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) · c = (a · c) + (b · c) for all a, b, c \in 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 \( b \neq 0 \), \( a/b = a \cdot b^{-1} \) 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 \( a \), \( a \mod p \) 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 \( p \).
Example 1. (prime field GF(29)) The elements of GF(29) are \{0,1,2, \ldots ,28\}. The following are some examples of arithmetic operations in GF(29).

(i) Addition: 17+20 = 8 since 37 mod 29 = 8.
(iii) Multiplication: 17 \cdot 20 = 21 since 340 mod 29 = 21.
(iv) Inversion: 17 ^{−1} = 12 since 17 \cdot 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:

\[
\begin{array}{ccc}
0 & x^2 & x^3 \\
1 & x^2 + 1 & x^3 + x \\
x & x^2 + x & x^3 + x^2 + 1 \\
x + 1 & x^2 + x + 1 & x^3 + x + 1
\end{array}
\]

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) \cdot (x^2 + x + 1) = x^2 + 1.
\]

since \((x^3 + x^2 + 1), (x^2 + x + 1) = x^5 + x + 1 and \(\mod x^4 + x + 1 = x^2 + 1.
(iv). Inversion: \((x^3 + x^2 + 1)^{−1} = x^2\) since \((x^3 + x^2 + 1) \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: \(a \cdot b = (a \times b) \mod p\).

(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: \(a \cdot b = (a \times b) \mod p\) 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 \((r)\) of 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.
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( \( i=0; i<\text{bitLength of } b; i++ \) ) 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(\( \text{bitLength of } a == \text{bitLength of } b + \text{term} \)) 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 = \( x \mod y \).

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.

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).

### Table (1). The results of performance

<table>
<thead>
<tr>
<th>Arithmetic Operations</th>
<th>Prime Field (ms/100000times)</th>
<th>Binary Field (ms/100000times)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Addition</td>
<td>31</td>
<td>31</td>
</tr>
<tr>
<td>Subtraction</td>
<td>62</td>
<td>62</td>
</tr>
<tr>
<td>Division</td>
<td>2262</td>
<td>2262</td>
</tr>
<tr>
<td>Multiplication</td>
<td>156</td>
<td>156</td>
</tr>
<tr>
<td>Multiplicative inverse</td>
<td>2028</td>
<td>2028</td>
</tr>
<tr>
<td>Multiplication</td>
<td>70153</td>
<td>70153</td>
</tr>
<tr>
<td>Subtraction</td>
<td>70493</td>
<td>70493</td>
</tr>
<tr>
<td>Division</td>
<td>156</td>
<td>156</td>
</tr>
<tr>
<td>Multiplication</td>
<td>70153</td>
<td>70153</td>
</tr>
<tr>
<td>Multiplicative inverse</td>
<td>2028</td>
<td>2028</td>
</tr>
</tbody>
</table>
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