times E(\mathbb{F}_{p^k})/nE(\mathbb{F}_{p^k}) \rightarrow \mu(n) \tag{2.4}$$

Note that the τ'_n in Equation 2.3 is also known as a coset value which represents a quotient group rather than an element in that group.
Assume point $P \in E(\mathbb{F}_p)[n]$ and point $Q \in E(\mathbb{F}_{p^k})$ with $P! = 0$, then Tate pairing can be denoted as $e_n(P, Q)$. In this dissertation, all P and Q satisfy this assumption. We choose Tate pairing for cryptography due to the following properties:

1. The Tate pairing is non-degenerated. This means for any given P, there always exists a Q such that $e_n(P, Q)! = 1$.

2. The Tate pairing is bilinear. This means for any given integers a, b and any points P and Q,

$$e_n(aP, bQ) = e_n(P, Q)^{ab} \tag{2.5}$$

2.1.2 Miller's algorithm for Tate Pairing

In 1986, Miller firstly found that pairings can be achieved through divisor theory [38]. This algorithm focuses on finding the principle divisor of P with some specific line functions. The algorithm is a computationally efficient approach for Tate pairing. In the recent years, variant versions of Miller's

algorithm were developed for the optimization and efficiency of Tate pairing computation [4, 54, 21, 18]. Despite all these optimizations, however, the time cost on pairing computation is still the most significant bottleneck of pairing based cryptography ([35]). Up until late 2006 Miller's algorithm was the only way to compute these multiples and so almost all applications, including the existing BIO-IBS [13, 19] system, were implemented with this algorithm.

2.2 Elliptic Nets Theory

The Elliptic Nets theory was proposed in [58] by Stange. The study is based on division polynomials of elliptic curves.

2.2.1 Elliptic Divisibility Sequence

Definition: A divisibility sequence, ψ, is an integer sequence that satisfies the following two properties:

1. For all positive integers $m > n$,

$$\psi_{m+n}\psi_{m-n} = \psi_{m+1}\psi_{m-1}\psi_n^2 - \psi_{n+1}\psi_{n-1}\psi_m^2 \tag{2.6}$$

2. ψ_n divides ψ_m whenever n divides m.

Elliptic Divisibility Sequence(EDS) was firstly defined by Morgan Wards in the 1940s [61]. More recently, R. Shipsey submitted her PhD thesis on EDS [53]. In her thesis, she discovered a specific relationship between elliptic divisibility sequences and elliptic curves. Consider a point $P = (x, y)$ and its multiples on an elliptic curve $E : y^2 = x^3 + Ax + B$ over a finite field $\mathbb{F}_p$ where p is prime. According to the division polynomial theory, $[n]P$ can be

represented as [62]:

$$[n]P = \left(\frac{\phi_n(x)}{\psi_n^2(x)}, \frac{\omega_n(x,y)}{\psi_n(x,y)^3} \right) \tag{2.7}$$

Regardless of the other parts, the square roots of the denominator of the x coordinates, $\psi_n(x)$, are division polynomials and they form the elliptic divisibility sequence. Shipsey illustrated initial formulas for calculating the first four terms in the sequence and two recursion formulas for any other remaining terms in the sequence. For example: define the Elliptic Curve as:

$$E : y^2 + y = x^3 + x^2 - 2x \ \ over \ \ \mathbb{F}_5$$

and $P = (0, 0)$ has order 9, then the EDS of the curve is:

$$0, 1, 1, 2, 1, 3, 4, 3, 2, 0, 3, 2, 1, 2, 4, 3, 4, 4, 0, 1, 1, 2, 1, 3, 4, 3, 2, 0, 3, 2, ...$$

Shipsey also found that EDS could be applied to cryptography, especially to the elliptic curve discrete logarithm problem (ECDLP). Later on in [32], Lauter and Stange defined hard problems with EDS terms which are equivalent to ECDLP and related one of them to Tate pairing and the MOV[36], Frey-Ruck[23] and Shipsey's attacks[53].

2.2.2 Stange's Elliptic Net

In late 2006 an alternative approach to computing fast multiples of points was introduced by Stange [58]. This new algorithm is based on the theory of division polynomials and is significantly different than Miller's algorithm. Basically Stange made the connection between terms in a sequence (called Elliptic Net) and multiples of points on an elliptic curve. This meant that by quickly moving up the sequence large multiples of points could be calculated.

Definition: The Elliptic Net is a generalization of the elliptic divisibility sequence. It is a function $W : A \to R$ from a finite rank free abelian group A to an integral domain R satisfying the properties:

1. $W(p+q+s)W(p-q)W(r+s)W(r)$

$$+W(q+r+s)W(q-r)W(p+s)W(p)$$

$$+W(r+p+s)W(r-p)W(q+s)W(q) = 0 \ ,$$

   for all p, q, r, $s \in A$.

2. $W(-z) = -W(z)$ for any $z \in A$.

For the rank 2 case, $W_{E,P,Q}$, denotes an Elliptic Net associated with two points, P and Q on the elliptic curve $E : y^2 = x^3 + Ax + B$ over a finite field $\mathbb{F}_p$ where p is prime and $p > 3$. The variable $W(n, m)$ represents the equivalent meaning of $[n]P+[m]Q$. Figure 2.1 gives an example of an Elliptic Net.

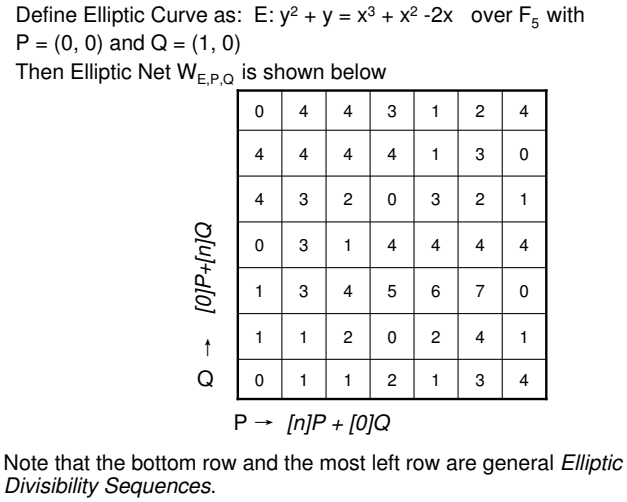

Figure 2.1: Example of Elliptic Net in [58]

In the figure, P and Q are incremented by scalar multiplications with the scalar in the range [0..6] and the values of the matrix are computed outward from the bottom left hand corner according to $W(n, m)$, where n and m also represent the indices of row and column respectively. For instance, $W(2, 1) = 2$ in this example.

Elliptic nets have two operations: doubling and double-adding. Figure 2.2 shows a block structure used for the computation and storage of an Elliptic Net.

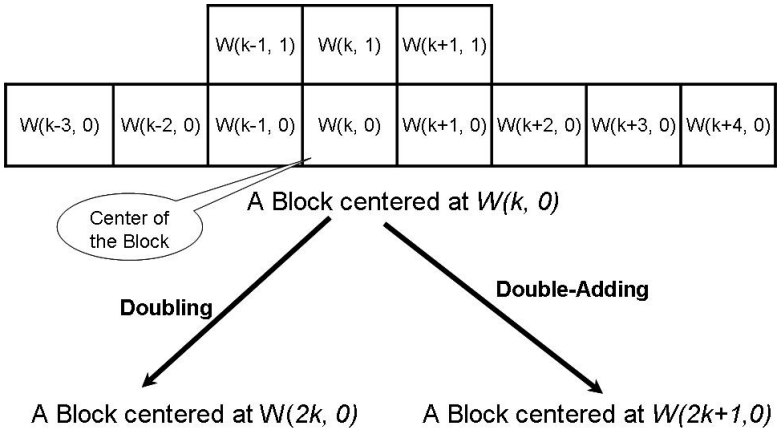

Figure 2.2: Doubling and Double-adding a Block Centered on $W(k, 0)$

The figure states that doubling a block centered on $W(k, 0)$ returns the block centered on $W(2k, 0)$, and double-adding a block centered on $W(k, 0)$ yields the block centered on $W(2k + 1, 0)$. The value of a $W(k, 0)$-centered blocked can be computed through the Elliptic Net algorithm and its corresponding doubling and double-adding strategy. They will be described in Section 2.2.3

2.2.3 Tate Pairing using Elliptic Net

Let E be an elliptic curve defined over a finite field K, m a positive integer, $P \in E(K)[m]$ and $Q \in E(K)$. If W_P is the Elliptic Net associated to E and P, then we can calculate the Tate Pairing as: [58]

$$\tau'(P, P) = \frac{W_P(m + 2)W_P(1)}{W_P(m + 1)W_P(2)} \tag{2.8}$$

Further, if $W_{P,Q}$ is associated to E, P and Q, then we have

$$\tau'(P, Q) = \frac{W_{P,Q}(m + 1, 1)W_{P,Q}(1, 0)}{W_{P,Q}(m + 1, 0)W_{P,Q}(1, 1)} \tag{2.9}$$

To summarize the connection between Elliptic Nets and the Tate pairing, from the equations above, the Tate pairing is calculated by certain terms in the Elliptic Net sequence. Note that the output of Elliptic Net algorithm is a coset value. However, it is still necessary to perform a final exponentiation as Miller's algorithm does, which means, [58]

$$\tau_m(P,Q) = e_m(P,Q) = \tau'(P,Q)^{\frac{(p^k-1)}{m}} \tag{2.10}$$

To compute terms in an Elliptic Net associated with the elliptic curve $E : y^2 = x^3 + Ax + B$, we assume that the points $P = (x_1, y_1) \in E[m]$ and $Q = (x_2, y_2) \in E(K)$, then we can initialize the following terms([58]):

$W(1,0) = W(0,1) = W(1,1) = 1$,
$W(2,0) = 2y_1$,
$W(3,0) = 3x_1^4 + 6Ax_1^2 + 12Bx_1 - A^2$,
$W(4,0) = 4y_1(x_1^6 + 5Ax_1^4 + 20Bx_1^3 - 5A^2x_1^2 - 4ABx_1 - 8B^2 - A^3)$,
$W(-1,1) = x_1 - x_2$,
$W(2,1) = 2x_1 + x_2 - \left(\frac{y_2-y_1}{x_2-x_1}\right)^2$,
$W(2,-1) = (y_1 + y_2)^2 - (2x_1 + x_2)(x_1 - x_2)^2$.

The rest of the terms in the Elliptic Net can be computed through the recursion shown below:

$$W(2k-1,0) = W(k+1,0)W(k-1,0)^3 - W(k-2,0)W(k,0)^3 \ ,$$
$$W(2k,0) = (W(k,0)W(k+2,0)W(k-1,0)^2 - W(k,0)W(k-2,0)W(k+1,0)^2)/W(2,0) \ ,$$
$$W(2k-1,1) = (W(k-1,1)W(k+1,1)W(k-1,0)^2 - W(k,0)W(k-2,0)W(k,1)^2)/W(1,1) \ ,$$
$$W(2k,1) = W(k-1,1)W(k+1,0)W(k,0)^2 - W(k-1,0)W(k+1,0)W(k,1)^2 \ ,$$

$$W(2k+1,1) = (W(k-1,1)W(k+1,1)W(k+1,0)^2 \\ -W(k,0)W(k+2,0)W(k,1)^2)/W(-1,1) \ ,$$
$$W(2k+2,1) = (W(k+1,0)W(k+3,0)W(k,1)^2 \\ -W(k-1,1)W(k+1,1)W(k+2,0)^2)/W(2,-1) \ ,$$

Then the Tate pairing τ'_m is given as $W(m+1,1)/W(m+1,0)$ where m is the order of P. This is a limited case of Equation 2.9 as $W(1,0) = W(1,1) = 1$. For the entire algorithm see Algorithm 1.

Algorithm 1 Tate Pairing via Elliptic Net Algorithm

Input: Initial terms $a = W(2,0)$, $b = W(3,0)$, $c = W(4,0)$, $d = W(2,1)$, $e = W(-1,1)$, $f = W(2,-1)$, and integer $m = (d_k d_{k-1} \dots d_1)_2$ with $d_k = 1$.
Output: Tate pairing $\tau(P,Q)$

1: $V \leftarrow [[1,1,d];[-a,-1,0,1,a,b,c,a^3c-b^3]]$
2: **for** $i = k-1$ to 1 **do**
3: **if** $d_i == 0$ **then**
4: $V \leftarrow$ doubling (V)
5: **else**
6: $V \leftarrow$ doubleAdding (V)
7: **end if**
8: **end for**
9: **return** $V[0,2]/V[1,4]$

2.2.4 Existing Approach

There are several existing implementations of the Elliptic Nets algorithm in a variety of computer languages. Stange implemented the original algorithm in PARI/GP [57]. Her implementation was initially used for testing and proving her Elliptic Nets algorithm. Stange also mentions that Michael Scott and Augusto Jun Devegili implemented a C++ version for a pairing-friendly curve of degree 2. She said that "Ben Lynn's C++ approach [34] is applicable to curves of various size and embedding degrees" in her paper as well. Graeme Taylor provided a SAGE version of the Elliptic Nets algorithm

[59]. He reduced the size of block structure (see Section 2.2.2) as a possible optimization. SAGE is a free open-source mathematics software and it can be directly used on a Unix platform . However, it requires an extra software tool, named *VirtualBox*([11]), to be run under the Microsoft Windows-based environment([16]).

2.3 Suitable Curves for Tate Pairing

By the definition of Tate pairing $\tau(P, Q)$, it requires the first point $P \in E(\mathbb{F}_p[n])$ and second point $Q \in E(\mathbb{F}_{p^k})/E(\mathbb{F}_p)$. However, a randomly-chosen elliptic curve will normally have a large embedding degree (which means large k), which intensifies its resistance to the MOV([36]) and Frey-Ruck([23]) attacks, but also renders it useless for pairing-based cryptography due to a low-speed and huge-cost computation. For practical purposes, we are looking for curves with small embedding degrees (i.e. small k) such that they guarantee the trade-off between efficiency and security. This leads to the use of supersingular curves to attain the balance.

2.3.1 Supersingular Curves

Supersingular curves have been proven to be the most efficient curves for pairing [35]. The curves we chose have the form $y^2 = x^3 + x$ over $\mathbb{F}_p$ where $p = 3(\text{ mod } 4)$. These curves also have the following properties:

1. $\#E = p + 1$.
2. For any odd $r|p + 1$ the group $E(\mathbb{F}_p)[r]$ has embedding degree $k = 2$.
3. There exists the distortion map:

$$\Psi(x, y) \mapsto (-x, iy) \tag{2.11}$$

where $i^2 = -1$. (Note -1 is quadratic nonresidue in $\mathbb{F}_p$.) It maps points of $E(\mathbb{F}_p)$ to points of $E(\mathbb{F}_{p^2})/E(\mathbb{F}_p)$.

Thus, the point Q can be easily obtained by getting a point in $E(\mathbb{F}_p)$ and then applying the distortion map. Based on the above advantages of supersingular curves, Tate pairing over these curves was studied in [4],[54] and several optimizations were discovered that could improve the efficiency of the pairing computation.

However, the small number of the order and the embedding degree of the supersingular curves result a small number of possible group structures, which enables many algorithms designed for attacking such curves. The result is a reduction in complexity to subexponential, and even polynomial time when they exploit this fact. Therefore, supersingular curves are considered to be 'weak' for cryptography [30].

2.3.2 Barreto-Naehrig Curves

Although supersingular curves have the benefit of speeding up of the computation of pairing, the weakness of theses curves is also obvious as in [52, 51, 30]. Therefore finding more secure curves becomes critical for pairing based cryptography. Generally a non-supersingular elliptic curve over $\mathbb{F}_p$ is called pairing-friendly if it contains a subgroup of order r with embedding degree k not too large. In 2005 Barreto and Naehrig [3] defined a new type of pairing-friendly curves which are known as BN-Curves. The BN-Curves are optimal elliptic curves of prime order and embedding degree $k = 12$.

Parameters of BN-Curves

BN-Curves have the form $y^2 = x^3 + b$ over $\mathbb{F}_p$ where $b \in \mathbb{F}_p$ and $p = 1($ mod 6). The order of the curve is denoted as n. Let point $g = (1, y)$ be the

generator of the curve. Then, a BN-Curve can be specified by the parameters of p, n, b, y. The Algorithm 2 gives an efficient construction of BN-Curves with a desired size([3]).

Algorithm 2 Constructing BN-Curves

Input: The approximate desired size m of the curve order (int bits)
Output: parameters p, n, b, y such that the curve $y^2 = x^3 + b$ has order n over $\mathbb{F}_p$ with point $g = (1, y)$ as the generator of the curve.

Let $P(x) \equiv 36x^4 + 36x^3 + 24x^2 + 6x + 1$
Computer she smallest $x \approx 2^{m/4}$ such that $\lceil log_2 P(-x) \rceil = m$.
loop
  $t \leftarrow 6x^2$
  $p \leftarrow P(-x)$
  $n \leftarrow p + 1 - t$
  if p and n are prime **then**
    exit loop
  end if
  $p \leftarrow P(x)$
  $n \leftarrow p + 1 - t$
  if p and n are prime **then**
    exit loop
  end if
  $x \leftarrow x + 1$
end loop
$b \leftarrow 0$
repeat
  repeat
    $b \leftarrow b + 1$
  until $b + 1$ is a quadratic residue mod p
  Compute y such that $y^2 = b + 1(\text{ mod } p)$
  $g \leftarrow (1, y)$ on the curve $E : y^2 = x^3 + b$
until $[n]g = \infty$
return p, n, b, y

Sextic Twisted Curves

As BN-Curves has $k = 12$, sextic twisted curves were also introduced to reduce the computations with BN-Curves. In [3], it was proven that when $p = 1(\bmod 6)$ there exists $\zeta \in \mathbb{F}_{p^2}^*$ such that $X^6 - \zeta$ is irreducible over $\mathbb{F}_{p^2}[X]$ and $\mathbb{F}_{p^{12}}$ can be represented as $\mathbb{F}_{p^2}[X]/[X^6 - \zeta]$. Therefore, a sextic twisted curve can be defined as $E'(\mathbb{F}_{p^2}) : y'^2 = x'^3 + b/\zeta$ with suitable ζ. This sextic twisted curve must satisfy:

1. $n|\#E'(\mathbb{F}_{p^2})$
2. $\#E'(\mathbb{F}_{p^2}) = n(2p - n)$

Let $z \in \mathbb{F}_{p^{12}}$ be a root of $X^6 - \zeta$. There exists a homomorphism Ψ for mapping points and it can be defined as:

$$\Psi : E'(\mathbb{F}_{p^2)} \rightarrow E(\mathbb{F}_{p^{12}})$$

$$(x', y') \mapsto (z^2 x', z^3 y') \tag{2.12}$$

Since that any element in $\mathbb{F}_{p^{12}}$ can be represented in the form

$$a_0 + a_1 z^1 + a_2 z^2 + a_3 z^3 + a_4 z^4 + a_5 z^5 \tag{2.13}$$

where $a_i \in \mathbb{F}_{p^2}$ with $z^6 = \zeta$.

The sextic twisted curve (in [3]) is useful because of two efficient improvements on BN-Curves. They are point compression and pairing compression. Basically, both compressions can be used to save memory space by about one third and in some cases even by one sixth. For a memory-limited device, these compressions are significant. However, such compressions require extra computations for implicit exponentiation, which directly slow down the

speed and increase the burden on the computing engine. The detail of the compressions were described in [3]. From a practical implementation point of view, using a sextic twisted curve can be an efficient technique to generate a random $Q \in \mathbb{F}_{p^{12}}$ for pairing computation.

2.4 Summary

In this chapter, we have described all the mathematic theory required in this project. We started at elliptic curves over finite field, the definition of the Tate pairing and Miller's algorithm. Then, we described Stange's elliptic net theory, especially its application with the Tate pairing. Finally we introduced two popular pairing friendly curves, named supersingular curves and BN-Curves, which will be used in this thesis. We represented how these tow types of curves be generated and how Tate pairing be yield for these curves. We pointed out both the benefits and the weakness of supersingular curves and we assumed that the complexity of the BN-Curves is supposed to enhance the security of the system but the efficiency would be worse than the one of supersingular curves.

Chapter 3

Design

This chapter discusses the initial design of the new IBS system with Elliptic Nets algorithm and the design of the BN-Curve. Then it describes three software development strategies, which is followed by a suitable choice for this thesis.

3.1 Current State of IBS

As we mentioned in Section 1.5, the IBS part of the product provides user validation checking, which is achieved through digital signature schemes with Tate pairing. Figure 3.1 expresses how Tate pairing is used in this system.

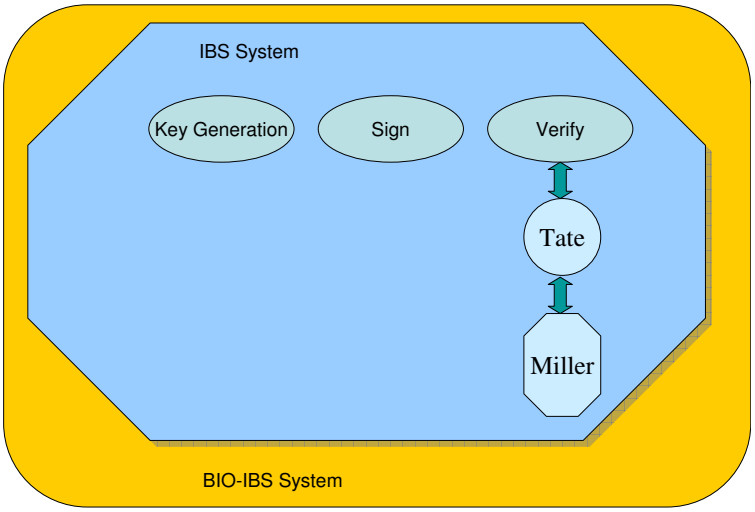

Figure 3.1: Tate Pairing inside IBS

As shown in Figure 3.1, Tate pairing is only employed for verification but not for either key generation or signing in this IBS protocol. For verifying, there exists public system parameters: $(\mathbb{G}_1, \mathbb{G}_2, \hat{e}, P, P_{pub}, H_1, H_2)$ where $H_1, H_2 : \{0,1\}^* \rightarrow \mathbb{G}_1^*$ and $\hat{e}$ is the Tate pairing. The signing service can provide signatures with the form $\sigma = \langle U, V, PAR \rangle$ and to verify such a signature σ on message M, the protocol will take three steps:

1. Get a biometric reading and input the variable PAR to produce the ID', where ID' is the identification the user wishes to identify;
2. Calculate $Q'_{ID} = H_1(ID') \in \mathbb{G}_1$ and $H = H_2(ID', M, U) \in \mathbb{G}_1$;
3. The signature σ is verified if $\hat{e}(P, V) = \hat{e}(P_{pub}, Q'_{ID})\hat{e}(U, H)$ and rejected otherwise.

In Step 3 where this verifier requires three Tate pairing computations. Only Miller's algorithm is currently provided for Tate pairing computation.

3.2 Designing a New IBS

As there are two Tate pairing approaches available now, we desired that the new Tate pairing computation to be implemented for the IBS as shown in the following Figure 3.2.

In this design, we embedded the Elliptic Nets algorithm into the Tate pairing

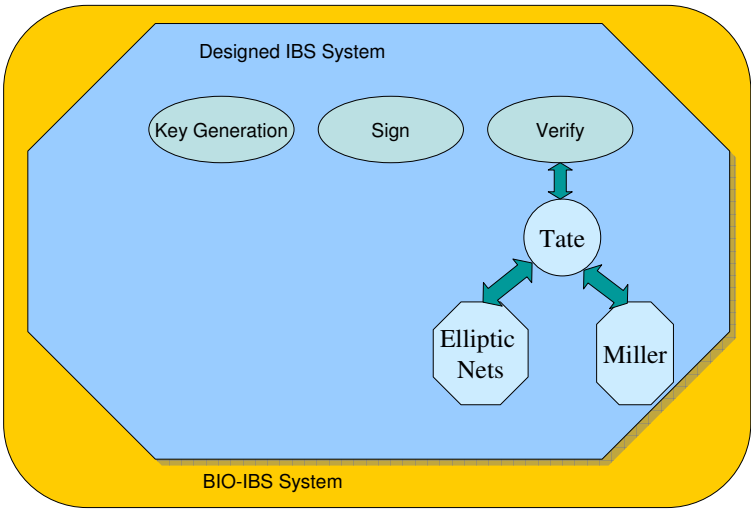

Figure 3.2: Desired New Tate Pairing inside IBS

procedure. This embedding should not conflict with the configuration for the Miller's algorithm and the Tate. This embedding should obey the system settings from its previous parts as well. We should make a decision on which algorithm is to be the default one as such a decision depends on the timing cost of the two algorithms. Moreover, when the verifier detects an inequality in Step 3 with the default algorithm, it should automatically switch to the second one for an alternative equality check rather than rejecting directly. Only when both Tate pairing algorithms result in the inequality status, the

signature will be rejected as an invalid user. If the two approaches yield different results, this means the IBS part has failed and needs to be re-built and meanwhile a notification should be sent to the system to stop the verification service. When a successful build is reached, the IBS should be re-deployed and the system should resume the verification service.

3.3 Design for new curves

As the supersingular curves are considered weak (see Section 2.3.1), we will add Barreto-Naehrig Curves (BN-Curves) for pairing friendly curves. As the BN-Curves have more complex math properties, we should implement an adequate extension field to hold both BN-Curves and their corresponding sextic twisted curves, including points on the curves and arithmetic over the extension field. We will also implement an optimized version of Miller's algorithm to be associated with the Tate computation over BN-Curves.

3.4 Logical view of the Design

In [12], Burnett, Byrne, Dowling and Duffy provide some of the basic finite field and elliptic curve arithmetic mentioned in Section 2.1. It is wrapped into the package called Blitz. Our project will be developed using this Blitz package as a starting point.

3.4.1 Logical View of existing APIs

A brief package view of the Blitz package is shown in Figure 3.3.

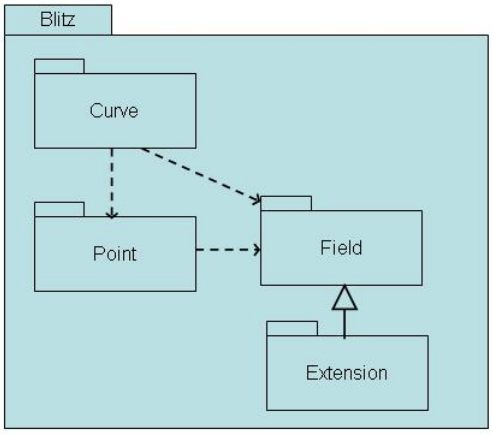

Figure 3.3: Package View of Existing Blitz in [12]

This figures shows four significant subpackages: Curve, Point, Field and Extension, which are all relevant and will be employed in this project. These subpackages cover the implementation of elliptic curves with points on the curves and associated elliptic group laws over finite fields. Note that Blitz also has other subpackages which we do not show because they are not necessary to this project. The logical view of the existing libraries are shown in Figure 3.4.

The dependencies of the APIs are represented from the bottom to top. The Finite Field API is the most basic one because we only need elliptic curves over integer groups. This API covers arithmetic over $\mathbb{F}_p$ and $\mathbb{F}_p^2$. Based on that the Elliptic Curve API is defined within the field. Then the top layer is the Bilinear Map API, which relies on the Finite Field API and the Elliptic Curve API. This API provides Weil pairing and Tate pairing computations through Miller's algorithm. However, we are only interested in Tate pairing here. The curve used in this API is the supersingular curve only.

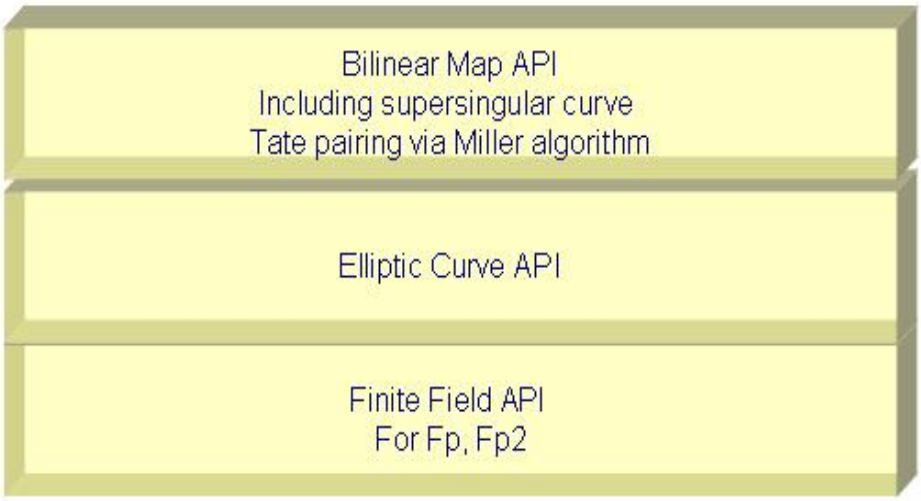

Figure 3.4: Logical View of the Existing APIs

3.4.2 Logical View of New APIs

Our new Elliptic Nets algorithm and the BN-Curve are intended to be implemented based on the existing APIs. Figure 3.5 gives a logical view of the new system. It is called a logical view because the new APIs may be physically packaged independently.

We designed four API's for our project. From the bottom to the top, firstly the Fp12 API is on the bottom. It is within the Finite Field API because it is supposed to be an inherited API as it represents the particular extension field $\mathbb{F}_{p^{12}}$. We keep the Elliptic Curve API as before and then on the third layer, we put the BN-Curves API and the Elliptic Net API. A BN-Curve is a special type of elliptic curve and it can use the Elliptic Curve API for common arithmetic operations. In Section 2.2.2, it was shown that an Elliptic Net can be defined and initialized from an elliptic curve, so there is some dependency on the Elliptic Net and thus its API is designed to rely on the Elliptic Curve API. Note that the supersingular curve API is put on

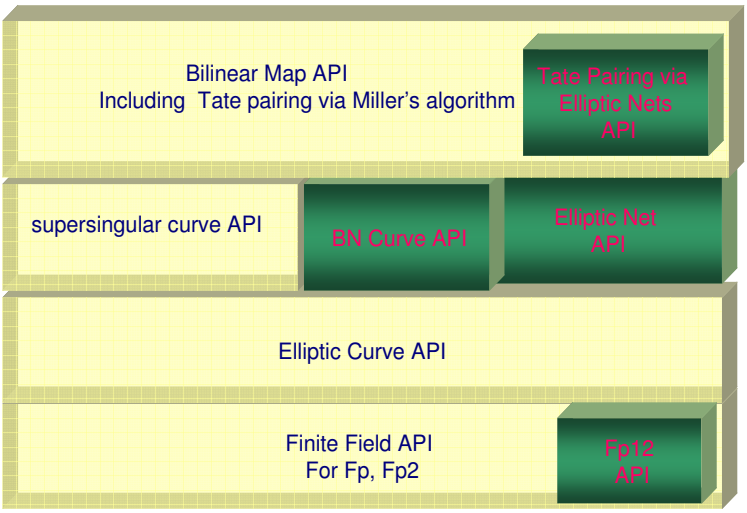

Figure 3.5: Logical View of New APIs

the third layer as well because it is logically parallel to BN-Curves, which means they are all pairing friendly curves without any dependency between them. Finally the top layer still represents the Bilinear map of Tate pairing. Again it depends on pairing friendly curves from the third layer. We add Tate Pairing Via the Elliptic Net API within the Bilinear map API as the new API also provides bilinear Tate pairing computation. It relies on the Elliptic Net API directly. In the actual development, these two APIs may be packaged together.

3.5 Software Development Strategy

In this section we will discuss some common software development models and determine the most suitable one for our project. There are three frequently used development models([56]): Waterfall model, V-model and Iterative and Incremental model. These models are well-structured software development life cycles for planning and organizing software products.

3.5.1 Waterfall Model

The waterfall model is visualized as a linear sequence of phases which begins with software requirements analysis then followed by system design, implementation, testing or verification, and maintenance [56]. Such a sequence is displayed in Figure 3.6. This model allows for departmentalization and

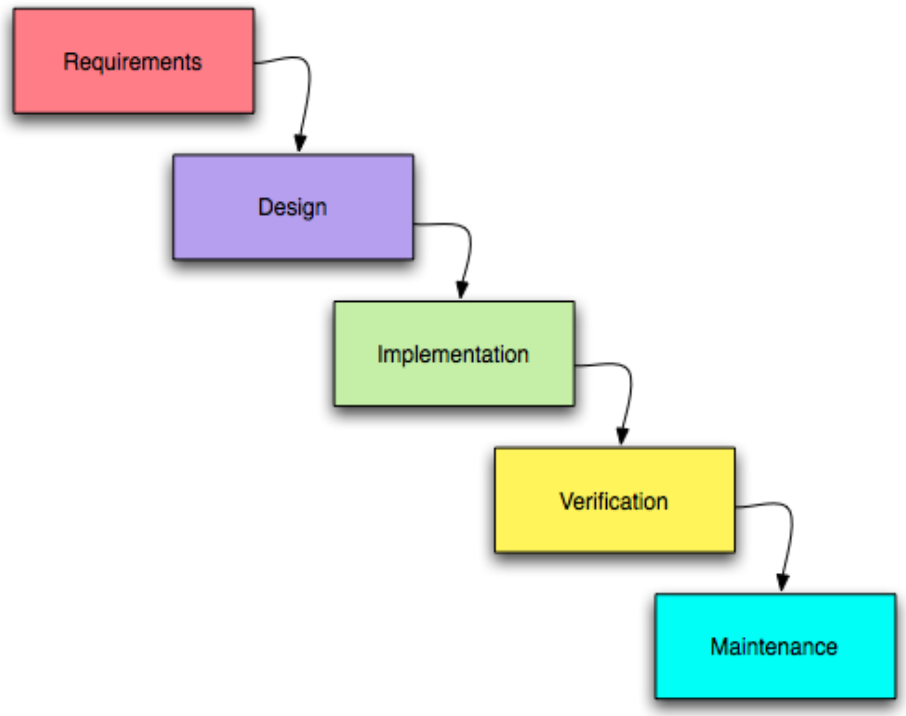

Figure 3.6: Waterfall Model of Software Development in [56]

managerial control. A schedule can be set with deadlines for each stage of development and a product can proceed through the development process and theoretically, be delivered on time. However, the disadvantage of the waterfall model is that it does not allow for much reflection or revision. All the planning should be completed at very beginning and there is no overlap between any phases. The system requirements and specifications should be absolutely correct otherwise the subsequent phases will overrun due to inadequate analysis([56]).

3.5.2 V-Model

The V-Model ([56]) can be presumed as a variation or extension of the waterfall model. Instead of moving down in a linear way as waterfall model, the process steps are bent upwards after the coding phase, to form the typical V shape. Figure 3.7 demonstrates the relationships between each phase of the development life cycle and its associated phase of testing.

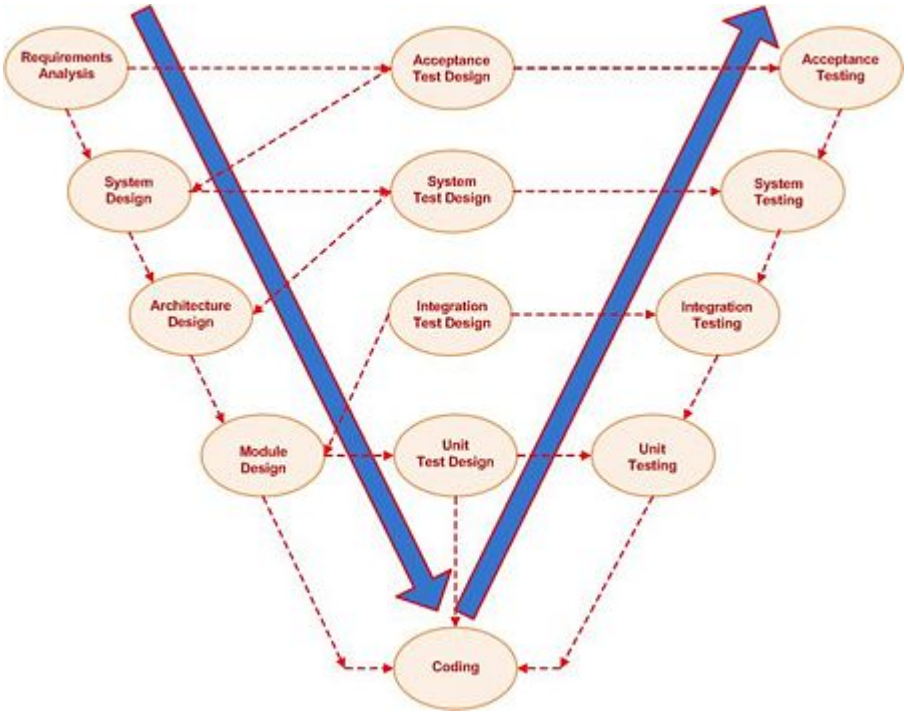

Figure 3.7: V-Model of Software Development in [56]

The advantage of V-Model is that it emphasizes verifications and validations for each phase. It is more test centered than the waterfall model. On the other hand a disadvantage is that this model needs lot of resources. It is thus costly and only suitable for implementation on big projects.

3.5.3 Iterative and Incremental Development Model

The iterative and incremental development model is at the heart of a cyclic software development process developed in response to the weaknesses of the waterfall model. It starts with an initial planning and ends with deployment with the cyclic interactions in between. Figure 3.8 demonstrates the iterative cycle of the model.

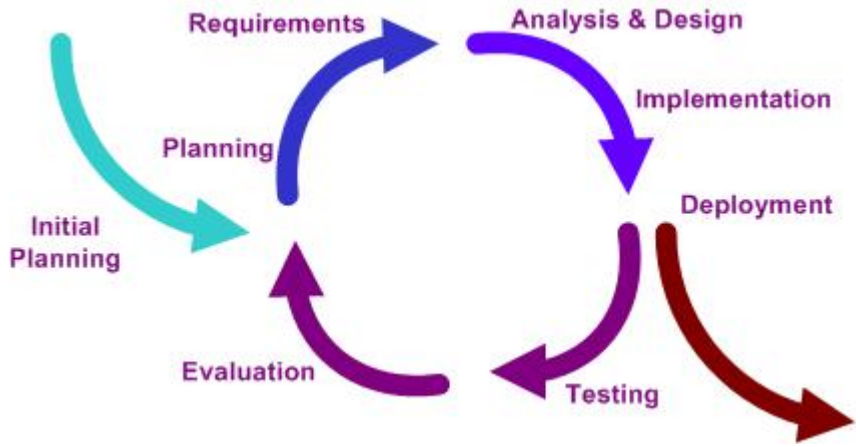

Figure 3.8: Iterative and Incremental Model of Software Development in [24]

Cycles are divided up into smaller, more easily managed iterations. Each iteration passes through the requirements, design, implementation and testing phases. With this model, a project can begin with a simple implementation of a part of the software system and then with each iteration the product evolves with enhancements being added every time until the final version is reached. The major advantages of the incremental model is:

- generating working software quickly and early during the software life cycle;
- more flexible due to being less costly to change the scope or the requirements;
- easier to test and debug during a smaller iteration;
- easier to manage risk because risky pieces are identified and handled during its iteration;
- each iteration is an easily managed milestone.

3.5.4 Our approach of Development process

Comparing and considering all the three general development models, we choose the iterative and incremental model for our project. First of all, this is an individual research orientated project and because the requirements are not fully fixed at the outset, the waterfall model and the V-Model are not suitable. Secondly, this model allows easy testing at each smaller iteration such that errors can be detected and fixed quickly at each stage. Thirdly, it is flexible to update the requirements at each iteration with a less costly implementation and it associated testing. Our project was partitioned into five iterations as shown in Figure 3.9.

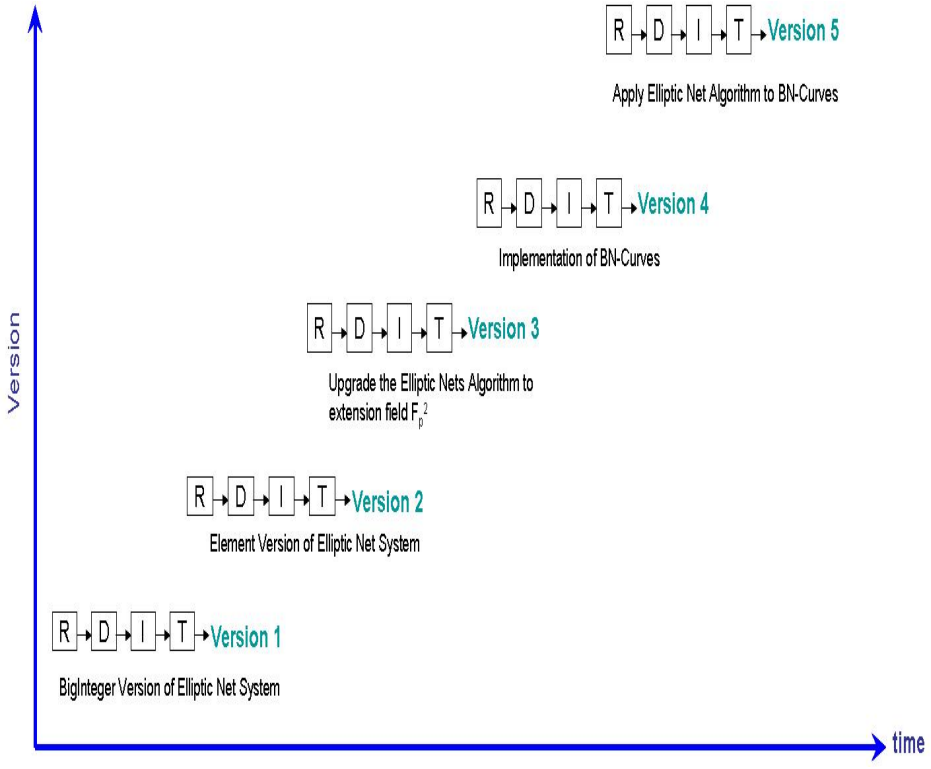

Figure 3.9: Iterations of the Project

In each iteration, it consists of a linear '**R**equirement - **D**esign - **I**mplementation

- **T**esting' sequence and completed with a corresponding version or milestone. All the subsequent iterations are dependent on the success of their previous ones. In brief,

1. **Version 1** will provide a temporary Elliptic Net system with data type of java.math.BigInteger. We start the project at this point to achieve a mathematically correct Elliptic Net system.

2. **Version 2** takes the temporary result and upgrades to the data type of 'element', which is a rich type defined by [12]. This version should allow the Elliptic Net system to operate over the field $\mathbb{F}_p$ and and keep consistency with [12].

3. **Version 3** is another enhancement of the Elliptic Net system as it makes the system work over the extension field $\mathbb{F}_{p^2}$. This is also an important milestone as the comparison of Elliptic Net algorithm with Miller's algorithm over supersingular curves is firstly obtained at this stage and the result of the comparison will dominate the selection of the default algorithm for Tate pairing at a later time.

4. **Version 4** is the implementation of BN-Curves as we described in Section 2.3.2. This version introduces new pairing friendly curves to the system which extends the software to deal with the mathematical concepts of the extension field $\mathbb{F}_{p^{12}}$, BN-Curves, and twisted curves with corresponding functionalities. It provides a more complex mathematical platform for the Elliptic Net system. Note that this iteration is independent from the previous versions and can be started in parallel with the testing phase of version 3.

5. **Version 5** applies the Elliptic Net system to the BN-curves. It is strongly dependent on all the previous versions. As a result the sys-

tem can accept BN-Curves as well as the supersingular curves for Tate pairing. This version will also allow a comparison of the two algorithms.

3.6 Summary

In this chapter, we reviewed the existing IBS product and its related API's. We provided our ideal design of the new system with the Elliptic Net Algorithm. We discussed and chose a suitable software development process to organize the project development. We provided a detailed development plan as the project was partitioned into five smaller sub-projects and each of them will achieve a significant milestone in the project

Chapter 4

Development

In this chapter, we discuss the complete software application that was developed. We firstly analyzed the two existing Java resources [12], [19] in our lab environment, then designed, implemented and integrated the Elliptic Nets system for Tate pairing computation. We then implemented new fields for BN-Curves, implemented BN-Curves and applied the Elliptic Nets algorithm to BN-Curves for Tate computation. We will provide details on the implementations including requirements, issues and solutions.

4.1 Analysis of existing Elliptic Curve and Pairing libraries

We used Java as the programming language as Java supports multi-platforms and once a program is compiled it can be executed by any Java Virtual Machine [45]. There is no need to re-configure the environment settings in contrast to C/C++ applications. In Section 3.4, the package Blitz in [12] provides operations over the finite field $\mathbb{F}_p$ and limited calculations over $\mathbb{F}_{p^2}$. It provides the functionalities of the elliptic curve group laws in different

coordinates with their corresponding point representation as well. The later development by Owens, Duffy and Dowling's IBE API [19] was dependent on [12]. Both APIs use the java.math.BigInteger class to represent and store large numbers. The benefit is because since real large numbers (i.e. a 256-bit number) can be defined easily and basic operations on these numbers are also provided by JAVA itself. The speed of the system is somewhat dependent on the running environment of the JVM. However, taking a closer look at the whole system: although it provides a Java oriented approach for an Identity Based Encryption system [7], the pairing computation part of the system is very limited on the supersingular curve (see Section 2.3.1). Furthermore, the algorithm for Tate pairing uses the standard Miller's algorithm. These characteristics were implemented to improve the efficiency of the whole system at that time but now have become drawbacks to the system, particularly since the weakness of supersingular curves were revealed in [36]. Additionally, now an optimized version of Miller's algorithm has been included. As a result, the libraries need to be evolved to include the extension field, to accept more types of curves, and to support the Elliptic Net algorithm.

4.2 Development of the Elliptic Net System

In this section, we discuss the implementation issues involved in the Elliptic Net system. It includes the Version 1 and Version 2 we mentioned in Section 3.5.4. This Elliptic Net system should meet the following requirements:

1. It implements the mathematical model EDS and Elliptic Net;
2. It can handle large numbers;
3. It is compatible with the existing Blitz package;

4. Tate pairing can be computed through the Elliptic Nets algorithm.

The idea is to provide a new Elliptic Net system based on the existing Blitz package and make the Blitz package more useful. We keep the java.math.BigInteger class to hold the large numbers such that we can reuse the Blitz package. By design, the package view of the Elliptic Net system is shown in Figure 4.1.

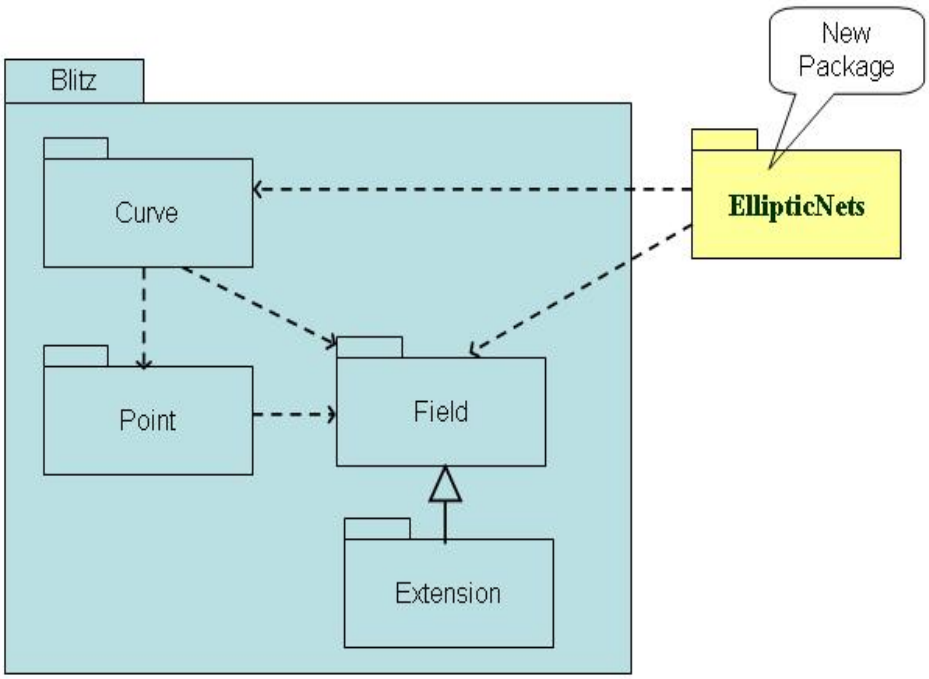

Figure 4.1: Package Diagram of Elliptic Nets System

We create a new package csi.crypto.EllipticNets to contain all classes of the Elliptic Net system. It depends on the Blitz.curve package and the Blitz.Field package. The former is used to define the elliptic curves and their related operations and the latter, with its subpackage, defines the arithmetic of the finite field associated with them. The csi.crypto.EllipticNets package will wrap all the essential components including:

- A component to represent an elliptic divisibility sequence;
- A component to represent an Elliptic Net;
- A component to represent the data structure of an Elliptic Net;
- A component to perform Tate pairing calculation through the Elliptic Nets algorithm;

The whole classes view of the package is depicted in Figure 4.2. The Block class is considered as a composite of the EllipticNet class. The TatePairingViaENet class may have at most one either EDS class or EllipticNet class depending on the content. Sections 4.2.1, 4.2.2, 4.2.3 and 4.2.4 give more information about each class in detail.

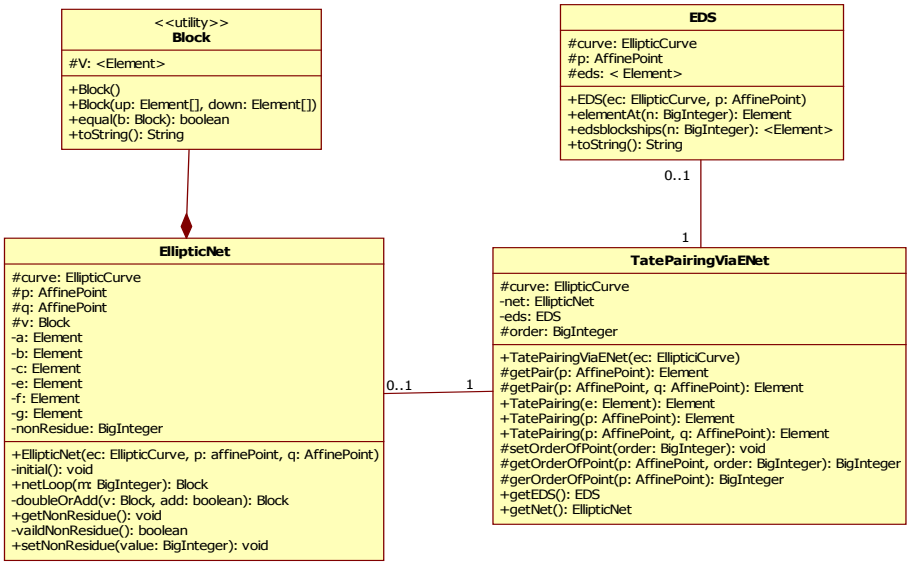

Figure 4.2: Class Diagram of EllipticNets package

4.2.1 Class EDS

This class represents an elliptic divisibility sequence object. It implements Shipsey's double-and-add algorithm to calculate terms in the elliptic divisibility sequence. It takes an elliptic curve and a point on the curve as parameters in its constructor to generate the first five elements in the corresponding elliptic divisibility sequence. An 8-element array is used for computation and storage of elliptic divisibility sequence, where the fourth element is called the center of this representation. It has the *elementAt()* function to take specified BigInteger n as parameter and calculate the n^{th} term (i.e. $W(n)$) in the sequence. The *edsblockships()* function will take BigInteger n and return the 8-element array which is centered at $W(n)$.

4.2.2 Class Block

This class represents the data structure defined in Figure 2.2. It uses a 2D array to hold the elements in a block. It is restricted to have exactly three elements in the upper row and eight in the lower row. It also has the *equal()* function to compare whether two blocks contain exactly the same values in the same order and then returns a corresponding boolean value.

4.2.3 Class EllipticNet

This class represents the Elliptic Net object. It takes an elliptic curve and two different points on the curve as parameters in its constructor to create an EllipticNet object. The private *initial()* function will be called from the constructor to initialize all necessary settings for the Elliptic Net, which includes all the private attributes of the class and the initial block (i.e. the block centered at $W(1,0)$) in the net. The *netLoop()* function will take the input BigInteger m, compute and then return a block centered at $W(m,0)$ in

the net. The *doubleOrAdd()* function takes the first block argument V and performs doubling or double-adding operations on the block V depending on the second boolean parameter. This function is set to be private as it could only be internally called by the *netLoop()* function in this class. In order to speed up the recursive part of the Elliptic Net algorithm (i.e. the performance of doubling and double-adding operations), we use the following Algorithm 3 as detailed in [58].

Algorithm 3 Doubling and Double-adding Algorithm

Input: Block V centered at k of an Elliptic Net. $A = W(2,0)^{-1}$, $E = W(-1,1)^{-1}$, $F = W(2,-1)^{-1}$, and boolean add

Output: Block centered at $2k$ if $add == 0$ and centered at $2k+1$ if $add == 1$

$S \leftarrow [0,0,0,0,0,0]$
$P \leftarrow [0,0,0,0,0,0]$
$S_0 \leftarrow V[0,1]^2$
$P_0 \leftarrow V[0,0]V[0,2]$
for $i = 0$ to 5 **do**
  $S_i \leftarrow V[1,i+1]^2$
  $P_i \leftarrow V[1,i]V[1,i+2]$
end for
if $add == 0$ **then**
  for $i = 0$ to 4 **do**
    $V[1,2i-2] = S[i-1]P[i] - S[i]P[i-1]$
    $V[1,2i-1] = A(S[i-1]P[i+1] - S[i+1]P[i-1])$
  end for
  $V[0,0] = S[1]P_0 - S_0P[1]$
  $V[0,1] = S[2]P_0 - S_0P[2]$
  $V[0,2] = E(S[3]P_0 - S_0P[3])$
else
  for $i = 0$ to 4 **do**
    $V[1,2i-2] = A(S[i-1]P[i+1] - S[i+1]P[i-1])$
    $V[1,2i-1] = S[i]P[i+1] - S[i+1]P[i]$
  end for
  $V[0,0] = S[2]P_0 - S_0P[2]$
  $V[0,1] = E(S[3]P_0 - S_0P[3])$
  $V[0,2] = F(S_0P[4] - S[4]P_0)$
end if
return V

4.2.4 Class TatePairingViaENet

This is the most important class in the package as the other classes are defined to support this class. It performs the Tate pairing computation from the Elliptic Nets algorithm. It requires an elliptic curve object in its constructor. The functions *getPair()* and *getTatePairing()* are overloaded with different numbers or types of parameters. The *getPair()* functions perform the Elliptic Net algorithm and returns a corset value τ' (in Section 2.1.1). If there is only one AffinePoint parameter passed to it, an EDS object will be created to calculate the Tate corset according to Equation 2.8. If there are two different AffinePoints passed to it, it will call for an Elliptic Net object and perform the Tate computation with Equation 2.9. Similarly the *getTatePairing()* can accept one or two AffinePoints as input and call the corresponding *getPair()* for a coset value and then perform Equation 2.10. The *getTatePairing()* can directly take a corset value and apply Equation 2.10 as well. Actually the *getPairing()* functions can be merged to *getTatePairing()* functions, but to facilitate testing we separated them. The functions *getEDS()* and *getNet()* are defined to return a corresponding EDS object or EllipticNet object that are used during the Tate calculation. The working procedure of this class is shown in Figure 4.3. The various options for invoking the Tate computation are illustrated in the figure with three possible outcomes depending on the decision that produce the Tate pairing.

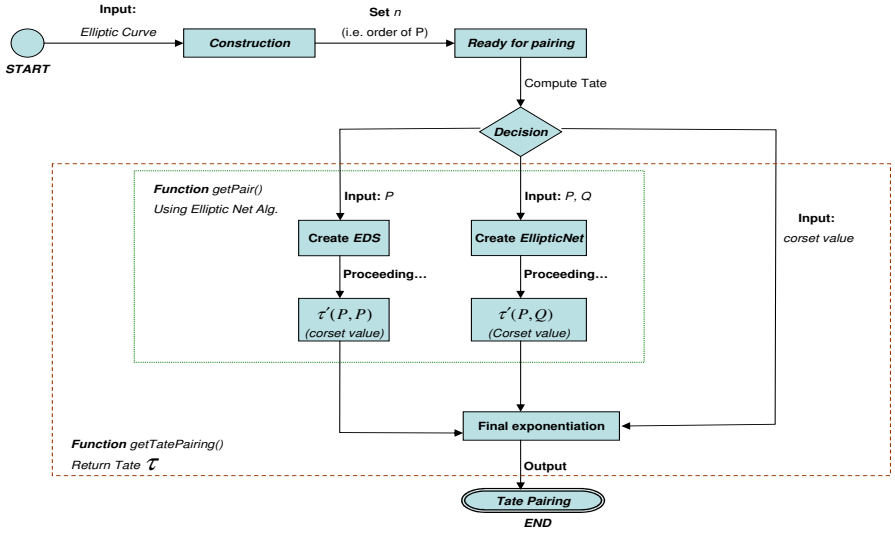

Figure 4.3: Flow of Tate Pairing Computation

By the definition of Tate calculation, the order of AffinePoint P is significant as it directly determines how many iterations are needed to give the Tate computation. Before applying the Equations 2.8 or 2.9, it is necessary to figure out the order of the point. Thus, the attribute order is introduced to contain the correct order of a point on the curve. The function *setOrderOfPoint()* can be called if the order of the point is known. However, if the order of point is unknown, the overloaded *getOrderOfPoint()* functions will be more useful. They can check whether the current order or a user-inputted order is correct for some particular point, and if not, calculate the exact order of the point using techniques in [62]. The order of P can be computed as a precondition offline at a time before the Elliptic Net algorithm is invoked. In fact, it can be considered as a known system parameter. In [19], it is fixed

during the generation of the system parameters. Even for the BN-Curves, it is fixed during the constructions of the curve.

4.3 Evolving the IBS system

This section describes the implementation of Version 3. As we mentioned the EllipticNet package relies on the Blitz, so there are some impacts on Blitz. In order to make the two systems compatible with each other, some classes in Blitz needed adjustment. Additionally, as the existing IBS system [19] also depends on the Blitz, any adjustment should not conflict with the current IBS. There are two main issues here:

1. The Elliptic Net system takes a curve in Weierstrass form, how can a general form of Elliptic curve be accepted by the system?

2. For a cryptographic application, the system should work over $\mathbb{F}_{p^2}$. How to achieve this requirement?

The following parts of this section will discuss and fix these issues.

4.3.1 Class blitz.curve.EllipticCurve

The Blitz.curve.EllipticCurve class defines the elliptic curve object. Originally, the class defines a curve in a general form. In the Elliptic Net system, all the elliptic curves we used are in Weierstrass form (see Equation 2.2). Thus, we need to add a new function, *toWeierstrassEqn()*, to the EllipticCurve class for transforming a curve from a general form (see Equation 2.1) to its Weierstrass form. Also, new attributes A and B are needed and some corresponding 'get' functions are essential to return their values. The *toWeierstrassEqn()* is even called as the very last step of all the constructors to guarantee that A and B always exist and are available after an elliptic

curve is defined. A function *toWEView()* that returns a string that represents the Weierstrass curve was added. A constructor that directly creates a curve in Weierstrass form is added as well. In that case, $A = a_4$, $B = a_6$, and a_1, a_2, $a_3 = 0$.

4.3.2 Class blitz.Field.Extension.Fp2

Point P must be an n-torsion point on an elliptic curve over a finite field $\mathbb{F}_p$. Point Q can be any point on the curve over $\mathbb{F}_p$. However, in cryptographic applications of the Tate pairing, Q is usually in $\mathbb{F}_{p^k}$. In our Blitz package there is a particular class, Blitz.Field.Extension.Fp2, that defines the arithmetic of $\mathbb{F}_{p^2}$. As mentioned before, all the functions were defined for the supersingular curves described in Section 2.3.1 in such a way that the software cost at run time could be minimized. However, this limits the reuse of the class.

To make this class more general, a new attribute, *nonResidue*, was added to this class to store the specified quadratic non-residue value. It is by default equal to -1, but can be changed when necessary. All related functions including the constructors in the class are redefined. The new Blitz.Field.Extension.Fp2 class can now:

- specify a customized nonResidue when creating a new Fp2 object with the following:

```
public Fp2 (BigInteger characteristic,
                              BigInteger nonResidue){
      super( characteristic, new BigInteger( "2" ));
      this.nonResidue = nonResidue;
}
```

- change existing Fp2 objects' *nonResidue*:

```
public void setNonResidue(BigInteger value){
     nonResidue = value;
}
```

- assign *nonResidue* to a specified Fp2Element by calling *setNonResidue(BigInteger value)* first and then calling the existing *element(Element e)* to accept the same settings.

- perform corrected multiplication, division, and power-mod operations. Take multiplication for example, the following code will take firstly convert the two non-null input arguments, a and b, into Fp2Element type, known as s and t, then perform multiplication function with t and s, and return the result. Note that the *nonResidue* attribute is applied within the calculation.

```
private Element mult( Element a, Element b ) {
   if( a == null ) {
          throw new NullPointerException(
           "a cannot be null" );
      }
      if( b == null ) {
          throw new NullPointerException(
                      "b cannot be null" );
      }
      Fp2Element s = null;
      Fp2Element t = null;
      if( a instanceof Fp2Element ) {
```

```
            s = (Fp2Element) a;
        }
        else {
            s = new Fp2Element( a );
        }
        if( b instanceof Fp2Element ) {
            t = (Fp2Element) b;
        }
        else {
            t = new Fp2Element( b );
        }
        // nonResidue is applied below...
        BigInteger r = s.real().multiply( t.real()).
            add(s.imag().multiply( t.imag().multiply
           (nonResidue)).mod(characteristic);
        BigInteger i =  s.real().multiply( t.imag()).
            add(s.imag().multiply( t.real() ).
            mod(characteristic);
        return( new Fp2Element( r, i, this ) );
    }
```

All these adjustments allow point Q to exist over any $\mathbb{F}_p(i)$ where i^2 equals to any quadratic nonresidue of the curve.

4.3.3 *nonResidue* in class csi.crypto.EllipticNets.EllipticNet

For the consistency of the system, the *nonResidue* attribute is also added to the EllipticNet class so that when the *initial()* function is called by the constructor, a corrected *nonResidue* for any Fp2 environment in the net can

be obtained if it detects that $Q \in \mathbb{F}_{p^2}$. This can be done by repeatedly selecting a quadratic nonresidue and detecting whether it is valid until a correct one obtained. It also allows the *nonResidue* to be specified by the user if it is known.

4.4 Adding BN-Curves

The previous Sections 4.2 and 4.3 implemented the Elliptic Nets algorithm and applied this algorithm to existing supersingular curves. This section will switch to the new pairing friendly curves, BN-curves. It will cover and complete Version 4 and 5 of the application.

Again, the Elliptic Nets system can accept general curves with embedding degree $k = 2$ and perform this exact pairing computation. A new question arose as whether this algorithm works for higher embedded degree curves. To obtain the answer the next stage adds BN-Curves to the system to enhance the security of Tate pairing as the BN-Curves are more secure than the supersingular curves. We created a new package *csi.crypto.pairing* to obtain all the components of the BN-Curves and pairing over BN-Curves. There are two steps to achieve this implementation:

1. Field extension
2. Curve Generation

4.4.1 Field extension

The BN-Curves have embedding degree $k = 12$ and so we need objects to define the extension field $\mathbb{F}_{p^{12}}$ with corresponding arithmetic and to represent elements in that field. In [12], there exists the classes blitz.field.Element, blitz.field.Fp and blitz.field.extension.Fp2Element. We need two new classes

named Fp12 and Fp12Element to fulfill the requirements. The new classes are described in Figure 4.4.

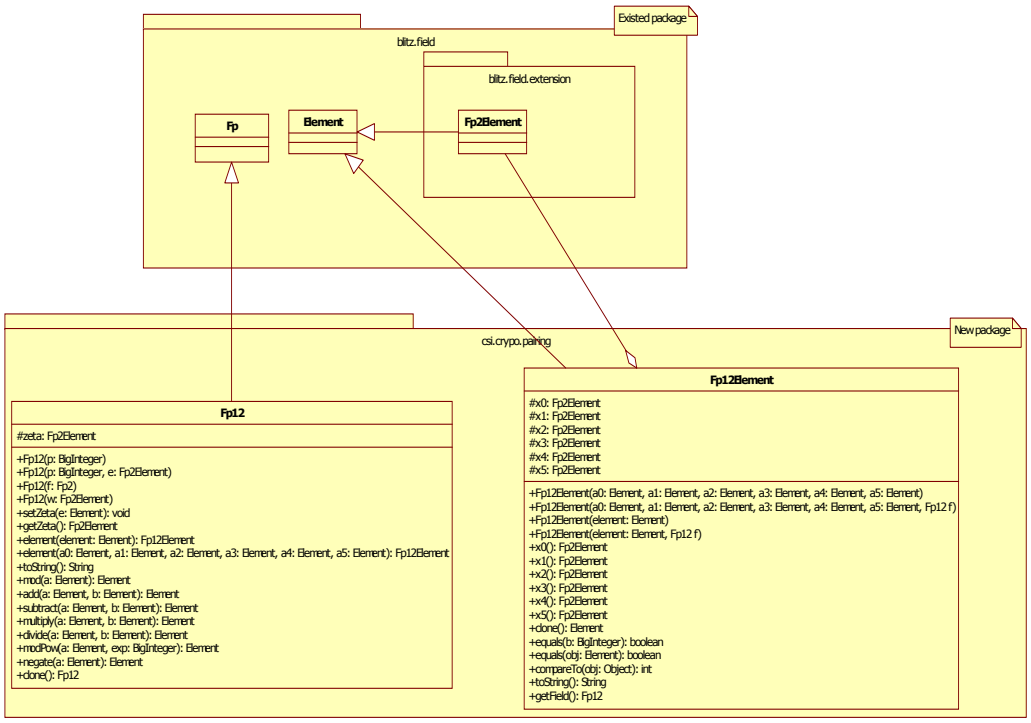

Figure 4.4: Class Diagram of Fp12 and Fp12Element

The arrows in the figure show the inheritance relationships between old classes and the new components. The class Fp12 represents $\mathbb{F}_{p^{12}}$ and it is inherited from class Fp such that it obeys the properties and rules defined by Fp. All functions besides constructors are overridden such that the system can automatically determine which version of functions to be actually invoked at run time. Considering the JAVA polymorphism technique, the Fp12Element class is also inherited from the Element class. Moreover,

- Fp12 has an extra attribute *zeta* of type Fp2Element. This *zeta* represents ζ mentioned in Section 2.3.2. The related *setZeta()* and *getZeta()* are provided for customized setting and reviewing.

- Fp12Element refers to element in $\mathbb{F}_{p^{12}}$. It is composited of six Fp2Element objects to define an element in $\mathbb{F}_{p^{12}}$ which satisfy the representation in Equation 2.13 as well.

- Both classes contain various constructions such that users can create instances with a preferred one to facilitate the testing.

4.4.2 Curve Generation

Since we have provided a suitable $\mathbb{F}_{p^{12}}$ environment, the next step is to obtain BN-curves. It includes three parts: generating curve parameters, representing BN-Curves and getting the corresponding twisted curves. To achieve this we add three new classes in the package pairing as shown in Figure 4.5.

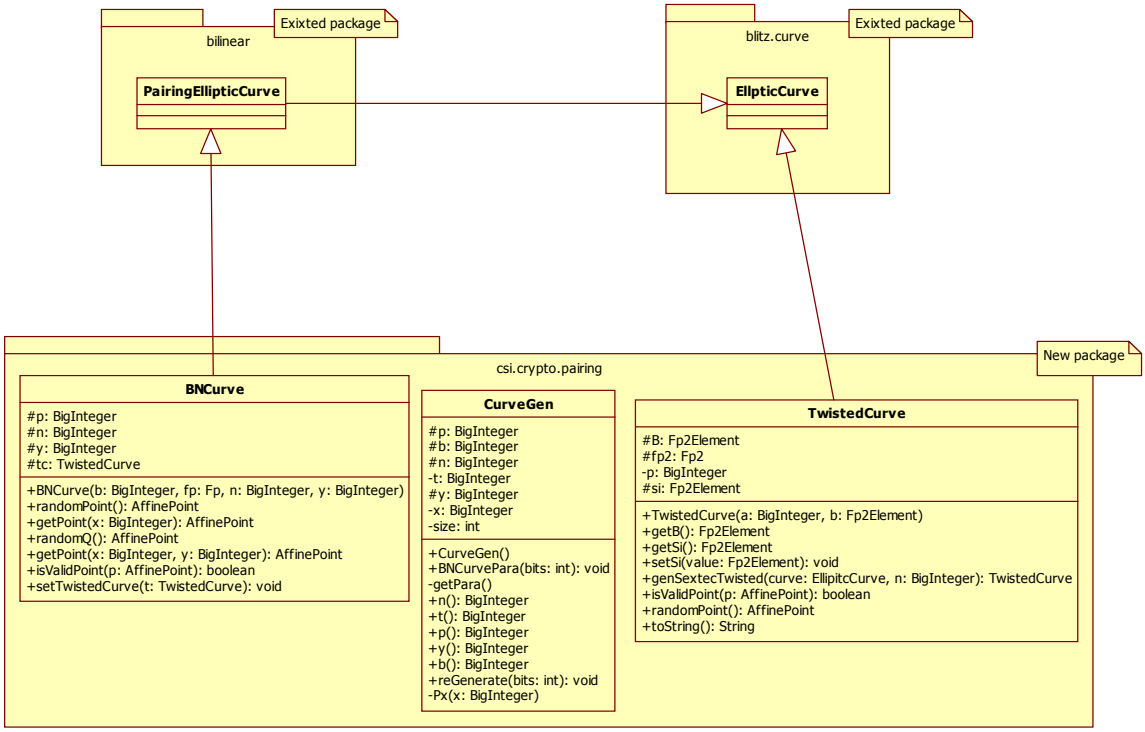

Figure 4.5: Class Diagram of CurveGen, BNCurve and TwistedCurve

These new classes are CurveGen, BNCurve and TwistedCurve. Basically, CurveGen class will generate suitable parameters for constructing BN-Curves; then the BNCurve class takes these parameters to create and represent BN-Curves; TwistedCurve class will take a BN-Curve and generate a corresponding sextic twisted curve to facilitate the generation of point Q which is the second argument for the Tate pairing.

class csi.crypto.pairing.CurveGen

This class can take a specified length as input and perform Algorithm 2 in Section 2.3.2 to generate suitable parameters for this specified-length BN-Curves (e.g. a 160-bits BNCurve). Algorithm 2 guarantees that the output $p = 1 \pmod 6$. However in order to simplify the calculations of the square root and cubic root for further use, we put two more conditions on p such that $p = 3 \pmod 4$ and $p = 4 \pmod 9$. The simplifications are based on the following properties: ([3])

- Let $p = 3 \pmod 4$ and x is a square modulo p, then there exists a square root $r = x^{\frac{p+1}{4}} \pmod p$ such that $r^2 = x \pmod p$.
- Let $p = 4 \pmod 9$ and $x \in \mathbb{F}_{p^2}^*$ is a cube, then one cube root $r \in \mathbb{F}_{p^2}^*$ can be obtained by $r = x^{\frac{p^2+2}{9}}$.

Another issue is about the *smallest* value of x in step 2 of Algorithm 2. This is the initial value of x and it is proposed to be as small as possible such that the final length of p will not exceed the expected length when the loop completes as in step 16 of Algorithm 2. In practice, it is hard and time-consuming to detect whether x is the smallest or not when the value of input *bits* is large (e.g. $bits \geq 160$). A suitable initial value or guess for x can also speed up the process of parameter generation. During implementation we picked an approximate value for the initial value of x. It is not the actual smallest one but small enough to complete the whole generation of the curve parameters within a sensible amount of time.

class csi.crypto.pairing.BNCurve

The BNCurve class represents the BN-Curve. It inherits the PairingElliptic-Curve class as it is pairing-friendly. Some methods in this class are overridden

to increase the speed of computation. There are three special methods in the class named *randomPoint()*, *setTwistedCurve()* and *randomQ()*. Firstly the method *randomPoint()* is specified to get a random point on $E(\mathbb{F}_p)$. The strategy is not unique. The one in pairingEllipticCurve class works as following:

1. Choose an x randomly;
2. Check whether there exists y for the x;
3. Calculate y and return point (x, y) if y exists by step 2 or update a new x and repeat steps 2 and 3 until a suitable point is found.

This strategy also works for the BN-Curves. However, we have a more efficient approach. The CurveGen class generates the parameter y for each curve and this parameter guarantees the point $g = (1, y)$ is a generator of the curve which means for any positive integer s, $[s]g \in E(\mathbb{F}_p)$. Thus, the following strategy works for random point generation:

1. Choose a random s;
2. Return point $P = [s]g$ if $[s]g \neq \infty$, otherwise choose another random s until a suitable point is found.

In the BNCurve class we use the second strategy because it is fast. The first strategy includes a random value generation for x, calculations for the check in step 2, and a normal (non-simplified) square root computation in step 3. The second one only relies on a random value generation for s and the multiplication of elliptic curves.

The *setTwistedCurve()* method is used to find a suitable sextic twisted curve for the current BN-Curve. This method can be called to customize a known twisted curve by passing a specified twisted curve argument. It is also allowed

to be invoked with a null value argument and in this case it will automatically call a method in TwistedCurve to get a corresponding twisted curve. Hence, once this method is called there always exists a twisted curve which will be used for the generation of random point $Q \in \mathbb{F}_{p^{12}}$ as the second argument for the Tate pairing computation.

The method *randomQ()* is important as it provides the second argument Q for Tate pairing. This is not as simple as the previous random point function in *randomPoint()*. It is hard and not practical to generate $Q \in \mathbb{F}_{p^{12}}$ directly. Instead the twisted curve provides an efficient manner to achieve Q through the following procedure:

1. Find a suitable twisted curve $E'(\mathbb{F}_{p^2})$;
2. Fine a random point $Q'(x', y') \in E'(\mathbb{F}_{p^2})$;
3. Let Q' be n-torsion on $E'(\mathbb{F}_{p^2})$;
4. Map $Q = \Psi(Q')$ via the homomorphism Ψ in the Section 2.3.2;
5. Return Q.

Steps 1 and 2 will be done with the class csi.crypto.pairing.TwistedCurve. They produce a suitable twisted curve E' with proper point $Q' \in E'(\mathbb{F}_{p^2})$. This is the primary reason we use a twisted curve for pairing. Step 3 makes Q' more special and as a result $Q \in \mathbb{F}_{p^{12}}[n]$. Recall from the definition of Tate pairing $\tau_n(P, Q)$ that the second argument Q is not necessary to be an n-torsion point. Considering the pairing compression technique, it is better to have Q be n-torsion over $\mathbb{F}_{p^{12}}$. It is obvious that the generation of a point over $\mathbb{F}_{p^2}$ is much easier than over $\mathbb{F}_{p^{12}}$. This method depends on the functionality of class csi.crypto.pairing.TwistedCurve and it can be applied to the pairing compression technique.

class csi.crypto.pairing.TwistedCurve

This is an essential class of the system as it represents the object of TwistedCurve and provides the services of twisted curve generation and point generation on the twisted curve such that it supports the functions in class csi.crypto.pairing.BNCurve. The core functions of this class are defined in methods *genSexticTwisted()* and *randomPoint()*.

The method *genSexticTwisted()* takes a BNCurve and its order n as input and output a corresponding sextic twisted curve. The hardest part of the procedure is to find a corrected $\zeta \in \mathbb{F}_{p^2}$. As mentioned in Section 2.3.2, a suitable ζ will satisfy the two conditions(in [3]):

1. $X^6 - \zeta$ is irreducible over $\mathbb{F}_{p^2}[X]$.
2. $\#E'(\mathbb{F}_{p^2}) = n(2p - n)$ where $E' : y^2 = x^3 + b/\zeta$.

Considering the mathematical property, one strategy to get a possible ζ is:

1. Find a non-cube $\lambda \in \mathbb{F}_p$.
2. Find a non-square $\mu \in \mathbb{F}_{p^2}$.
3. Calculate $1/\zeta = \lambda^2\mu^3$.
4. calculate the order of $E' : y^2 = x^3 + b/\zeta$.
5. Return E' if the equality $\#E' = n(2p - n)$ holds. Otherwise repeat the whole procedure until a twisted curve is obtained.

In practical implementation, the first two steps are difficult to achieve and a smaller value of ζ may be preferred to reduce the computation cost. Another issue that may cost time is the calculation of the order of E' directly. Hence, the strategy of a linear search is quicker as follows:

1. Set $\zeta = 1 + i$.
2. Define E' with ζ.
3. Randomly choose a non-infinity point $R \in E'$.
4. Return E' if $[n(2p - n)]R = 0$. Otherwise, update ζ with minimal increment and repeat steps 2-4 until a twisted curve is obtained.

Within this strategy, step 3 is the most time consuming part and it can be achieved by invoking the method *randomPoint()* which is the other core function of this class. As the required twisted curve always has the order $\#E' = n(2p - n)$, we skipped the calculation of $\#E'$ and instead we used this particular value of $n(2p-n)$ to detect whether a random $R \in E'$ satisfies the property $[n(2p - n)]R = 0$. If it holds then we get the correct ζ and E'. The *randomPoint()* method is used to generate a random point $R \in E'(\mathbb{F}_{p^2})$. It is significant because it is invoked during both the generation of the twisted curve for order checking and the generation of Q' for the second argument Q in Tate pairing. Assume the twisted curve $E' : y^2 = x^3 + B$ is generated by *genSexticTwisted()* and then the basic idea of finding a point is given by the following steps:

1. Randomly choose a non-zero $x \in \mathbb{F}_{p^2}$;
2. Let $rhs := x^3 + B$;
3. If rhs is a quadratic residue then compute the corresponding square root r. Otherwise go back to step 1.
4. Return the point $R(x, r)$.

The most difficult part of the above procedure is step 3. It involves a quadratic residue test and square root computation over $\mathbb{F}_{p^2}$. Smart's algorithm [6] can be applied and adjusted to achieve this efficiently. The

pseudocode in Algorithm 4 merges the above 4 steps and shows a completed procedure of random point generation in the class TwistedCurve. The work flow of this algorithm is briefly shown in Figure 4.6.

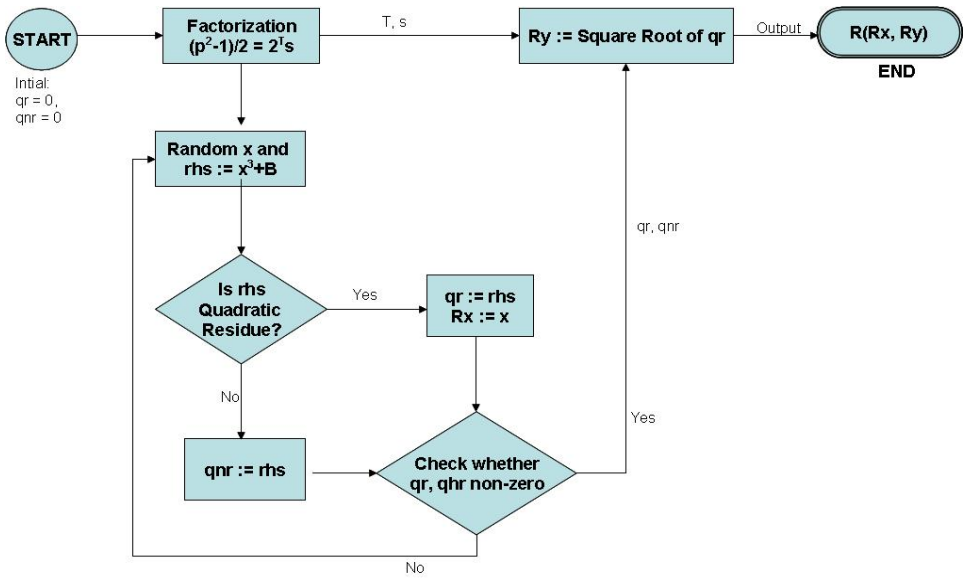

Figure 4.6: Flow of Generation of Random Point R on $E' : y^2 = x^3 + B$

It firstly calculate integers s and T through a factorization such that

$$(p^2 - 1)/2 = 2^T s \tag{4.1}$$

where $T \geq 0$ and s is odd. It then lets the system randomly generate $x \in \mathbb{F}_{p^2}$ and calculate the corresponding $rhs := x^3 + B$ and this rhs is passed to the quadratic residue test. If rhs is quadratic residue, then the x-coordinates of R can be set as $R_x := x$ and we let the variable $qr := rhs(= R_x^3 + B)$ to keep this quadratic residue for square root computation later on. If rhs is quadratic non-residue, we still keep the value in another variable qnr for further use. We repeat the random generation of x until we get value for both qr and qnr. Next we gather all four variables s, T, qr and qnr to

compute the square root of qr which is also the y-coordinates of R (i.e. $R_y = \sqrt{qr} \Leftrightarrow R_y^2 = qr = R_x^3 + B$). Finally we return $R := (R_x, R_y)$.

Algorithm 4 Random Point on $E'(\mathbb{F}_{p^2})$

Assume: Assume twisted curve $E' : y^2 = x^3 + B$
Output: Affine point $R = (R_x, R_y) \in E'(\mathbb{F}_{p^2})$

$getQR \leftarrow$ **false**, $getQNR \leftarrow$ **false**,$T \leftarrow 0$, $t \leftarrow 0$, $b \leftarrow 0$
$x \leftarrow 0$, $check \leftarrow 0$, $rhs \leftarrow 0$, $temp \leftarrow 0$, $qnr \leftarrow 0$, $qr \leftarrow 0$, $tk \leftarrow 0$
$s \leftarrow \frac{p^2-1}{2}$
while s is even **do**
  $s \leftarrow s/2$
  $T \leftarrow T + 1$
end while
while $!getQR \,||\, !getQNR$ **do**
  $x \leftarrow$ Random value over $\mathbb{F}_{p^2}$
  $rhs \leftarrow x^3 + B$
  $t \leftarrow 0$
  $temp \leftarrow rhs^s$
  if $temp = \pm 1$ **then**
    $check \leftarrow 1$
  else
    repeat
      $temp \leftarrow temp^2$
      $t \leftarrow t + 1$
    until $temp = -1$
    if $t < T$ **then**
      $check \leftarrow 1$
    else
      $check \leftarrow 2$
    end if
  end if
  if $check = 1$ && $!getQR$ **then**
    $getQR \leftarrow$ **true**
    $R_x \leftarrow x$
    $qr \leftarrow rhs$
  end if
  if $check = 2$ && $!getQNR$ **then**
    $getQNR \leftarrow$ **true**
    $qnr \leftarrow rhs$
  end if
end while
$b \leftarrow qr^{\frac{s-1}{2}}$
$t_0 \leftarrow 0$
for $k := 0$ to $T - 1$ **do**
  if $((qnr^{t_k} b)^2 \cdot qr)^{2^{T-1-k}} == -1$ **then**
    $t_{k+1} \leftarrow t_k + 2^k$
  end if
end for
$R_y \leftarrow a^{t_k} b \cdot qr$
return $R(R_x, R_y)$

4.4.3 Tate Pairing over BN-Curves

In the previous Sections 4.4.1 and 4.4.2, we implemented suitable fields for BN-Curves and their corresponding twisted curves. We also have supported math components to hold new curves thus completing Version 4. Based on that we can use these resources to build a new protocol for Tate pairing computation over BN-Curves to achieve Version 5. It should meet the following requirements:

1. The system can recognize BN-Curves as pairing friendly curves;
2. The system can figure out which type of curves are passed in for pairing;
3. The system can still calculate Tate pairing through Miller's algorithm or the Elliptic Nets algorithm for supersingular curves;
4. The system can now calculate Tate pairing over BN-curves for both Miller's algorithm and Elliptic Nets algorithm;
5. The performance of the system should be enhanced by exploiting an optimized version of Miller's algorithm designed specially for BN-Curves.

In order to achieve this, we decided to put the Elliptic Net algorithm and the optimized Miller's approach separately into two classes. Each class can accept both types of pairing friendly curves and perform just one particular algorithm for Tate computation.

Tate Pairing through Miller's Algorithm

As mentioned in Section 4.1, the drawback of the IBE system [19] is that it can only accept supersingular curves and perform the standard (non-optimal) Miller's algorithm for Tate pairing. Considering the maintenance of [19] with its related applications, it is better to create a new class of Tate pairing

computation for our purpose. Figure 4.7 compares the existing TatePairing class in [19] and the new TatePairing class in our *csi.crypto.pairing* package.

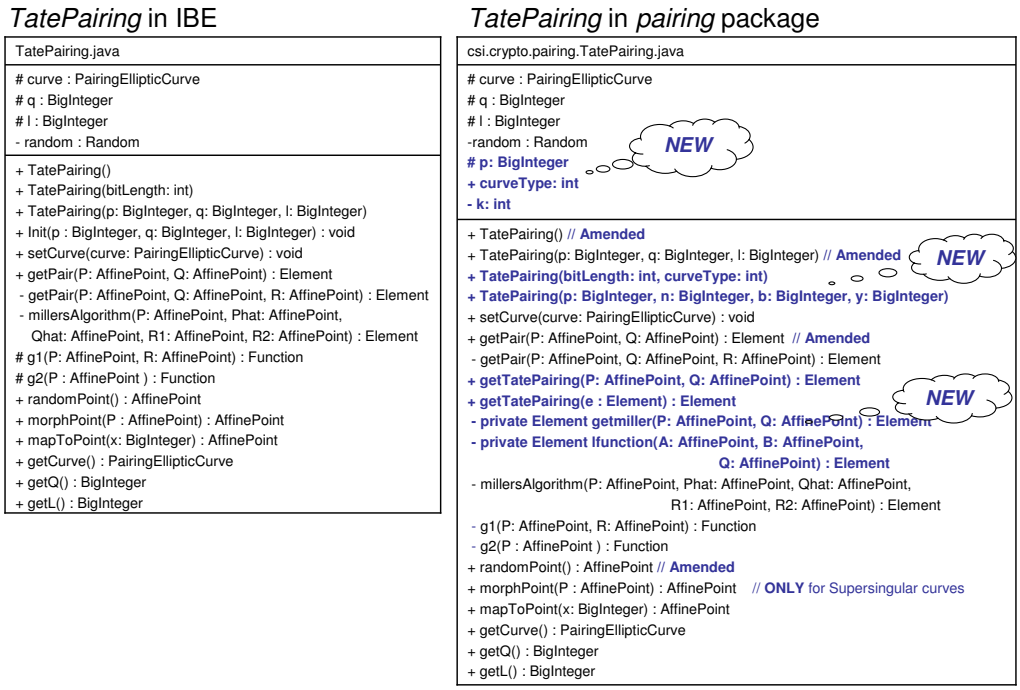

Figure 4.7: Comparison of the Two TatePairing Classes

In the new class, we kept the original settings and pairing computations for supersingular curves much the same as they existed previously. We added several new attributes and methods and we modified some existing functions for consistency. The attribute p represents the prime for field. An integer variable $curveType$ is used to mark the type of curves. It is set to 0 for supersingular curves and 1 for BN-Curves. Although we have only two types of curves, we chose $curveType$ as an integer rather than boolean such that we may add a third or more types of pairing friendly curves into this class

with minimal amendment in the future. The private integer k represents the embedded degree of the curves and cannot be accessed outside the class. The *initial()* method is removed in the new class and its functionality is merged into the two existing constructors. Since these two constructors were designed for the setup of supersingular curves, they are also affected by the new attributes. The proper settings of p, *curveType* and k need to be configured inside the constructors as well. Additionally, we inserted two more constructors: one allows the class to generate a specified type of curves with specified length and the other is used to directly set a customized BN-Curve. The existing two *getPair()* methods calculate and return a coset value of the Tate (i.e. τ'). In order to obtain the exact Tate value τ, we added two new public methods *getTatePairing()*: one can accept two affine points P, Q as input and output $\tau(P, Q)$, and the other accepts a corset value τ' as input and outputs the corresponding τ. There are two more extra private methods called *getmiller()* and *lfunction()*. The former implements the optimal Miller's algorithm for BN-Curves in [18] and the latter is the line function used inside this optimized Miller's algorithm. Both the methods are declared as private since they are only internally invoked during Tate computation for BN-Curves. The public *getPair()* and *randomPoint()* methods were adapted as well such that they can automatically detect the current type of curves and execute the correct program. Note that the private *getPair()* does not affect by the new system as it only invoked by supersingular curves. Moreover, the methods *g1()* and *g2()* also change their access modifiers from *protected* to *private* since there is no need to call them outside the class. The method *morphPoint()* implements Equation 2.11 and therefore it only works for supersingular curves. If a BN-Curve is detected, then this method will throw an exception at run time.

Tate Pairing through Elliptic Nets Algorithm

Up to now we have the Elliptic Nets algorithm in *csi.crypto.EllipticNets* package and BNCurve implementation in *csi.crypto.pairing* package; We can get Tate pairing on a BN-Curve via Miller's algorithm. The next step is to apply the Elliptic Nets algorithm to BN-Curves. We should check that:

1. BN-Curves can be passed to create an Elliptic Net;
2. Tate pairing over BN-Curves can be calculated with the Elliptic Net algorithm.

By reviewing the TatePairingViaENet class in Section 4.2.4, we noticed that firstly this class is designed to accept any kind of EllipticCurve object. In fact, the BNCurve class is defined as a sub sub class of the EllipticCurve class. In Figure 4.5, it is clear that the BNCurve class is inherited from the PairingEllipticCurve class and the PairingEllipticCurve class is directly inherited from the EllipticCurve class. Secondly, this class also calculates the embedded degree k within the *getTatePairing()* method. To avoid this duplicated calculation we can add a conditional statement into the method itself as following:

```
public Element getTatePairing(Element e){
int k;
// New if-else-statement here for BNCurve:
if(curve instanceof BNCurve) k=6;
else{
 // calculate the embedded degree k as before
 ...
}
....
```

```
}
```

Note that the TatePairingViaNet class and the BNCurve class are in different packages so we need to import the BNCurve at the beginning of this class to allow the class recognize the BNCurve object. There is no other amendment in this class.

4.5 Complete System

By achieving Version 5, the whole system has two choices for Tate pairing computation. It is capable to configure a preferred algorithm before computing Tate pairing as shown in Figure 4.8.

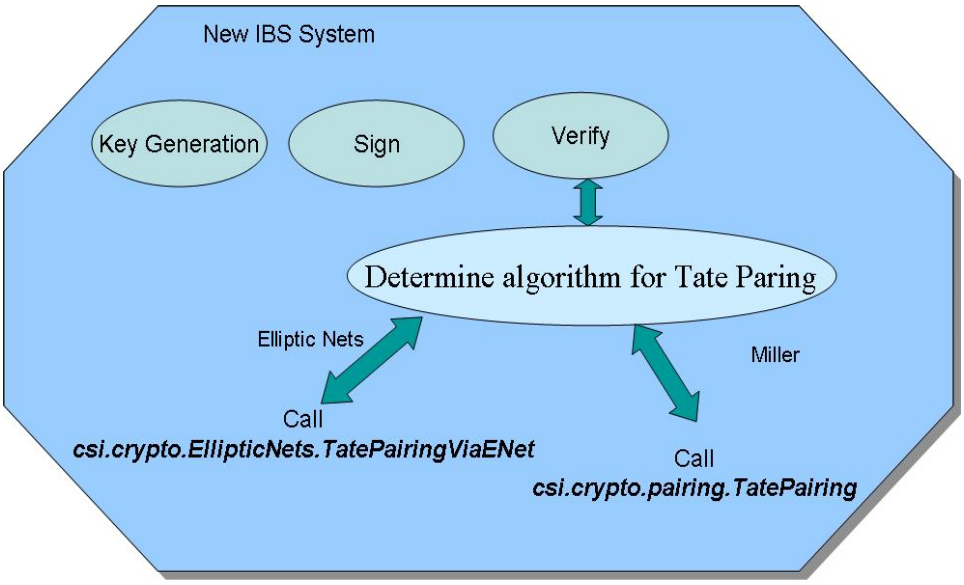

Figure 4.8: System View

4.6 Summary

In this chapter, we have provided all the detail of the implementation of this project. We discussed how each version of the project is achieved. We gave the complete descriptions of two new packages and their classes including their properties and their behaviors. The *csi.crypto.EllipticNets* package implements the Elliptic Nets theory with its application to the Tate pairing with both pairing friendly types of curves. The *csi.crypto.pairing* package provides Tate pairing computation from Miller's algorithm with both pairing friendly types of curves.

Chapter 5

Testing and Results

This chapter will display the testing results of the whole system and also carrying out experiments that will assess it performance. Section 5.1 provides the testing environment configurations to support the application. The set of test cases picking will benchmark against previous implementation. Also, they will use random values to ensure correctness for arbitrary input and verify them in a statistical sense. Section 5.2 describes functional testing procedures that illustrate that the Elliptic Nets algorithm can be used as an alternative way of Tate pairing computation. Section 5.3 provides a benchmark of the system performance.

5.1 Portability and Pre-settings

The main testing tool we use is Artima Suite Runner [49], which is a free open source of testing toolkit for Java applications. Before each test case, we kick off a build of the latest version of source code through Apache Ant [48]. Apache Ant is an XML-based build scripting language for building software projects. We use the Apache Ant to include the source code compiling, javadoc, test suite running and packaging as a completed build pro-

cedure. Besides the latest JRE library, some external libraries are loaded as pre-settings for our whole system: blitz.jar, eyebee.jar [19], tender-dev.jar [13], suiterunner-1.0beta7.jar [49], asrat.jar, and versioner.jar. Note that the blitz.jar that was used for testing is not the original Blitz of [12] as it has added functionality as described in Section 4.3. We have rebuilt the latest version of blitz package as well to support our system. The versioner.jar is our other internal software tool to provide a version service on a successful project build. Our product can be tested and run on the following operating systems: Ubuntu, Window XP and Window Vista. It is supported by the Java SDK 1.5, and JDK 1.6.21 [46].

5.2 User Test Cases

The point of this section is to give some sample test cases to check and verify the correct implementation of Elliptic Nets algorithm with Tate pairing over different curves. For each test case we have specific settings and expected outputs, we pass these to the system and verify whether the run-time output equals to our expected one. The testing results also indicate that the functionality of the system meets our initial requirements at the beginning of Section 4.2 and Section 4.4.3. All the test cases described in this section are wrapped in the class csi.crypto.EllipticNets.NetsTestSuite.java.

5.2.1 Test cases of EDS and EllipticNet

The test cases in this sections were initially developed for Version 1 and have been updated to support Version 2 during the development cycle. The final version also supports these test cases. These two test cases(i.e. test case 01 and 02) were drawn from tests that were published in Stang's paper [58], and they were chosen to verify that the software system can

1. produce a Elliptic Divisibility Sequence;

2. create and populate an Elliptic Net.

TestCase ID:	01	
TestCase Name:	Unit Test of Elliptic Divisibility Sequence	
Description:	This is Stange's example in ECC'07 [58]	
Steps:	Define Elliptic Curve as: $y^2 + y = x^3 + x^2 - 2x$ *over* $\mathbb{F}_5$ Define Affine point $P = (0, 0)$ on the curve Create Elliptic Divisibility Sequence with EDS myeds = new EDS(curve, p); Print the first two blocks on the screen Print the first 30 elements on the screen	
Actual Input: (Pseudo Code)	Fp fp = new Fp(5); curve = new EllipticCurve(0, 1, 1, -2, 0, fp); P = new AffinePoint(0, 0)	
Verifications:	**Expected Output:**	**Actual Output:**
	The printed results should equal to the example in Section 2.2.1	Yes, they are equal.

Table 5.1: Test case of class EDS

TestCase ID:	02	
TestCase Name:	Unit Test of Elliptic Net	
Description:	This is Stange's example in ECC'07 [58]	
Steps:	Define Elliptic Curve as: $y^2 + y = x^3 + x^2 - 2x$ *over* $\mathbb{F}_5$ Transform the curve to Weierstrass Form Define Affine point $P = (2, 3), Q = (3, 3)$ Create Elliptic Net with EllipticNet en = new EllipticNet(curve, p, q); Print the first 5 blocks on the screen	
Actual Input: (Pseudo Code)	Fp fp = new Fp(5); curve = new EllipticCurve(0, 1, 1, -2, 0, fp); P = new AffinePoint(2, 3); Q = new AffinePoint(3, 3)	
Verifications:	**Expected Output:**	**Actual Output:**
	The printed results should equal to the example in Figure 2.1	Yes, they are equal.

Table 5.2: Test case of class EllipticNet

5.2.2 Test Cases of Tate Pairing Via Elliptic Nets algorithm

In this section, we test the Elliptic Net algorithm for Tate pairing.

Test Case 03 checks to see if the software can correctly compute a value for the Tate pairing given a finite field, Elliptic curve derived from this and an affine point P. Again, an example of a correct affine point value is taken from Stange's paper [58].

TestCase ID:	03	
TestCase Name:	Tate Pairing TestCase 1	
Description:	This is Stange's example in ECC'07 [58]	
Steps:	Define Elliptic Curve as: $y^2 + y = x^3 + x^2 - 2x$ *over* $\mathbb{F}_{73}$ Transform the curve to Weierstrass Form Define Affine point $P = (2, 3)$ Create TatePairingViaENet object as: TatePairingViaENet tp = new TatePairingViaENet(testCurve); Calculate order of P... BigInteger order = tp.getOrderOfPoint(P); Call for pairing... Element t = tp.getPair(P);	
Actual Input: (Pseudo Code)	Fp fp = new Fp(73); curve = new EllipticCurve(0, 1, 1, -2, 0, fp); P = new AffinePoint(2, 3);	
Verifications:	**Expected Output:**	**Actual Output:**
	Order of $P = 9$ $\tau'(P, P) = 24$	$order = 9$ $t = 24$

Table 5.3: Test case of Elliptic Nets Algorithm 1

Test case 04, uses a result published in Washington's book [62] to verify that the software can compute a Tate pairing and modified Tate pairing using similar input parameters to the previous test case.

TestCase ID:	04	
TestCase Name:	Tate Pairing TestCase 2	
Description:	Example in [62], Chapter 11, page 345	
Steps:	Define Elliptic Curve as: $y^2 = x^3 - x + 1$ *over* $\mathbb{F}_{11}$ Define Affine point $P = (3, 6)$ Create TatePairingViaENet object as: TatePairingViaENet tp = new TatePairingViaENet(testCurve); Calculate order of P... BigInteger order = tp.getOrderOfPoint(P); Call for pairing... get $\tau'(P, P)$... Element t = tp.getPair(P); get $\tau'(P, P)$... Element e = tp.getTatePairing(P);	
Actual Input: (Pseudo Code)	Fp fp = new Fp(11); curve = new EllipticCurve(-1, 1, fp); P = new AffinePoint(3, 6);	
Verifications:	**Expected Output:**	**Actual Output:**
	Order of $P = 5$ $\tau'(P, P) = 5$ $\tau(P, P) = 3$	$order = 5$ $t = 5$ $e = 3$

Table 5.4: Test case of Elliptic Nets Algorithm 2

Test case 05 checks that software can applied with supersingular curves. The input parameters are taken from [19].

TestCase ID:	05	
TestCase Name:	Tate Pairing TestCase 3	
Description:	test Supersingular curves in [19]	
Steps:	Define Elliptic Curve as: $curve : y^2 = x^3 + x \ \ over \ \ \mathbb{F}_{43}$ Set up the parameters in IBE... ModifiedTatePairing pair = new ModifiedTatePairing(); pair.init(43, 11, 4); Define Affine point $P = (23, 8)$ and $Q = (23, 8)$ Calculate Tate pairing with Miller's algorithm in [19] ...: Element e = pair.getPair(P, Q); Calculate Tate Pairing with Elliptic Nets Algorithm... EllipticCurve curve = pair.getCurve(); TatePairingViaENet tp = new TatePairingViaENet(curve); Element wp = tp.getTatePairing(P, Q); Adjust Q... Q = pair.morphPoint(P); Call Tate Pairing with Elliptic Nets Algorithm again... Element te = tp.getTatePairing(P, Q);	

Actual Input:
(Pseudo Code)
	Fp fp = new Fp(43);
	pair.intial(fp, 11, 4); //using IBE parameters settings in [19]
	P = Q = new AffinePoint(23, 8);	

Verifications:	**Expected Output:**	**Actual Output:**
	$e = \tau(P, Q) = 11 + 3i$ wp is wrong value. After adjustment, $Q = (20 + 0i, 0 + 8i)$ $te = \tau(P, Q) = e$	$e = 11 + 3i$ $wp \neq e$ Yes, it is. $te = 11 + 3i = e$

Table 5.5: Test case of Elliptic Nets Algorithm 3

Test case 06 tests the customized quadratic non residue settings of the software. It is derived from Version 3 of the software in Section 4.3. It is the example in [15].

TestCase ID:	06	
TestCase Name:	Tate Pairing TestCase 4	
Description:	Example in [15], Chapter 16 Test the evolution of customized *nonResidue* setting in Section 4.3.2	

Steps:	Define Elliptic Curve as: $curve : y^2 = x^3 + 6$ *over* $\mathbb{F}_{13}$
	Define Affine point $P = (2, 1)$ and $Q = (10 + 3i, 11 + 2i)$
	Set the specified *nonResidue*...:
	
	// We have prime = 13;
	Fp2 fp2 = new Fp2(prime);
	fp2.setNonResidue(Constant.TWO);
	
	Set up for Tate Pairing with Elliptic Nets Algorithm...
	
	TatePairingViaENet tp = new TatePairingViaENet(curve);
	
	Calculate order of P...
	
	BigInteger order = tp.getOrderOfPoint(P);
	
	Call for pairing...
	
	Element e = tp.getTatePairing(P, Q);	

Verifications:	**Expected Output:**	**Actual Output:**
	order of $P = 7$ $\tau(P, Q) = 4 + i$	$order = 7$ $e = 4 + 1i$

Table 5.6: Test case of Elliptic Nets Algorithm 4

Test case 07 tests that the software can apply to BN-Curves. This test case sets up the system parameters from the example in [3]. generates random affine points pt and Q, and passes the two points for Tate pairing computation through both algorithms.

TestCase ID:	07	
TestCase Name:	Tate Pairing TestCase 5	
Description:	Tate Pairing over BN-Curves The curve settings came from the appendix of [3]	
Assume:	The points *pt* and *qhat* used in the test are valid points	

Steps:	Define BN-Curve as...
	
	BigInteger p = new BigInteger
	("14615016244967902651454485899207854937172588908 19");
	BigInteger n = new BigInteger
	("14615016244967902651454473809949711884993000276 13");
	Fp fp = new Fp(p);
	BigInteger b = Constant.THREE;
	BigInteger y = Constant.TWO;
	BNCurve bc = new BNCurve(b, fp, n, y);
	
	Define corresponding Twisted curve as...
	
	Fp2 fp2 = new Fp2(p);
	Fp2Element temp = fp2.element(
	Constant.EIGHT.negate(), Constant.EIGHT);
	Element si = temp.modInverse();
	TwistedCurve c = new TwistedCurve(BigInteger.ZERO,
	(Fp2Element)(temp.multiply(b)));
	c.setSi((Fp2Element)si);
	bc.setTwistedCurve(c);
	
	Define Affine point $pt \in bc$ and $qhat \in c$
	Morphmap $qhat \rightarrow Q$...
	Calculate Tate pair via Miller's Algorithm...
	
	TatePairing tp1 = new TatePairing(p, n, b, y);
	tp1.setCurve(bc);
	Element em = tp1.getTatePairing(pt, Q);
	
	Calculate Tate pairing via Elliptic Nets Algorithm...
	
	TatePairingViaENet tp2 = new TatePairingViaENet(bc);
	tp2.setOrderOfPoint(n);
	Element ee = tp2.getTatePairing(pt, Q);	

Verifications:	**Expected Output:**	**Actual Output:**
	pt has order *n* *qhat* has order *n* $Q \in bc$ $em == ee$	YES YES YES $em == ee$

Table 5.7: Test case of Elliptic Nets Algorithm 5

5.2.3 Test Cases with a Random Input Value

The previous testing results show that for a particular input to the system, the Elliptic Net algorithm can be used as a second approach to Tate pairing. In this section, we test random inputs to the system. The test cases will simulate arbitrary input from a user and check whether the algorithm works with such unknown input parameters. The random numbers are derived using Java.Math.BigInteger class. With a specified length of the integer (i.e. number of *bits*), the chosen number is from a uniform distribution over the range $[0, 2^{length} - 1]$. However, the length of the integer is defined as type of Java primitive *int*, which has the maximum value of $2,147,483,647$. So, the range of the valid numbers is actually $[0, 2^{214748364} - 1]$. The BigInteger class also guarantees a non-negative result for modular operations in Java [46]. This is acceptable as only positive integers are used for cryptography [37].

TestCase ID:	08	
TestCase Name:	Tate Pairing TestCase 6	
Description:	Tate Pairing over Random Supersingular Curves curve length (i.e. bit-length of p) is chosen from 150 - 260 bits p value is randomly generated by the system at run time	

Steps:	Set up for the test case...
	
	int n=25;
	int bits[] = new int[n];
	Element e1=null; // store Tate from Miller's algorithm
	Element e2=null; // store Tate from Elliptic Nets algorithm
	String check[] = new String[bits.length]; // store comparison result
	
	Calculate and compare Tate Pairing via both algorithm...
	
	for(int i=14; i<n; i++){
	bits[i] = 10*(i+1);
	TatePairing tp = new TatePairing(bits[i],0);
	EllipticCurve curve = tp.getCurve();
	AffinePoint P = tp.mapToPoint(BigInteger.ONE);
	curve.toWeierstrassEqn();
	e1 = tp.getPair(P, P);
	TatePairingViaENet tpe = new TatePairingViaENet(curve);
	tpe.setOrderOfPoint(tp.getQ());
	AffinePoint Q = tp.morphPoint(P);
	e2 = tpe.getTatePairing(P, Q);
	verify(e1.equals(e2));
	if(e1.equals(e2)) check[i] = "PASS";
	else check[i] = "FAILED";
	}
	
	Display the content of array *check* on screen	

Verifications:	**Expected Output:**	**Actual Output:**
	The screen should show the message "PASS" for each length setting.	All "PASS" printed. see Table 5.9

Table 5.8: Test case of Random Value for Supersingular Curves

The Table 5.9 displays the real run-time result of test case in Table 5.8. The first column is the bit length of p passed to the system and the second column is the associated information of PASS/FAILED stored in the array *check*.

bits	**Result**
150	PASS
160	PASS
170	PASS
180	PASS
190	PASS
200	PASS
210	PASS
220	PASS
230	PASS
240	PASS
250	PASS
260	PASS

Table 5.9: Test Result for TestCase ID: 08

Similarly, Table 5.11 displays the real run-time result of test case in Table 5.10.

TestCase ID:	09	
TestCase Name:	Tate pairing TestCase 7	
Description:	Tate Pairing over BN-Curves with random points curve length (i.e. bit-length of p) is chosen with 160, 192, and 256 bits BN-Curve settings are generated by the system at run time The points P and Q are randomly generated by the system at run time	

Steps:	Set up for the test case...
	
	int bits[] = 160, 192, 256;
	Element e1=null; // store Tate from Miller's algorithm
	Element e2=null; // store Tate from Elliptic Nets algorithm
	String check[] = new String[bits.length]; // store comparison result
	
	Calculate and compare Tate Pairing via both algorithm...
	
	for(int i=0; i<bits.length; i++){
	TatePairing tp = new TatePairing(bits[i],1);
	BNCurve curve = (BNCurve)tp.getCurve();
	curve.toWeierstrassEqn();
	AffinePoint P = curve.randomPoint();
	AffinePoint Q = curve.randomQ();
	e1 = tp.getPair(P, P);
	TatePairingViaENet tpe = new TatePairingViaENet(curve);
	tpe.setOrderOfPoint(tp.getQ());
	e2 = tpe.getTatePairing(P, Q);
	verify(e1.equals(e2));
	if(e1.equals(e2)) check[i] = "PASS";
	else check[i] = "FAILED";
	}
	
	Display the result on screen	

Verifications:	**Expected Output:**	**Actual Output:**
	The screen should show the message "PASS" for each length setting.	All "PASS" printed. See Table 5.11

Table 5.10: Test case of Random Values for BN-Curves

bits	**Result**
160	PASS
192	PASS
256	PASS

Table 5.11: Test Result for TestCase ID: 09

5.2.4 Condition Testing (White box testing)

All the previous test cases provide functional testings of the Elliptic Net algorithm and implicate that the Elliptic Net algorithm can provide Tate pairing computation once the system parameters are appropriately set. The csi.crypto.pairing.TatePairing class is developed to set up and compute Tate pairing with Miller's algorithm over both the pairing friendly curves (see Section 4.4.3). The condition testing here is to test whether this class can automatically determine and configure the curve settings for Miller's algorithm. This class has 4 constructors depends on the number and type of inputs. In short the specification of them are:

1. TatePairing()

   This is the default constructor and it will set up a random 250-bit length supersingular curve environment.

2. TatePairing(BigInteger p, BigInteger q, BigInteger l)

   This is designed for a customized setting of the supersingular curve environment.

3. TatePairing(BigInteger p, BigInteger n, BigInteger b, BigInteger y)

   This is designed for the customized setting of the BN-Curve environment.

4. TatePairing(int bitLength, int curveType)

   This is designed for random curves with the length specified by the first argument *bitLength* and type of curve specified by the second argument *curveType*. This *curveType* must be either 0 or 1 but nothing else.

This condition testing is focused on the 4^{th} constructor. Figure 5.1 shows a control flow graph for these constructor. It hides the information of the detailed setting up for supersingular curves and BN-Curves as we are only interested in the condition and decision part of the method.

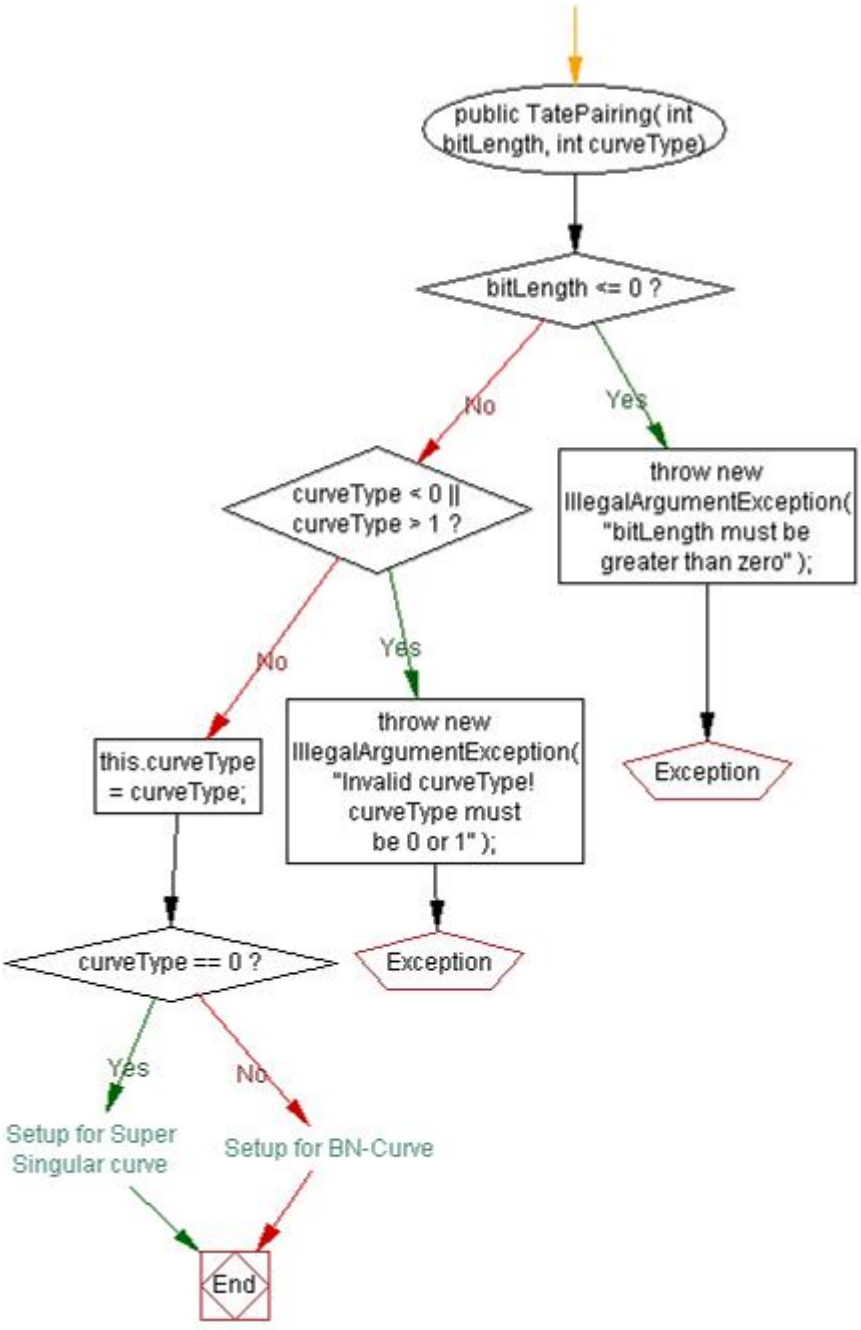

Figure 5.1: Control Flow Graph of TatePairing(int bits, int curveType) Generated by Visustin [1]

According to the program show in Figure 5.1, we can partition the inputs and design the test cases depending on the partitions as follows:

TestCase ID:	**Inputs:**		**Expected Output:** (Exception / Set up)
	bitLength	*curveType*	
10	0	0	Exception due to $bitLength = 0$
11	-1	0	Exception due to negative *bitLength*
12	40	0	Set up a 40-bit supersingular curve
13	40	1	Set up a 40-bit BN-Curve
14	60	2	Exception due to invalid *curveType*
15	80	-1	Exception due to invalid *curveType*

Table 5.12: Test cases for *TatePairing(int bitLength, int curveType)*

With the above input partitions, we have the following corresponding run-time results in Table 5.13 and all the results meet our expected outputs in Table 5.12.

TestCase ID	**Actual Output on Screen**
10	Error: java.lang.IllegalArgumentException: bitLength must be greater than zero.
11	Error: java.lang.IllegalArgumentException: bitLength must be greater than zero.
12	Succeed. Ready for Tate pairing...
13	Succeed. Ready for Tate pairing...
14	Error: java.lang.IllegalArgumentException: Invalid curveType! curveType must be 0 or 1
15	Error: java.lang.IllegalArgumentException: Invalid curveType! curveType must be 0 or 1

Table 5.13: Test Result for Test Cases in Table 5.12

5.3 Performance Test

The efficiency of the system are also important. According to the theory [58], the Elliptic Nets algorithm is a polynomial time consuming algorithm like Miller's algorithm. It is based on an efficient method for calculating terms in the Elliptic Net. It requires no inversions, while Miller's algorithm in

affine coordinates requires one or two $\mathbb{F}_p$ inversions per step. The inversions can be costly depending on the implementation.To benchmark the Elliptic Nets algorithm and compare with Miller's algorithm, we need some efficiency testing to measure the real timing cost of both algorithms. The inputs used for measurement should be randomly generated by the system itself. As the program is developed in Java, the speed somewhat depends on the actual JVM. A specified Java method, *System.currentTimeMillis()*, can be used to evaluate the timing costs of there algorithms. This method is available in the current JDK [46].

5.3.1 Comparison over supersingular curves

We compared the speed of two algorithms over supersingular curves with the following tests.

TestCase ID: 16 Benchmark for Elliptic Nets Algorithm for Supersingular curves

Requirement:

1. Calculate average time for Miller's algorithm in [13].
2. Calculate average time for Elliptic Nets algorithm with the same settings.
3. Output the results to obtain a chart of comparison.

Environment:

- **CPU:** Intel Core(TM)2 Duo CPU T7300 2.00Ghz
- **RAM:** 2.00GB
- **Operating System:** 32-bit Window Vista Service Package 2

- **Java Runtime Environment:** jre1.6.0_21
- **Others:** The computer is disconnected from the network without any other applications running
- **Single Threaded Application**

Critical Program Code: The following code will calculate an average time cost for both algorithms over random supersingular curves with length p from 150 to 250 bits and the resulted time costs will be printed on the screen.

```
public static void benchmark1() {
 int bits[] = new int[n];
 long timingM; // time for Miller
 long timingN; // time for Elliptic Nets
 Element e1=null;
 Element e2=null;
 System.out.println("bitsOfP\t\tMiller\t\tEllipticNet");
 System.out.println("_____________________________________");
 for(int i=14; i<n; i++){
 bits[i] = 10*(i+1);
 System.out.print(bits[i]+"\t\t");

 // Setup for Miller's algorithm...
 ModifiedTatePairing mtp = new ModifiedTatePairing(bits[i]);
 PairingEllipticCurve curve = mtp.getCurve();
 AffinePoint P = mtp.mapToPoint(BigInteger.ONE);
 curve.toWeierstrassEqn();
```

```
// Timing for Miller's algorithm...
long before = System.currentTimeMillis();
for(int j=0; j<10; j++)
e1 = mtp.getPair(P, P);
  timingM =  (System.currentTimeMillis()-before)/10;
System.out.print(" "+timingM+"\t\t");

// Setup for Elliptic Nets algorithm...
TatePairingViaENet tpe = new TatePairingViaENet(curve);
tpe.setOrderOfPoint(mtp.getQ());
AffinePoint Q = mtp.morphPoint(P);

// Timing for Elliptic Nets algorithm...
before = System.currentTimeMillis();
for(int j=0; j<10; j++)
e2 = tpe.getTatePairing(P, Q);
  timingN = (System.currentTimeMillis()-before)/10;
System.out.println(" "+timingN);
}
}
```

Statistical Steps:

1. Run the critical program code above 20 times to obtain 20 records of the time costs;
2. Calculate the mean and the standard deviation over the 20 records;

Output result: Translate the statistical results to Table 5.14, Figures 5.2 and 5.3.

Length of p	Miller Mean(mS)	Miller StdDev(mS)	Elliptic Nets Mean(mS)	Elliptic Nets StdDev(mS)	Difference (mS)	Ratio (EllipticNets/Miller)
150	1565.25	27.39	1020.25	8.10	545	0.6518
160	1803.25	73.99	1172.05	12.69	631.2	0.6500
170	2099.3	36.09	1375.95	9.20	723.35	0.6554
180	2367.75	51.85	1553.3	9.44	814.45	0.6560
190	2634.25	56.91	1723.5	12.58	910.75	0.6543
200	3032.45	55.85	2000.85	20.16	1031.6	0.6598
210	3377.2	85.37	2229.4	13.47	1147.8	0.6601
220	3711.8	77.63	2437.4	19.26	1274.4	0.6567
230	4233.85	88.83	2790.15	20.00	1443.7	0.6590
240	4612.95	64.70	3052.9	87.46	1560.05	0.6618
250	5210.05	578.97	3352.3	93.04	1857.75	0.6434

Table 5.14: Raw Benchmark for Supersingular Curve

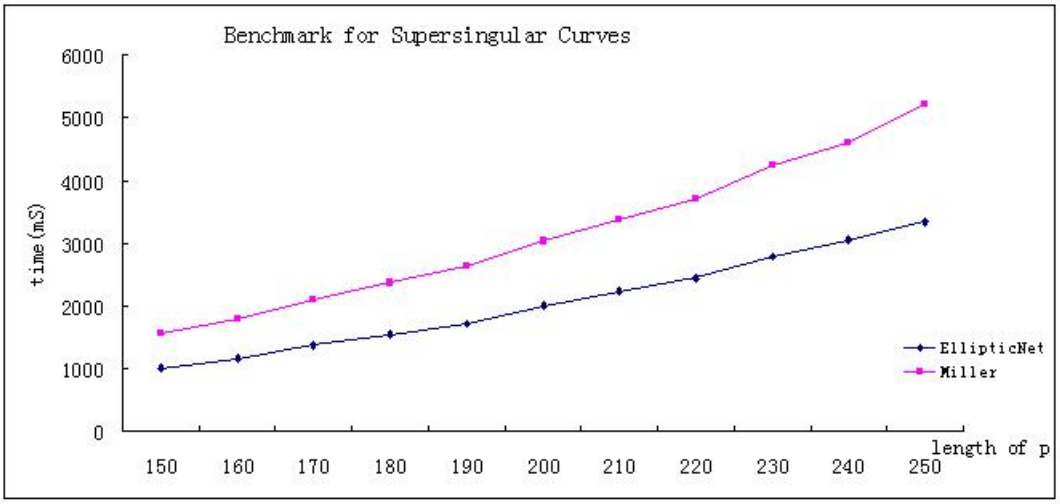

Figure 5.2: Comparison of Computation Time for Miller's Algorithm with the Elliptic Nets Algorithm on Supersingular Curves as the Length of p is increasing

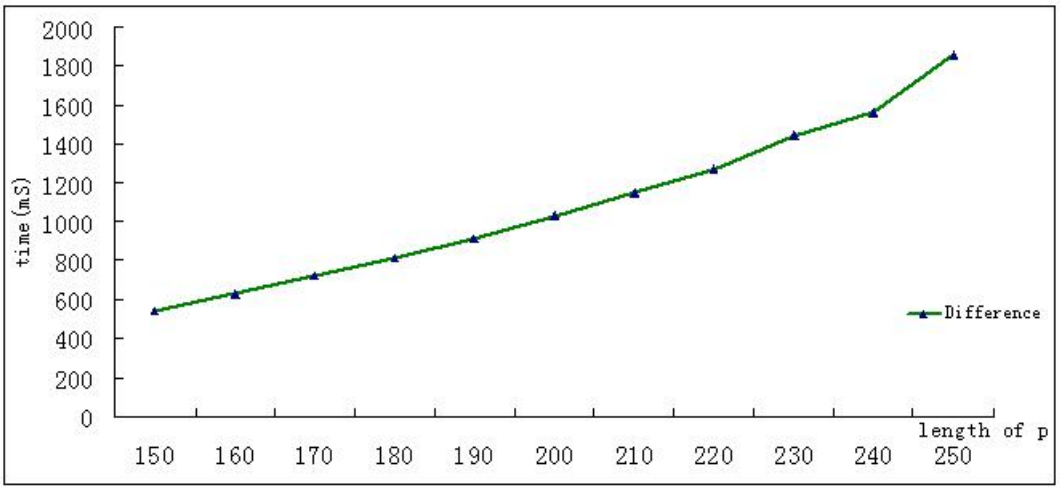

Figure 5.3: The Relationship between the Length of P and the Difference of the Time Cost between the Two Algorithms

Table 5.14 displays the result of the test case 16, where the first column is for the bit length of p passed to the system, the second and third columns hold the mean and the standard deviation of the real time costs of Miller's algorithms, the fourth and the fifth hold the mean and the standard deviation of the time costs of the Elliptic Nets algorithm, the next column shows the difference between the mean time costs of the two algorithms and the final column is the ratio between the mean time costs of the two algorithms. It shows that the Elliptic Nets algorithm is always more efficient than Miller's algorithm in [13]. It nearly saves about 1/3 time cost on average. Thus, as an alternative algorithm for Tate pairing, the Elliptic Nets algorithm can speed up the IBE application in [13]. Figure 5.2 is a graphical representation of Table 5.14. Note that there was some unavoidable noise in the testing environment due to the testing platform itself from sources such as memory and proccess scheduling by the operating system. Figure 5.3 expresses the sixth column of Table 5.14. By taking the difference, it decreased the noise and approximately shows that as the length p grows, or equivalently to say

that as the cryptographic key size is increasing, the speed gap between the two algorithms becomes wider and the Elliptic Net algorithm has obvious benefits of the efficiency.

5.3.2 Comparison over BN-Curves

The test case (Id:17) for the BN-Curves benchmark is similarly to the previous one for supersingular curves. Table 5.15 and Figure 5.4 give the raw timing cost of the two algorithms for Tate pairing over random BN-Curves and random points on the curves.

Length of p	Miller Mean(mS)	Miller StdDev(mS)	Elliptic Nets Mean(mS)	Elliptic Nets StdDev(mS)	Difference (mS)	Ratio (EllipticNets/Miller)
160	21149.2	349.67	25312.5	291.39	-4163.3	1.1969
192	34343.5	370.22	41048	160.13	-6704.5	1.1952
256	64197.3	247.75	76844.5	334.69	-12647.2	1.1970

Table 5.15: Raw Benchmark for BN-Curves

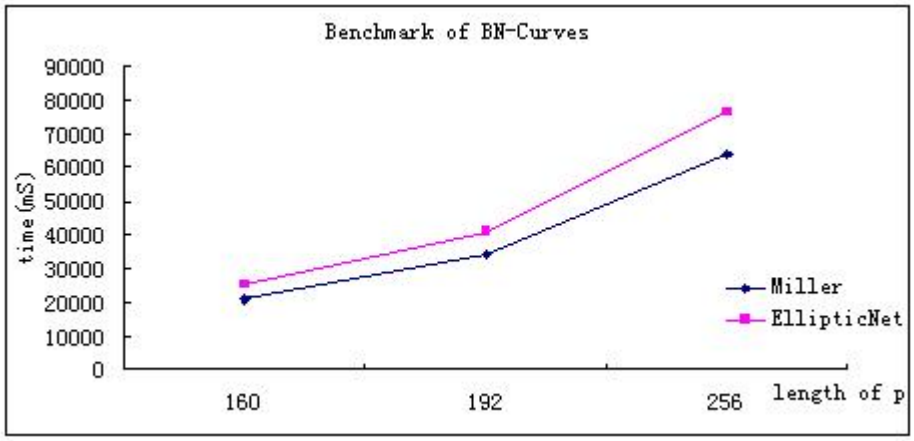

Figure 5.4: Comparison of Computation Time for Miller's Algorithm with the Elliptic Nets Algorithm on BN-Curves as the Length of p is increasing

Notice that there are negative values in the sixth column (i.e. $Difference$)

in Table 5.15 which means that the Elliptic Nets algorithm is slower than the optimized Miller's algorithm for BN-Curves. In practice, it takes nearly 20% more time than Miller's algorithm on average. This situation means that BN-Curves is only suitable for deployment in an environment that offers very significant computational resources. However, they would currently be unsuitable for performing a cryptographic encoding on a computational-power limited device. To get an idea of the relationship between execution time and a user's expectations, Nielson [42] notes that an execution time of about 1 second for a typical software process that does not supply feedback while it is executing is acceptable, otherwise some form of feedback is recommended if the completion time reaches up to 10 seconds. If the process was actually taking of the order of minutes or even hours to execute it would be totally unsuitable in practice.

5.4 Summary

The chapter illustrated the testing process of the software system, which covered unit testing, functional testing, white box testing and performance testing. The testing results proved that the Elliptic Net algorithm is more efficient that the standard Miller's algorithm on Supersingular curves and thus it can be a suitable alternative choice of the Tate pairing computation. However, both algorithms suffer a heavy timing cost for BN-Curves, which limit the application of BN-Curves for cryptographic protocols.

Chapter 6

Conclusion

6.1 Summary

Recall the initial motivation of the project in Section 1.6, in this thesis we have attempted to find answers to these question through our development of the system. We have considered alternative approaches to the Tate pairing computation currently used in the BIO-IBS system. It was proposed to offer both an optimized version of Miller's algorithm for the Tate pairing and to allow for the use of Elliptic Nets. The design of the existing system was thus evolved to incorporate our new Elliptic Nets API. A software engineering approach was adopted for this system development and the integration of the new functionality was planned to be carried out in separate stages. As part of the Elliptic nets API, BN-Curves were added as a second type of pairing friendly curves available in the cryptosystem. The initial idea of adding BN-Curve is to enhance the security of the cryptosystem. Black and white box testing was carried out to validate the correct operation of the new system. Performance comparisons were then made between the two pairing approaches over both type of curves. These showed that the new

Elliptic Nets approach compares favorably with the existing Miller based system particularly for supersingular curves. A time speedup of the order of 20% on average was obtained. Therefore, the Elliptic Net algorithm can improve the efficiency of the original BIO-IBS system in [13]. However, this was shown not to be the case when using BN-Curves as there was an increase in the time taken which indicates that they are not suitable for the current system.

6.2 Future Work

In the short term there are some aspects of the application that could be completed in the future. Firstly, a switch function between two algorithms is missing. The system would benefit from some extra functions that could detect the current algorithm and decide whether should the second one being invoked at run time. Such a switch should be automatically performed in the background without having to notify the front-end user but with some notification to the sever side. Secondly, the system should provide some user-friendly information to the front-end when the service is stopped. Thirdly, we need to include some capacity or stability testing for the system before it employed into some J2EE services. Moreover, we used java.math.BigInteger as the core base of data type. This data type is not supported by J2ME, which means a new data structure (e.g. byte-based type) should be considered for real applications on mobile devices.

On the other hand, from the cryptographic side, as we mentioned before, the heavy computation associated with pairings is the main issue in pairing-based cryptography. Although we now have two algorithms for Tate computation but since they both have a polynomial time cost, the efficiency of pairing computation is still an open problem in pairing based cryptography.

Secondly, Section 5.3 also indicated that as a new approach, the Elliptic Nets algorithm with BN-Curves is slower than optimized Miller's algorithm. There may be some techniques behind or beyond which can be used to optimize the new algorithm to make it a more efficient alternative choice for Tate pairing computation. Moreover, with the development of division polynomial theory with twisted Edwards curves [40], the hypothesis that whether Elliptic Nets algorithm can be refined for twisted Edwards curves as well is another open question. All these ideas should be addressed in a future long-term research study.

Bibliography

[1] Aivosto. Visustin v6 Flow chart generator. `http://www.aivosto.com/visustin.html`.

[2] A.Joux. A one round protocol for tripartite diffie-hellman. In *ANTS-IV: Proceedings of the 4th International Symposium on Algorithmic Number Theory*, pages 385–394, London, UK, 2000. Springer-Verlag.

[3] Paulo S.L.M. Barreto and Michael Naehrig. Pairing-friendly elliptic curves of prime order. *Proceedings of SAC 2005, volume 3897 of LNCS*, pages 319–331, 2005.

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

[5] I. Blake, G. Seroussi, and N. Smart. *Elliptic Curves in Cryptography.* Cambridge University Press, 1st edition, 1999.

[6] I.F. Blake, G. Seroussi, and N.P. Smart. *Elliptic Curves in Cryptography.* LMS lecture note; 265. Cambridge ; New York : Cambridge University Press, 1999.

[7] D. Boneh and M. Franklin. Identity Based Encryption from the Weil Pairing. *SIAM Journal of Computing, Vol. 32, No. 3*, pages 586–615, 2001.

[8] D. Boneh, C. Gentry, B. Lynn, and H. Shacham. Aggregate and Verifiably Encrypted Signature from Bilinear Maps. *Proceedings from Advances in Cryptology - EuroCrypt*, 2003.

[9] D. Boneh, B. Lynn, and H. Shacham. Short Signatures from the Weil Pairing. *Proceedings from Advances in Cryptology - Asiacrypt*, 2001.

[10] Dan Boneh. The decision diffie-hellman problem. *LNCS*, 1432:48–63, 1998.

[11] Virtual Box. Virtual Box. `http://www.virtualbox.org`.

[12] A. Burnett, F. Byrne, T. Dowling, and A. Duffy. Elliptic Curve Arithmetic API. `http://www.crypto.cs.nuim.ie/software/`.

[13] A. Burnett, F. Byrne, T. Dowling, and A. Duffy. A Biometric Identity Based Signature Scheme. *International Journal of Network Security*, 5:317–326, 2007.

[14] Jaemin Choi, Jongsung Kim, Jaechul Sung, Sangjin Lee, and Jongin Lim. Related-key and meet-in-the-middle attacks on triple-des and des-exe. In *Computational Science and Its Applications - ICCSA 2005*, volume 3481 of *Lecture Notes in Computer Science*, pages 567–576. Springer Berlin / Heidelberg, 2005.

[15] Henri Cohen, Gerhard Frey, Roberto Avanzi, Christophe Doche, Tanja Lange, Kim Nguyen, and Frederik Vercauteren. *Handbook of Elliptic and Hyperelliptic Curve Cryptography*. Chapman & Hall/CRC, 2005.

[16] Microsoft Corporation. Windows. `http://www.microsoft.com/en/us/default.aspx`.

[17] PGP Corporation. Pgp products. `http://www.pgp.com/`.

[18] Augusto Jun Devegili, Michael Scott, and Ricardo Dahab. Implementing cryptographic pairings over barreto-naehrig curves. *LNCS*, 4575:197–207, 2007.

[19] A. Duffy and T. Dowling. An Object Oriented Approach to an Identity Based Encryption Cryptosystem. *The 8th IASTED International Conference on Software Engineering and Applications*, 2004.

[20] R. Dutta, R. Barua, and P. Sarkar. Pairing-Based Cryptographic Protocol: A Survey. *Cryptology ePrint Archive*, 2004.

[21] Kirsten Eisentrager, Kristin Lauter, and Peter L. Montgomery. Fast elliptic curve arithmetic and improved weil pairing evaluation. In *Topics in Cryptology ¨C CT-RSA 2003*, pages 343–354. Springer-Verlag, 2003.

[22] David Freeman, Michael Scott, and Edlyn Teske. A taxonomy of pairing-friendly elliptic curves, 2006.

[23] Gerhard Frey and Hans-Georg Rück. A remark concerning m-divisibility and the discrete logarithm in the divisor class group of curves. *Math. Comput.*, 62(206):865–874, 1994.

[24] D. R. Graham. Incremental development: review of nonmonolithic life-cycle development models. *Inf. Softw. Technol.*, 31(1):7–20, 1989.

[25] Jeffrey Hoffstein, Jill Pipher, and Joseph H. Silverman. Ntru: A ring-based public key cryptosystem. *Lecture Notes in Computer Science*, 1423:267–288, 1998.

[26] IAIK. Iaik-jce, crypto toolkit, 2010.

[27] ECRYPT II. BlueCrypt Crypographic Key Length Recommendation, 2010. `http://www.keylength.com/en/3/`.

[28] A. Juels and M. Wattenberg. A Fuzzy Commitment Scheme. *Proceedings of the 6th ACM conference on Computer and Communications Security*, pages 28–36, 1999.

[29] N. Keller. A biometric identity based signature scheme, Jan 2010. Seminar report, Biometric and security seminar, Bonn-Aachen international center for information technology,.

[30] Sarah Knoop. Math 842: Final project 12/15/04 project: Supersingular curves and the weil pairing in elliptic curve cryptography, 2004.

[31] N. Koblitz. *A Course in Number Theory and Cryptography*. Springer, 2nd edition, 1994.

[32] K. Lauter and K. Stange. The elliptic curve discrete logarithm problem and equivalent hard problems for elliptic divisibility sequences. *LNCS*, 5381, 2009.

[33] Isaac Liu and David McGrogan. Elimination of side channel attacks on a precision timed architecture, Jan 2009.

[34] Ben Lynn. Pairing-Based Cryptography library. `http://crypto.stanford.edu/pbc/`.

[35] Ben Lynn. *On the Implementation of Pairing-Based Cryptosystems*. PhD thesis, Department of Computer Science, Stanford University, 2007.

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

[37] Alfred J. Menezes, Paul C. Van Oorschot, Scott A. Vanstone, and R. L. Rivest. Handbook of applied cryptography, 1997.

[38] V. S. Miller. The Weil Pairing, and Its Efficient Calculation. *Journal of Cryptology*, 17, 2004.

[39] Richard A Mollin. *An Introduction to Cryptography.* Discrete mathematics and its applications. Boca Raton : Chapman & Hall/CRC, 2007.

[40] Richard Moloney and Gary McGuire. Division Polynomials for Twisted Edwards Curves, July 2009.

[41] N. El Mrabet and S. Ionica. Pairing computation for elliptic curves with embedding degree 15. `http://nelmrabe.perso.info.unicaen.fr/recherche.htm`.

[42] Jakob Nielsen. *Usability Engineering.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1993.

[43] Leonardo B. Oliveira, Diego Aranha, Eduardo Morais, Felipe Daguano, Julio L¨®pez, and Ricardo Dahab. Identity-based encryption for sensor networks. In *In 5th IEEE Int¡¯l Conference on Pervasive Computing and Communications Workshops (PERCOMW ¡¯07*, pages 290–294, 2007.

[44] Paul C. Van Oorschot and Michael J. Wiener. Parallel collision search with cryptanalytic applications. *Journal of Cryptology*, 12:1–28, 1996.

[45] Oracle. Java Strategy and Directions. `http://www.oracle.com/us/technologies/java/index.html`.

[46] Oracle. Java, 2010.

[47] L. Owens, A. Duffy, and T. Dowling. An Identity Based Encryption System. *Proceedings of the 3rd International Conference on Principles and Practice of Programming in Java*, 2004.

[48] Apache Ant Project. Apache ant 1.8.1, May 2010. `http://ant.apache.org/`.

[49] Artima SuiteRunner Project. Artima SuiteRunner, 2004. `http://www.artima.com/suiterunner/index.html`.

[50] Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman. A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. *Communications of the ACM*, pages 120–126, 1978.

[51] Michael Scott. Scaling security in pairing-based protocols, 2005.

[52] Michael Scott and Paulo S. Barreto. Generating more mnt elliptic curves. *Des. Codes Cryptography*, 38(2):209–217, 2006.

[53] R. Shipsey. *Elliptic Divisibility Sequences*. PhD thesis, Goldensmiths, University of London, 2000.

[54] S.D. Galbraithand K. Harrisonand D. Soldera. Implementing the tate pairing. In *ANTS-V: Proceedings of the 5th International Symposium on Algorithmic Number Theory*, pages 324–337, London, UK, 2002. Springer-Verlag.

[55] Ljiljana Spadavecchia. *A Network-based Asynchronous Architecture for Cryptographic Devices*. PhD thesis, Institute for Computing Systems Architecture, School of Informatics, University of Edinburgh, 2005.

[56] Thomas Stahl and Markus Völter. *Model-Driven Software Development: Technology, Engineering, Management*. Wiley, Chichester, UK, 2006.

[57] Katherine E. Stange. Pari/gp scripts for tate pairing via elliptic net. `http://www.math.brown.edu/~stange/scripts.html`.

[58] Katherine E. Stange. The Tate Pairing via Elliptic Nets. *Pairing-Based Crytography - Pairing 2007*, 4575/2007, 2007.

[59] Graeme Taylor. Tate pairing computation in SAGE III, 2008. `http://maths.straylight.co.uk/archives/category/sage`.

[60] Peng Wang, Dengguo Feng, Wenling Wu, and Liting Zhang. On the correctness of an approach against side-channel attacks. In *ISPEC '09: Proceedings of the 5th International Conference on Information Security Practice and Experience*, pages 336–344, Berlin, Heidelberg, 2009. Springer-Verlag.

[61] Morgan Ward. Memoir on elliptic divisibility sequences, 1948.

[62] L. Washington. *Elliptic Curves: Number Theory and Cryptography*. Chapman and Hall, CRC, 1st edition, 2003.

[63] F. Zhang and K. Kim. ID-Based Blind Signature and Ring Signature from Pairings. *Proceedings from Advances in Cryptology - Asiacrypt*, 2002.