Implementation of Finite Field Arithmetic Operations for Large Prime and Binary Fields using java BigInteger Class

Ni Ni Hla
University of Computer Studies, Yangon
Myanmar

Tun Myat Aung
University of Computer Studies, Yangon
Myanmar

***Abstract*—Many cryptographic protocols are based on the difficulty of factoring large composite integers or a related problem. Therefore, we implement the finite field arithmetic operations for large prime and binary fields by using java BigInteger class to study our research under large integers for public key cryptosystems and elliptic curve.**

Keywords — Finite Field Arithmetic; Prime Field; Binary Field, Large Integer

I. INTRODUCTION

The origins and history of finite fields can be traced back to the 17th and 18th centuries, but there, these fields played only a minor role in the mathematics of the day. In more recent times, however, finite fields have assumed a much more fundamental role and in fact are of rapidly increasing importance because of practical applications in a wide variety of areas such as coding theory, cryptography, algebraic geometry and number theory.

Nowadays, a finite field is very important structure in cryptography. Many cryptographic applications use finite field arithmetic. Public key systems based on various discrete logarithm problems are frequently implemented over finite fields to provide structure and efficient arithmetic.

The finite field arithmetic operations need to be implemented for the development and research of stream ciphers, block ciphers, public key cryptosystems and cryptographic schemes over elliptic curves. Many cryptographic protocols are based on the difficulty of factoring large composite integers or a related problem. Therefore, we implement the finite field arithmetic operations for large prime and binary fields by using java *BigInteger* class to study our research under large integers.

The organization of this paper is as follows: section 2 is devoted to finite fields and their properties. In section 3, how to implement finite field arithmetic operations under prime field and binary field are described. Some algorithms applied in the implementation are listed in section 4. The results of implementation for finite field arithmetic operations under prime field and binary field are shown in section 5. Finally, we conclude our discussion in section 6.

II. INTRODUCTION TO FINITE FIELDS

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: $(a+b) \cdot c = (a \cdot c) + (b \cdot c)$ 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 $a,b \in F$ 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 $\{0,1,2, \ldots, p-1\}$ 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, $a \bmod p$ 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*.

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).

(i) Addition: 17+20 = 8 since 37 mod 29 = 8.
(ii) Subtraction: 17−20 = 26 since −3 mod 29 = 26.
(iii) Multiplication: 17 · 20 = 21 since 340 mod 29 = 21.
(iv) 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)*.

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 $f(x) = x^4 + x + 1$.

(i). Addition: $(x^3+x^2+1) + (x^2+x+1) = x^3 + x$
(ii). Subtraction: $(x^3+x^2+1) - (x^2+x+1) = x^3 + x$
(iii). 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 $(x^5+x+1)\ mod\ (x^4+x+1) = x^2 + 1$.
(iv). Inversion: $(x^3+x^2+1)^{-1} = x^2$ since $(x^3+x^2+1).x^2\ mod\ (x^4+x+1) = 1$.

III. IMPLEMENTATION OF FIELD OPERATIONS

The finite field arithmetic operations: addition, subtraction, division, multiplication and multiplicative inverse, need to be implemented for the development and research of stream ciphers, public key cryptosystems and cryptographic schemes over elliptic curves. We implement the finite field arithmetic operations by using java *BigInteger* class to study our research under large numbers.

A. Arithmetic Operations of Prime Field

The arithmetic operations of *prime field* need to be implemented to study our research under prime fields. Therefore, we implement a *PrimeField* class with methods of arithmetic operations for addition, subtraction, multiplication and division of elements *(a, b)* in the prime field *GF(p)*. The methods of *PrimeField* class are implemented as follows.

(i). The *addition* method is implemented by *add* and *mod* methods of *BigInteger* class for the logic statement: *a + b = (a + b) mod p*.
(ii). The *subtraction* method is implemented by *add*, *subtract*, and *mod* methods of *BigInteger* class for the logic statement: *a - b = (a + (- b)) mod p*. In this case, *-b* is an additive inverse of prime number *p*. The logic statement of additive inverse *-b* is *(p – b)*.
(iii). The *multiplication* method is implemented by *multiply* and *mod* methods of *BigInteger* class for the logic statement: $a\,.\,b = (a \times b)\ mod\ p$.
(iv). The *division* method is implemented by *multiply* and *modInverse* methods of *BigInteger* class for the logic statement: $a \div b = (a \times b^{-1})\ mod\ p$. In this case, b^{-1} is a multiplicative inverse of prime number *p*.
(v). The *multiplicative inverse* method is adopted from the *modInverse* method.

B. Arithmetic Operations of Binary Field

The arithmetic operations of *binary field* need to be implemented to study our research under prime fields. Therefore, we implement a *BinaryField* class with methods of arithmetic operations for addition, subtraction, multiplication and division of elements *(a, b)* in the binary field *GF(*2^m*)* with reduction polynomial *p*. The methods of *BinaryField* class are implemented as follows.

(i). The *addition* method is implemented by *xor* method of *BigInteger* class for the logic statement:$a + b = a \oplus b$. In this case, The *addition* operation is implemented by bitwise XOR operation of all bits of the two operands.
(ii). The *subtraction* method is identical to the *addition* method as above.
(iii). The *multiplication* method is implemented by *shifLeft* and *xor* methods of *BigInteger* class for the logic statement: $a\,.\,b = (a \times b)\ mod\ p$. The algorithm for multiplication of two polynomials in *GF(*2^m*)* is given in Algorithm (1)[1].
(iv). The *quotientAndRemainder* method is implemented by *shifLeft* and *setBit* methods of *BigInteger* class for the logic statement:$(q, r) = (a \div b)$. The algorithm to find quotient (q) and remainder (r) from division of two polynomials in *GF(*2^m*)* is given in Algorithm (2).
(v). The *multiplicativeInverse* method is implemented by *quotientAndRemainder* and *multiplication* methods of *BinaryField* class and *xor* method of *BigInteger* for the logic statement: $b \cdot b^{-1}\ mod\ p = 1$. The multiplicative inverse b^{-1} is computed by using Extended Euclidean GCD algorithm given in Algorithm (3)[2].
(vi). The *division* operation is implemented by *multiplication* and *multiplicativeInverse* methods of *BinaryField* class for the logic statement: $a \div b = (a \times b^{-1})\ mod\ p$. In this case, b^{-1} is a multiplicative inverse of prime

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 6 Issue 08, August - 2017

polynomial p. The multiplicative inverse is adopted from the *multiplicativeInverse* method.

IV. ALGORITHMS

Algorithm (1). *shift-and-xor method*
Input: a, b, p as polynomials
Output: result
Begin
Set result= 0;
For(i=0; i<bitLength of b; i++)
begin
  If(b_i == 1)
    Set result = result xor a.
  endIf
  Set a = shiftLeft(1) of a.
  If(a_{LSB} == 1)
    Set a = a xor p.
  endIf
end
Return Result
End

Algorithm (2). *shift-and-setBit method*
Input: a, b as polynomials
Output: quotient, remainder
Begin
Set q = 0.
for (term = bitLength of a – bitLength of b; term >= 0; term--)
begin
if (bitLength of a == bitLength of b + term)
Set a = a xor shiftLeft(term of b).
Set quotient = setBit(term of quotient).
endIf
end
Set remainder = a.
Return quotient, remainder
End

Algorithm (3). *Extended Euclidean GCD algorithm*

Input: x, p as polynomials
Output: a
Begin
    Set y = x.
    Set x = p.
Set a = 0.
    Set b = 1.
    while $(y \neq 0)$
    begin
        Set $q = x / y$.
        Set $r = x \bmod y$.
        Set x = y.
        Set y = r.
        Set $temp = a \oplus (q \times b)$.
        Set a = b.
        Set b = temp;
    end
    if (x = 1) return a.
    endIf
End

V. RESULTS OF IMPLEMENTATION

We measure the performance of finite field arithmetic operations: addition, subtraction, division, multiplication and multiplicative inverse, under prime field and binary field for comparison of execution time on the processor Intel Core i5@1.60GHz, 2.30GHz. The finite field arithmetic operations use the large integers of the prime field and the binary field defined by NIST recommended elliptic curve for federal government [6]. The results are listed in Table (1).

Prime Field (P-192)
P= 62771017353866807638357894232076664160839087003903 24961279.
X = 188da80eb03090f67cbf20eb43a18800f4ff0afd82ff1012.
Y = 07192b95ffc8da78631011ed6b24cdd573f977a11e794811.
Addition
Z = X + Y.
Z = 776096614669310688163071032867745522280722465564271335459.
Subtraction
Z = X - Y.
Z = 427995950082066625353355928307306701552675487709498034177.
Multiplication
Z = X . Y.
Z = 463980704477630344363893383854114350541460842267 8862314472.
Division
Z = X % Y.
Z = 102023148406326899839785129772657290128628412720 7709149774.
Multiplicative Inverse of X
Z = 450148766166845920113120162576033894528685541159 2992703750.

Binary Field (K-163)
$p(t) = t^{163} + t^7 + t^6 + t^3 + 1$
X = 2fe13c0537bbc11ac aa07d793de4e6d5e5c94eee8
Y = 289070fb05d38ff58321f2e800536d538ccdaa3d9
Addition
Z = X + Y.
Z = 7714cfe32684eef49818f913db78b866904e4d31
Subtraction
Z = X - Y.
Z = 7714cfe32684eef49818f913db78b866904e4d31
Multiplication
Z = X . Y.
Z = 4d741872162b253d5a381f1f680b47e5c0ad3aa2a
Division
Z = X % Y.
Z = 498d03bb544d83614e0b5963052f604eb8ec8d0cd
Multiplicative Inverse of X
Z = 63f514f39f4587684f96c8dd6558e69339a1efed9

Table (1). The results of performance

Finite Field Arithmetic Operations	Prime Field (ms/100000times)	Binary Field (ms/100000times)
Addition	31	16
Subtraction	62	16
Division	2262	70497
Multiplication	156	2808
Multiplicative inverse	2028	70153

VI. CONCLUSION

This is the first step to study our research under large integers for public key cryptosystems and elliptic curve. The performance of addition and subtraction operations of binary field are more efficient than prime field. The performance of division, multiplication and multiplicative inverse operations of prime field are more efficient than binary field. Therefore, a java BigInteger class is more efficient for the software implementation of finite field arithmetic operations in prime field.

REFERENCES

[1]. Annabell Kuldmaa, Efficient Multiplication in Binary Fields, 2015.
[2]. Behrouz A. Forouzan, Cryptography and Network Security, McGraw-Hill press, International Edition, 2008.
[3]. Darrel Hankerson, Alfred Menezes, Scott Vanstone. Guide to Elliptic Curve Cryptography, Springer press, 2004.
[4]. Dave K. Kythe, Prem K. Kythe. Algebraic and Stochastic Coding Theory, CRC Press, 2012.
[5]. Rudolf Lidl and Harald Niederreiter, Introduction to Finite Field Arithmetic and their Applications, Cambridge University Press, 1986.
[6]. Recommended Elliptic Curves for Federal Government Use, NIST, 1999.