Implementation of Elliptic Curve Arithmetic Operations for Prime Field and Binary Field using java BigInteger Class

Tun Myat Aung
University of Computer Studies, Yangon
Myanmar

Ni Ni Hla
University of Computer Studies, Yangon
Myanmar

Abstract—**The security of elliptic curve cryptosystems depends on the difficulty of solving the Elliptic Curve Discrete Log Problem (ECDLP). Elliptic curves with large group order are used for elliptic curve cryptosystems not to solve ECDLP. We implement elliptic curve arithmetic operations by using java BigInteger class to study and analyze any elliptic curve cryptographic protocol under large integer for prime field and binary field.**

Keywords— Implementation; Elliptic Curve Arithmetic Operations; Java Biginteger Class

I. Introduction

Elliptic Curve Arithmetic was applied on cryptography known as of Elliptic Curve Cryptography (ECC) was discovered in 1985 by Victor Miller (IBM) and Neil Koblitz (University of Washington) as an alternative mechanism for implementing public-key cryptography (PKC). ECC is a public key cryptography. In public key cryptography each user or the device taking part in the communication generally have a pair of keys, a public key and a private key, and a set of operations associated with the keys to do the cryptographic operations. Only the particular user knows the private key whereas the public key is distributed to all users taking part in the communication. Some public key algorithm may require a set of predefined constants to be known by all the devices taking part in the communication. “Domain parameters” in ECC is an example of such constants. Public key cryptography, unlike private key cryptography, does not require any shared secret between the communicating parties but it is much slower than the private key cryptography.

ECC can be used for providing the following security services:

- confidentiality,
- authentication,
- data integrity,
- non-repudiation,
- authenticated key exchange.

The recent progress in factorization and parallel processing leads to the need of larger and larger keys for public-key cryptosystems. But, the growth of keys length will do these cryptosystems slower than before. The use of ECC allows the increasing of security. In the same time, ECC decreases the overloading. ECC security consists in the difficulty to calculate logarithms in discrete fields (discrete logarithms problem): being given A (an element from a finite field) and A^x, it is practically impossible to calculate x when A is big enough. Actually, there are several cryptosystems which are based on discrete logarithms problem in multiplicative group $Z_p{}^*$. But these cryptosystems can be also defined in any other finite group, as the group of points of an elliptic curve.

The elliptic curves are suitable in applications where:

- the computing power is limited (intelligent cards, wireless devices, PC boards);
- memory size on integrated circuit is limited;
- a great speed of computing is necessary;
- digital signing and its verification are used intensively;
- signed messages have to be transmitted or memorized;
- digital bandwidth is limited (mobile communications, certain computer networks).

From the advantages of ECC usage, there can be mentioned:

- increased security: cryptographic resistance per bit is much greater than those of any public-key cryptosystem known at present time;
- substantial economies in calculus and memory needs in comparison with other cryptosystems;
- great encryption and signing speed both in software and hardware implementation;
- ECC are ideal for small size hardware implementations (as intelligent cards);
- encryption and signing can be done in separate stages.

The intense research done on public-key cryptosystems, based on elliptic curves, demonstrated that ECC are suitable for the vast majority of existing applications. An ECC with 160-bit key offers a security level equivalent with that offered by a cryptosystem based on a 1024-bit *Zp* field. Because of this, ECC provide a feasible method of implementation for a high level security system on a PC card, on an intelligent card or on a mobile communications device.

The purpose of this paper is to provide a detailed implementation for elliptic curve arithmetic operations over prime field and binary field under large integers. This work supports to implement, analyze and study any elliptic curve cryptosystems over prime field and binary field under large integers. The organization of this paper is as follows. The section 2 includes finite field arithmetic operations over prime field and binary field and their properties. In section 3, we describe in details elliptic curve arithmetic operations over prime field and binary field and their geometric properties. The section 4 illustrates the implementation of elliptic curve arithmetic operations and their experimental results. Finally, we conclude this paper by discussion on performance results in section 5.

II. FINITE FIELD ARITHMETIC

A finite field is a field containing a finite number of elements. *Fields* are abstractions of familiar number systems (such as the rational numbers Q, the real numbers R, and the complex numbers C) and their essential properties. They consist of a set F together with two operations, addition (denoted by +) and multiplication (denoted by ·), that satisfy the usual arithmetic properties:

- (F,+) is an abelian group with (additive) identity denoted by 0.
- (F\{0}, ·) is an abelian group with (multiplicative) identity denoted by 1.
- The distributive law holds: (a+b) · c = (a · c) + (b · c) for all a, b, c ∈ F.

If the set F is finite, then the field is said to be *finite*. Galois showed that for a field to be finite, the number of elements should be p^m , where p is a prime number called the *characteristic* of F and m is a positive integer. The finite fields are usually called *Galois fields* and also denoted as *GF*(p^m). If m = 1, then *GF* is called a *prime field*. If $m \geq 2$, then F is called an *extension field*. The *order* of a finite field is the number of elements in the field. Any two fields are said to be *isomorphic* if their orders are the same [4].

A. Field Operations

A field F is equipped with two operations, *addition* and *multiplication*. *Subtraction* of field elements is defined in terms of addition: for $a,b \in$ F, $a - b = a + (-b)$ where $-b$ is the unique element in F such that $b+(-b) = 0$ ($-b$ is called the *negative or additive inverse* of *b*). Similarly, *division* of field elements is defined in terms of multiplication: for a, b ∈ F with $b = 0$, $a/b = a \cdot b^{-1}$ where b^{-1} is the unique element in F such that $b \cdot b^{-1} = 1$. (b^{-1} is called the *multiplicative inverse* of *b*.)

B. Prime Field

Let p be a prime number. The integers modulo p, consisting of the integers {0,1,2, . . ., p −1} with addition and multiplication performed modulo p, is a finite field of order p. We shall denote this field by *GF(p)* and call *p* the *modulus* of *GF(p)*. For any integer a, a *mod* p shall denote the unique integer remainder r, $0 \leq r \leq p-1$, obtained upon dividing a by p; this operation is called *reduction modulo p* [1].

Example (1). (*prime field* GF(29)) The elements of GF(29) are {0,1,2, . . .,28}. The following are some examples of arithmetic operations in GF(29).

(a). Addition: 17+20 = 8 since 37 mod 29 = 8.
(b). Subtraction: 17−20 = 26 since −3 mod 29 = 26.
(c). Multiplication: 17 · 20 = 21 since 340 mod 29 = 21.
(d). Inversion: 17^{-1} = 12 since 17 · 12 mod 29 = 1.

C. Binary Field

Finite fields of order 2^m are called *binary fields* or *characteristic-two finite fields*. One way to construct *GF*(2^m) is to use a *polynomial basis representation*. Here, the elements of *GF*(2^m) are the binary polynomials (polynomials whose coefficients are in the field *GF(2)* = {0,1}) of degree at most m −1:

$$GF(2^m) = a_{m-1}x^{m-1} + a_{m-2}x^{m-2} + \cdots + a_2x^2 + a_1x + a_0 : a_i \in \{0,1\}.$$

An irreducible binary polynomial f (x) of degree m is chosen. Irreducibility of *f(x)* means that *f(x)* cannot be factored as a product of binary polynomials each of degree less than *m*. Addition of field elements is the usual addition of polynomials, with coefficient arithmetic performed modulo 2. Multiplication of field elements is performed modulo the *reduction polynomial f(x)*. For any binary polynomial *a(x)*, *a(x)* mod *f(x)* shall denote the unique remainder polynomial *r(x)* of degree less than *m* obtained upon long division of *a(x)* by *f(x)*; this operation is called *reduction modulo f(x)* [1].

Example (2). (*binary field GF*(2^4)) The elements of *GF*(2^4) are the 16 binary polynomials of degree at most 3:

0	x^2	x^3	x^3+x^2
1	x^2+1	x^3+1	x^3+x^2+1
x	x^2+x	x^3+x	x^3+x^2+x
$x+1$	x^2+x+1	x^3+x+1	x^3+x^2+x+1

The following are some examples of arithmetic operations in *GF*(2^4) with reduction Polynomial $f(x) = x^4 + x + 1$.

(a). Addition: $(x^3 + x^2 + 1) + (x^2 + x + 1) = x^3 + x$.
(b). Subtraction: $(x^3 + x^2 + 1) - (x^2 + x + 1) = x^3 + x$.
(c).Multiplication: $(x^3 + x^2 + 1).(x^2 + x + 1) = x^2 + 1$ since $(x^3 + x^2 + 1).(x^2 + x + 1) = x^5 + x + 1$ and $(x^5 + x + 1)\ mod\ (x^4 + x + 1) = x^2 + 1$.
(d). Inversion: $(x^3 + x^2 + 1)^{-1} = x^2$ since $(x^3 + x^2 + 1). x^2\ mod\ (x^4 + x + 1) = 1$.

III. ELLIPTIC CURVE ARITHMETIC

A. Elliptic Curves over Prime Field -GF(p)

The elliptic curve over finite field E(GF) is a cubic curve defined by the general Weierstrass equation: $y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6$ over *GF* where $a_i \in GF$ and *GF* is a finte field. The following elliptic curves are adopted from the general Weierstrass equation. The elliptic curve E(GF(p)) over prime field GF(p) is defined by the equation [1]:

$$y^2 = x^3 + ax + b$$

where $p > 3$ is a prime and $a, b \in GF(p)$ satisfy that the discriminant $4a^3 + 27b^2 \neq 0$ ($a_1 = a_2 = a_3 = 0$; $a_4 = a$ and $a_6 = b$ corresponding to the general Weierstrass equation).

1) Points on E(GF(p))

The elliptic curve E(GF(p)) consists of a set of points $\{P = (x, y) |\ y^2 = x^3 + ax + b, x, y, a, b \in GF(p)\}$ together with a point at infinity defined as *O*. Every point on the curve has its *inverse*. The inverse of a point (*x*, *y*) on E(GF(p)) is (*x*, -*y*). The number of points on the curve, including a point at infinity, is called its *order #E*. The pseudocode for finding the points on the elliptic curve E(GF(p)) is shown in Algorithm (1).

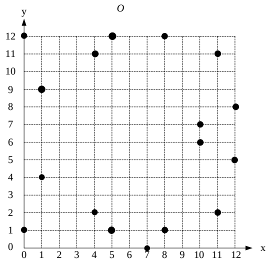
Algorithm (1). Pseudocode for finding the points on the elliptic curve E(GF(p))

```
Input: a, b, p
Output: $P_i = (x_i, y_i)$
Begin
x = 0;
while( x < p ){
   $w = (x^3 + ax + b) mod\ p$.
   If(w is perfect square in $Z_p$) output $(x, \sqrt{w})$ $(x, -\sqrt{w})$
   x = x + 1.
}
End
```

Example (3). Let p = 13 and consider the elliptic curve $E: y^2 = x^3 + 5x + 4$ defined over GF(p) where a = 5 and b = 4. Note that $4a^3 + 27b^2 = 500 + 432 = 932\ mod\ 13 = 9$, so E is indeed an elliptic curve. The points on E(GF(p)) and its graph are shown in Figure (1). The *order* of the elliptic curve $E: y^2 = x^3 + 5x + 4$ over GF(13) is 17.

(0, 2)	(0, 11)
(1, 6)	(1, 7)
(2, 3)	(2, 10)
(4, 6)	(4, 7)
(6, 4)	(6, 9)
(8, 6)	(8, 7)
(10, 1)	(10, 12)
(11, 5)	(11, 8)

Points



Graph

Figure (1). Points on $E: y^2 = x^3 + 5x + 4$

2) Arithmetic Operations on E(GF(p))

There is a rule, called the *chord-and-tangent rule*, for adding two points on an elliptic curve E(GF(p)) to give a third elliptic curve point. Together with this addition operation, the set of points E(GF(p)) forms a group with *O* serving as its identity. It is this group that is used in the construction of elliptic curve cryptosystems. The addition rule is best explained geometrically. Let $P = (x_1, y_1)$ and $Q = (x_2, y_2)$ be two distinct points on an elliptic curve *E*. Then the *sum* of *P* and *Q*, denoted $R = (x_3, y_3)$, is defined as follows. First draw the line through *P* and *Q*; this line intersects the elliptic curve in a third point. Then *R* is the reflection of this point in the *x*-axis. This is depicted in Figure (2.a). The elliptic curve in the figure consists of two parts, the ellipse-like figure and the infinite curve. If $P = (x_1, y_1)$, then the *double* of *P*, denoted $R = (x_3, y_3)$, is defined as follows. First draw the tangent line to the elliptic curve at *P*. This line intersects the elliptic curve in a second point. Then *R* is the reflection of this point in the *x*-axis. This is depicted in Figure (2.b).

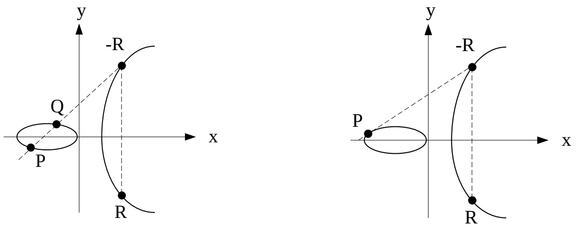


Fig (a). Addition.
(*R* = *P* + *Q*)

Fig (b). Doubling.
(*R* = *P* + *P*)

Figure (2). Geometric Description

The following algebraic formula[1] for the sum of two points and the double of a point can now be derived from the geometric description.

(a). $P + O = O + P = P$ for all $P \in E(GF(p))$.

(b). If $P = (x, y) \in E(GF(p))$, then $(x, y) + (x, -y) = O$. The point (*x*, -*y*) is denoted by (-*P*), and is called the *inverse* of *P*; observe that –*P* is indeed a point on the curve.

(c). (*Point addition*). Let $P = (x_1, y_1) \in E(GF(p))$ and $Q = (x_2, y_2) \in E(GF(p))$, where $P \neq \pm Q$. Then $P + Q = (x_3, y_3)$, where $x_3 = \lambda^2 - x_1 - x_2$ and $y_3 = \lambda(x_1 - x_3) - y_1$. In this case, $\lambda = (y_2 - y_1)/(x_2 - x_1)$.

(d). (*Point doubling*). Let $P = (x_1, y_1) \in E(GF(p))$, where $P \neq -P$. Then $2P = (x_3, y_3)$, where $x_3 = \lambda^2 - 2x_1$ and $y_3 = \lambda(x_1 - x_3) - y_1$. In this case, $\lambda = (3{x_1}^2 + a)/2y_1$.

Example (4). (*elliptic curve addition and doubling*) Let's consider the elliptic curve defined in Example (3).

a. *Addition*. Let $P = (1,\ 6)$ and $Q = (4, 6)$. Then $P + Q = (8, 7)$.
b. *Doubling*. Let $P = (1,\ 6)$. Then $2P = (10, 1)$.
c. *Inverse*. Let $P = (1,\ 6)$. Then $-P = (1, 7)$.

B. Elliptic Curves over Binary Field - GF(2^m)

A reduction polynomial $f(x)$ must be firstly chosen to construct a binary field GF(2^m). The elements generated by the reduction polynomial are applied to construct an elliptic curve E(GF(2^m)). The elliptic curve E(GF(2^m)) over binary field GF(2^m) is defined by the equation [1]:

$$y^2 + xy = x^3 + ax + b$$

where $a, b \in GF(2^m)$ and $b \neq 0$.

1) Points on E(GF(2^m))

The elliptic curve E(GF(2^m)) consists of a set of points:$\{P = (x, y) | y^2 + xy = x^3 + ax + b,\ x, y, a, b \in GF(2^m)\}$ together with a point at infinity. Every point on the curve has its *inverse*. The inverse of a point (*x*, *y*) on E(GF(2^m)) is $(x, x \oplus y)$. The number of points on the curve, including a point at infinity, is called its *order #E*. The pseudocode for finding the points on the elliptic curve E(GF(2^m)) is shown in Algorithm (2).

Algorithm (2). Pseudocode for finding the points on the elliptic curve E(GF(2^m))

```
Input: a, b, $f(x)$
Output: $P_i = (x_i, y_i)$
Begin
$x_i = \{0, 1, g^1, \dots, g^{m-2}\}$
$y_j = \{0, 1, g^1, \dots, g^{m-2}\}$

for(i=0; i<$2^m$; i++){
    for(j=0; j < $2^m$ ;j++){

    $w_1 = {x_i}^3 \oplus ax_i \oplus b$.
    $w_2 = {y_j}^2 \oplus x_i y_j$
    If($w_1 = w_2$) output $(x_i, y_j)$ $(x_i, y_j \oplus x_i)$
    }
}
End
```

Example (5). Let $f(x) = x^4 + x + 1$ be the reduction polynomial. Then 16 elements of GF(2^4) are shown in Table (1).

Table (1). Elements of GF(2^4)

0000	0	1000	x^3
0001	1	1001	x^3+1
0010	x	1010	x^3+x
0011	$x+1$	1011	x^3+x+1
0100	x^2	1100	x^3+x^2
0101	x^2+1	1101	x^3+x^2+1
0110	x^2+x	1110	x^3+x^2+x
0111	x^2+x+1	1111	x^3+x^2+x+1

Table (2) shows the power representation of g for elements of GF(2^4) generated by the polynomial $f(x) = x^4 + x + 1$. The element of $g = x = (0010)$ is a generator of GF(2^4) because its order is 15 $(2^4 - 1)$ as the following calculation show.

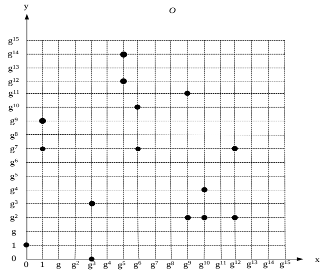
Table (2). Power representation of elements

g	0010	g^5	0110	g^9	1010	g^{13}	1101
g^2	0100	g^6	1100	g^{10}	0111	g^{14}	1001
g^3	1000	g^7	1011	g^{11}	1110	g^{15}	0001
g^4	0011	g^8	0101	g^{12}	1111		

Using the elliptic curve $E: y^2 + xy = x^3 + gx + 1$, with $a = g$ and $b = 1$, we can find the points on the curve, as shown in Figure (3). The *order* of the elliptic curve $E: y^2 + xy = x^3 + gx + 1$ over GF(2^4) is 16.

(0, 1)	O
$(1, g^7)$	$(1, g^9)$
$(g^3, 0)$	(g^3, g^3)
(g^5, g^{12})	(g^5, g^{14})
(g^6, g^7)	(g^6, g^{10})
(g^9, g^2)	(g^9, g^{11})
(g^{10}, g^2)	(g^{10}, g^4)
(g^{12}, g^2)	(g^{12}, g^7)

Points



Graph

Figure (3). Points on $E: y^2 + xy = x^3 + gx + 1$

2) Arithmetic Operations on E(GF(2^m))

As with elliptic curves over GF(2^m), there is a rule, called the *chord-and-tangent rule*, for adding two points on an elliptic curve E(GF(2^m)) to give a third elliptic curve point. Together with this addition operation, the set of points E(GF(2^m)) forms a group with O serving as its identity. The algebraic formula [1] for the sum of two points and the double of a point are the following.

(a). $P + O = O + P = P$ for all $P \in E(GF(2^m))$.

(b). If $P = (x, y) \in E(GF(2^m))$,
then $(x, y) + (x, x + y) = O$.
The point (x, x+y) is denoted by (-P), and
is called the *inverse* of P; observe that –P is indeed a point on the curve.

(c). (*Point addition*). Let $P = (x_1, y_1) \in E(GF(2^m))$ and $Q = (x_2, y_2) \in E(GF(2^m))$, where $P \neq \pm Q$. Then $P + Q = (x_3, y_3)$, where
$x_3 = \lambda^2 + \lambda + x_1 + x_2 + a$ and
$y_3 = \lambda(x_1 + x_3) + x_3 + y_1$.
In this case, $\lambda = (y_2 + y_1)/(x_2 + x_1)$.

(d). (*Point doubling*). Let $P = (x_1, y_1) \in E(GF(2^m))$, where $P \neq -P$. Then
$2P = (x_3, y_3)$, where $x_3 = \lambda^2 + \lambda + a$ and
$y_3 = {x_1}^2 + \lambda x_3 + x_3$. In this case, $\lambda = x_1 + (y_1/x_1)$.

Example (6). (*elliptic curve addition and doubling*) Let's consider the elliptic curve defined in Example (5).

a. *Addition*. Let $P = (g^5,\ g^{12})$ and $Q = (g^6, g^7)$. Then $P + Q = (g^{12}, g^7)$.

b. *Doubling*. Let $P = (g^5,\ g^{12})$.
Then $2P = (1, g^7)$.

c. *Inverse*. Let $P = (g^5,\ g^{12})$.
Then $-P = (g^5 + g^{14})$.

C. Elliptic Curve Discrete Logarithm Problem

The security of ECC depends on the difficulty of Elliptic Curve Discrete Logarithm Problem (ECDLP). Let P and Q be two points on an elliptic curve such that $kP = Q$, where k is a scalar. Given P and Q, it is computationally infeasible to obtain k, if k is sufficiently large. k is the discrete logarithm of Q to the base P. Hence the main operation involved in ECC is *point multiplication*. i.e. multiplication of a scalar k with any point P on the curve to obtain another point Q on the curve.

1) Point Multiplication

Scalar multiplication is the computation of the form $Q = k.P$ where P and Q are the elliptic curve points and k is an integer. This is achieved by repeated *point addition and doubling* operations. To calculate the above, integer k is represented as $k = k_{n-1}2^{n-1} + k_{n-2}2^{n-2} + \cdots + k_1 + k_0$ where $k_{n-1} = 1$ and $k_i \in \{0, 1\}, i = 0, 1, 2, \ldots, n-1$. This method is called *binary method* [2] which scans the bits of k either from left-to-right or right-to-left. The Algorithm-3 given below illustrates the computation of kP using binary method. It can be used for both elliptic curves over prime field GF(p) and binary field GF(2^m).

Algorithm (3). Binary Method

```
Input : Binary representation of k and point P
Output : Q = kP
Q=P
For i = n-1 to 0 do
Q = 2Q (Doubling)
If $k_i$ = 1 then
Q = Q + P (Addition)
Return Q
```

The cost of multiplication depends on the length of the *binary representation* of k and the number of 1s in this representation. The number of non-zero digits is called the *Hamming Weight* of scalar. In an average, binary method requires (n-1) *doublings* and (n-1)/2 *additions*. For each bit .1., we need to perform *point doubling* and *point addition*, if the bit is .0., we need only *point doubling* operation. So if we reduce the number of 1s in the scalar representation or hamming weight, the speed of elliptic curve scalar multiplication will improve.

2) Order of Points

Let $P \in E$(GF(p)). The *order* of P is the smallest positive integer, r, such that $rP = O$ where O is the group identity. Hasse's theorem [4] say that

$$p + 1 - 2\sqrt{p} \leq r \leq p + 1 + 2\sqrt{p}.$$

3) Attacks on ECDLP

The following algorithms can compute the elliptic curve discrete logarithm. Attacks on the ECDLP can be divided into three main categories:

(a). Algorithms that work in arbitrary groups, such as the exhaustive search and the Baby-Step Giant-Step algorithm,
(b). Algorithms that work in arbitrary groups with special conditions present in the group, like Pollard's rho method and Pollard's lambda method, and
(c). Algorithms that work only in specific groups, such as the Index Calculus and Pohlig-Hellman method.

The discrete logarithm problem is of fundamental importance to the area of public key cryptography. Many of the most commonly used cryptography systems are based on the assumption that the discrete logarithm is extremely difficult to compute; the more difficult it is, the more security it provides a data transfer. One way to increase the difficulty of the discrete logarithm problem is to base the cryptosystem on a larger group.

IV. IMPLEMENTATION

The elliptic curve cryptosystems on a small group are susceptible to the attacks described above. Therefore, we have to implement the elliptic curve cryptosystems under large integers to increase the difficulty of the discrete logarithm problem. *At first level*, we implement finite field arithmetic operations using java BigInteger class[6]. For prime fields, we implement a *PrimeField* class with methods of arithmetic operations for addition, subtraction, multiplication, multiplicative inverse and division of elements in the prime field *GF(p)*. And, for binary fields, we implement a *BinaryField* class with methods of arithmetic operations for addition, subtraction, multiplication, multiplicative inverse and division of elements in the binary field *GF(*2^m*)* with reduction polynomial *p*. *At second level*, we implement elliptic curve arithmetic operations by using *PrimeField* class and *BinaryField* class. The *ECCfp* class is implemented by using methods of *PrimeField* class for point addition and point doubling over prime field *GF(p)*. And the *ECCf2m* class is implemented by using methods of *BinaryField* class for point addition and point doubling over binary field *GF(*2^m*)*. *At third level*, we implement point multiplication for both *ECCfp* and *ECCf2m* by using algorithm (3). *At fourth level*, we will implement elliptic curve cryptosystems for our future research. For the implementation logic design of elliptic curve cryptosystems, the general hierarchy is shown in Figure (4).

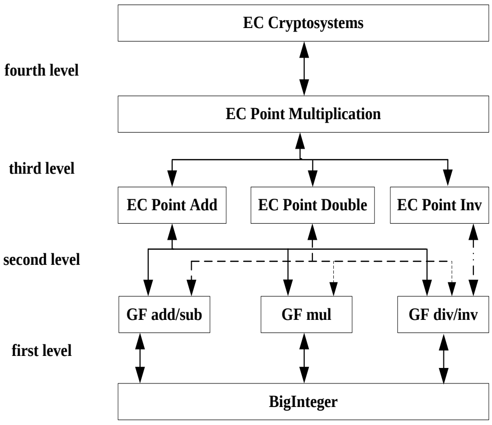


Figure (4). General logic design of implementation

We measure the performance of elliptic curve arithmetic basic operations: point addition and point doubling under prime field and binary field for comparison of execution time on the processor Intel Core i5@1.60GHz, 2.30GHz. The National Institute of Standards and Technology (NIST) submitted a report to recommend a set of elliptic curves for federal government use with larger key sizes [5]. We use NIST recommended elliptic curves for our research. The experimental results of elliptic curve arithmetic operations are shown in section (4.1). The performance results are listed in Table (3).

A. *Experimental Results*

Prime Field (P-192)
$E(GF(p)): y^3 = x^2 + ax + b$.
p=6277101735386680763835789423207666416083908700390324961279.
$a = -3$.
b=64210519e59c80e70fa7e9ab72243049feb8deecc146b9b1.
The order r = 627710173538668076383578942317605
901376719477318284228 4081.

Points on the curve
$P = (x_1, y_1)$.
x_1 = 188da80eb03090f67cbf20eb43a18800f4ff0afd82ff1012.
y_1 = 07192b95ffc8da78631011ed6b24cdd573f977a11e794811.

$Q = (x_2, y_2)$.
x_2 = 5701b8be342fb767752f13a308e2eff016b41fd348ef1ea.
y_2 = 77aeacae8fd493a524b9b18509c9a60e7e2a7da86882d82c.

Point Addition
$R = P + Q = (x_3, y_3)$.
x_3 = c5675f8265cf98e933db304666558478ca70c5ebba4da630.
y_3 = 2c2560e527695bbe883084abf6736e0a7e06b489ba57cb39.

Point Doubling
$2P = (x_4, y_4)$.
x_4 = dafebf5828783f2ad35534631588a3f629a70fb16982a888.
y_4 = dd6bda0d993da0fa46b27bbc141b868f59331afa5c7e93ab.
Point Multiplication
$r.P = O$.

Binary Field (K-163)
$E\left(GF(2^m)\right): y^2 + xy = x^3 + ax^2 + b$.
$p(t) = t^{163} + t^7 + t^6 + t^3 + 1$.
$a = 1$.
$b = 1$.
The order r = 5846006549323611672814741753598448348329118574063.

Points on the curve.
$P = (x_1, y_1)$.
x_1 = 2fe13c0537bbc11ac aa07d793de4e6d5e5c94eee8.
y_1 = 289070fb05d38ff58321f2e800536d538ccdaa3d9.

$Q = (x_2, y_2)$.
x_2 = 4ec251aba46dc1dd4d6a0d18d25f98deb6e0cba68.
y_2 = f35a742feaeb3ec71b751605457513c6f20a82b.

Point Addition
$R = P + Q = (x_3, y_3)$.
x_3 = fd5f739b49278be0cc385ab2a0f66a695775312e.
y_3 = 44dfc418c0590df2951cbc876ba30dbffd4b945b1.

Point Doubling
$2P = (x_4, y_4)$.
x_4 = cb5ca2738fe300aacfb00b42a77b828d8a5c41eb.
y_4 = 229c79e9ab85f90acd3d5fa3a696664515efefa6b.
Point Multiplication
$r.P = O$.

Table (3). The results of performance

EC Arithmetic Operations	Prime Field (ms/100000times)	Binary Field (ms/100000times)
Addition	3770	87548
Doubling	3521	85244

V. CONCLUSION

The performance of elliptic curve arithmetic basic operations, point addition and point doubling, under prime field and binary field, depends on the performance of equivalent finite field arithmetic operations. As a result of the performance for finite field arithmetic operations in the paper [6], it proved that the java BigInteger class is more efficient for the software implementation of finite field arithmetic operations in prime field than in binary field. Therefore, the results of performance in the table (1) more proved that the java BigInteger class is more efficient for the software implementation of elliptic curve arithmetic operations in prime field than in binary field.

REFERENCES

[1]. Behrouz A. Forouzan, Cryptography and Network Security, McGraw-Hill press, International Edition, 2008.

[2]. Darrel Hankerson, Alfred Menezes, Scott Vanstone. Guide to Elliptic Curve Cryptography, Springer press, 2004.

[3]. Lawrence C. Washington, Elliptic Curves: Number Theory and Cryptography, Second Edition, Taylor & Francis Group, LLC, 2008.

[4]. Rudolf Lidl and Harald Niederreiter, Introduction to Finite Field Arithmetic and their Applications, Cambridge University Press, 1986.

[5]. Recommended Elliptic Curves for Federal Government Use, NIST, 1999.

[6]. Ni Ni Hla, Tun Myat Aung. Implementation of Finite Field Arithmetic Operations for Large Prime and Binary Fields Using java BigInteger class, International Journal of Engineering Research and Technology (IJERT), Volume. 6, Issue. 08, August - 2017.