Speeding up Elliptic Curve Arithmetic over Finite Field

2001027 Kim Jong-seong
2001093 Lee Jung-yeun

abstract

In recent years, there has been increasing interest in the use of elliptic curve groups for public key cryptosystems such as digital signature and key exchange. one of primary concerns in this area is to speed up elliptic curve arithmetic in software. An important question is how fast this exponentiation can be done, which often determines whether a given system is practical. The best method for exponentiation depends on the group being used, the hardware the system is implemented on. In this paper we survey the known methods for fast exponentiation

1. Introduction

Elliptic curve cryptosystems, methods for building public key cryptosystems in the group of points on an elliptic curve over a finite field, were first introduced by koblitz and then investigated by many researchers. In particular, much research has been conducted on fast algorithms and implementation techniques of elliptic curve arithmetic over various finite fields.

Since elliptic curve groups can provide a higher level of security with smaller key size, there is increasing interest also in the industry and thus a lot of active standardization processes are going on. Thus we can expect that elliptic curve cryptosystem will be widely used for many security applications in the near future. In this regard, it is also expected that there will be a strong demand on efficient algorithms for fast exponentiation of elliptic curve cryptosystems. An elliptic curve over $GF(p^n)$ is particularly attractive, since we can achieve a very good performance with a suitable choice of parameters. The elliptic curve variant of KCDSA already noticed the usefulness of such fields and focused on implementations using elliptic curve over $GF(p^n)$.

This paper deals with efficient algorithms and implementation techniques for fast

arithmathmatic in bith the underlying field and elliptic curve group. For fast group operations, we present efficient formulas for point addition and doubling for elliptic curves over $GF(p^n)$. To speed up field arithmatic, we provide efficient algorithms for field multiplication and inversion.

This paper is organized as follows. we briefly introduce the elliptic curve over finite field and explain the OEF method, ONB method and Windows method

2. Elliptic curves over finite fields

We recall well known properties of elliptic curves. All these can be found in [6].

The general equation of an elliptic curve E is given as:

$$F(X, Y, Z): -Y^2Z + a_1XYZ + a_3YZ^2 - (X^3 + a_2X^2Z + a_4XZ^2 + a_6Z^3) = 0$$

where the a_i's are in K and the discriminant Δ defined by

$$d_2 = a_1^2 + 4a_2, \; d_4 = 2a_4 + a_1a_3, \; d_6 = a_3^2 + 4a_6, \; d_8 = a_1^2a_6 + 4a_2a_6 - a_1a_3a_4 + a_2a_3^2 - a_4^2,$$

$$c_4 = d_2^2 - 24d_4, \; \Delta = -d_2^2d_8 - 8d_4^3 - 27d_6^2 + 9d_2d_4d_6$$

is invertible in K. The j-invariant of the curve is $j(E) = c_4^3/\Delta$.

It is possible to define on the set of points $E(K)$ of E

$$E(K) = \{(x, y) \in K^2, F(x, y, 1) = 0\} \cup \{O_E\}$$

an Abelian law using the so-called tangent-and-chord method, O_E being the neutral element (0, 1, 0).

3. Optimal Extension Field Arithmetic

3.1 Approach

The OEF approach is based on the observation that several well-known optimizations exist for software implementation of finite field arithmetic and that

when they are used in conjunction they yield significant performance gains for implementation of elliptic and hyper-elliptic curve cryptosystems. To optimize arithmetic in GF(p^m) They stipulate the following properties on the choice of p and m:

1. choose p to be less than but close to the word size of the processor so that all subfield operations take advantage of the processor's fast integer arithmetic.

2. Choose p to be a pseudo-Mersenne prime, that is, of the form $2^n \pm c$ for some $\log_2 c \leq \frac{1}{2} n$ to allow for efficient subfield modular reduction.

3. Choose m so that we have an irreducible binomial $x^m - \omega$ for efficient extension field modular reduction. The extension degree m can be small if the processor word size allows for large values of p.

3.2 Previous Work

Previous work on optimization of software implementations of finite field arithmetic has often focused on a single cryptographic application, such as designing a fast implementation for one particular finite field. One popular optimization involves the use of subfields of characteristic two. A paper due to DeWin etal. [17] analyzes the use of $GF((2^n)^m)$, with a focus on $n = 16$, $m = 11$. This construction yields an extension field with 2^{176} elements. The subfield $GF(2^{16})$ has a Cayley table of sufficiently small size to fit in the memory of a workstation. Optimizations for multiplication and inversion in such composite fields of characteristic two are described in [3].

Schroeppel et al. [16] report an implementation of an elliptic curve analogue of Diffie-Hellman key exchange over $GF(2^{155})$ with an irreducible trinomial as the field polynomial. The arithmetic is based on a polynomial basis representation of the field elements. Elements of the field are each stored in three 64-bit registers.

Much optimization work has been done in selection of Optimal Normal Bases(ONB) to speed computations in $GF(2^m)$. Draft standards such as [18] [19], and [9] suggest use of ONB for elliptic curve systems.

Others have investigated use of pseudo-Mersenne primes to construct Galois fields $GF(p)$ in connection with elliptic curve cryptography as found in [12], [14] and some patents have been issued on their use.

Unlike the methods in [17, 3] which use Cayley tables to implement subfield arithmetic, our approach requires no additional memory and is therefore attractive in memory-constrained applications. In addition, our system is faster in real-world tests as described in Section 8.

3.3 Optimal Extension Field Arithmetic

This section describes the basic construction for arithmetic in fields $GH(p^m)$, of which an OEF is a special case. The subfield is $GF(p)$ and the extension degree is denoted by m, so that the field can be denoted by $GF(p^m)$. This field is isomorphic to $GF(p)[x]/(P(x))$, where $P(x) = x^m + \sum_{i=0}^{m-1} b_i x^i$, $b_i \in GF(p)$, is a monic irreducible polynomial of degree m over $GF(p)$. In the following, a residue class will be identified with the polynomial of least degree in this class. We consider a standard (or polynomial or canonical) basis representation of a field element $A \in GF(p^m)$:

$$A(x) = a_{m-1}x^{m-1} + \cdots + a_1 x + a_0, \qquad (1)$$

where $a_i \in GF(p)$. Since we choose p to be less than the processor's word size, we can represent $A(x)$ with m registers.

All arithmetic operations are performed modulo the field polynomial. The choice of field polynomial determines the complexity of the operations required to perform the modular reduction. In this paper, we will only be concerned with the operations of addition, multiplication, and squaring.

3.3.1 Addition and Subtraction

Addition and subtraction of two field elements is implemented in a straightforward manner by adding or subtracting the coefficients of their polynomial representation and if necessary, performing a modular reduction by subtracting p once from the intermediate result. Previous implementations in $GF(2^m)$ offer a slight computational advantage since addition or subtraction is

simply an XOR that does not require modular reduction. When compared to the addition operation in $GF(p)$ for large p, we observe that an OEF does not require carry between computer words in computing a sum while $GF(p)$ does. This property results in a modest performance gain over $GF(p)$.

3.3.2 Multiplication

Multiplication is performed in two stages. First, we perform an ordinary polynomial multiplication of two field elements $A(x)$ and $B(x)$, resulting in an intermediate product $C'(x)$ of degree less than or equal to $2m-2$:

$$C'(x) = A(x) \times B(x) = c'_{2m-2}x^{2m-2} + \cdots + c'_1 x + c'_0; \quad c'_i \in GF(p). \tag{2}$$

The schoolbook method to calculate the coefficients c'_i, $i = 0, 1, \ldots, 2m-2$, requires m^2 multiplications and $(m-1)^2$ additions in the subfield $GF(p)$.

Since field multiplication is the time critical task in many public-key algorithms this paper will deal extensively with fast multiplication methods, and later sections are devoted to aspects of this operation. In Section 3.3 we present an efficient method to calculate the residue $C(x) \equiv C'(x) \bmod P(x)$, $C(x) \in GF(p^m)$. Section 3.4 gives a method to quickly perform the coefficient multiplication in $GF(p)$.

3.3.3 Squaring

Squaring may be implemented using the method for general multiplication outlined above. However, we observe that squaring a field element affords some additional computational efficiencies. For example, consider the field element $A(x) = a_2x^2 + a_1x + a_0$, $A(x) \in GF(p^3)$. We compute the square of $A(x)$ and obtain:

$$(a_2x^2 + a_1x + a_0)^2 = a_2^2x^4 + 2a_2a_1x^3 + (2a_2a_0 + a_1^2)x^2 + 2a_1a_0x + a_0^2 \tag{3}$$

Multiplication by two may be implemented in a computer as a left shift operation by one bit. On many computer architectures, a left shift is faster than an explicit integer multiplication. Thus instead of requiring m^2 multiplications, we need only $m(m+1)/2$ explicit multiplications. The

remainder may be performed as shifts.

3.4 Extension Field Modular Reduction

After performing a multiplication of field elements in a polynomial representation, we obtain the intermediate result $C'(x)$. In general the degree of $C'(x)$ will be greater than or equal to m. In this case, we need to perform a modular reduction. The canonical method to carry out this calculation is long polynomial division with remainder by the field polynomial. We observe that we must perform subfield multiplications to implement the reduction, proportional to the number of terms in the field polynomial. However, if we construct a field polynomial with low coefficient weight, the modular reduction will require fewer subfield multiplications.

Since monomials a^m, $m > 1$ are obviously always reducible, we turn our attention to irreducible binomials. An OEF has by definition a field polynomial of the form:

$$P(x) = x^m - \omega \tag{4}$$

The use of irreducible binomials as field polynomials yields major computational advantages as will be shown below. Observe that irreducible binomials do not exist over $GF(2)$.

In section 3.5, we will demonstrate that such irreducible binomials can be constructed. Once such a binomial has been determined, modular reduction can be performed with the following complexity:

Theorem 1. Given a polynomial $C'(x)$ over $GF(p)$ of degree less than or equal to $2m-2$, $C'(x)$ can be reduced modulo $P(x) = x^m - \omega$ requiring $m-1$ multiplications by ω and $m-1$ additions, where both of these operations are performed in $GF(p)$.

Proof. By assumption, $C'(x)$ has the form:

$$C'(x) = c'_{2m-2}x^{2m-2} + \cdots + c'_m x^m + c'_{m-1}x^{m-1} + \cdots + c'_1 x + c'_0 \tag{5}$$

Only the terms $c'_{m+i}x^{m+i}$, $i \geq 0$, must be reduced modulo $P(x)$. We observe

that:

$$c'_{m+i}x^{m+i} \equiv \omega c'_{m+i}x^i \bmod P(x),\ i = 0, 1, \ldots, m-2 \quad (6)$$

Since the degree of $C'(x) \leq 2m-2$, we require at most $m-1$ multiplications by ω and $m-1$ additions to combine the reduced terms.

A general expression for the reduced polynomial is given by:

$$C(x) \equiv c'_{m-1}x^{m-1} + \omega c'_{2m-2}x^{m-2} + \ldots + \omega c'_{m+1}x + \omega c'_m \bmod P(x) \quad (7)$$

As an optimization, when possible we choose those fields with an irreducible binomial $x^m - 2$, allowing us implement the multiplications as shifts.

3.5. Fast Subfield Multiplication

As shown above, fast subfield multiplication is essential for fast multiplication in $GF(p^m)$. Subfield arithmetic in $GF(p)$ is implemented with standard modular integer techniques, which are previously reported in the literature, see for example [12]. For actual implementation of OEF arithmetic, optimization of subfield arithmetic is critical to performance, so we include these remarks in this paper for completeness.

We recall that multiplication of two elements $a, b \in GF(p)$ is performed by $a \times b \equiv c \bmod p$. Modern workstation CPUs are optimized to perform integer arithmetic on operands of size up to the width of their registers. An OEF takes advantage of this fact by constructing subfields whose elements may be represented by integers in a single register. For example, on a workstation with 64-bit registers, the largest prime we may represent is $2^{64} - 59$. So we choose a prime may be performed with 2 shifts and 2 adds if the intermediate result is contained in a single word, a substantial improvement over the $c > 1$ case. An OEF that offers this optimization is known as Type I. In our implementation as reported in Section 8, we have included $p \equiv 2^{31} - 1$ for this reason. Our implementation takes advantage of its special form, making $p = 2^{31} - 1$ the best performing choice of p we consider.

3.6. Irreducible Binomials

In Section 3.4 we showed that irreducible binomials allow modular reduction with low complexity. The following theorem from [11] describes the cases when an irreducible binomial exists:

Theorem 2. Let $m \geq 2$ be an integer and $\omega \in GF(p)$. Then the binomial $x^m - \omega$ is irreducible in $GF(p)$ if and only if the following two conditions are satisfied:

(i) each prime factor of m divides the order e of ω in $GF(p)$, but not (p -1)/e;

(ii) $p \equiv 1 \bmod 4$ if $m \equiv 0 \bmod 4$.

An important corollary is given in [5]:

Corollary 1. Let ω be a primitive element for $GF(p)$ and let m be a divisor of p - 1. Then $x^m - \omega$ is an irreducible polynomial of order(p -1) m over $GF(p)$.

We present the following new corollary which follows directly from the above, since $p-1$ is always an even number:

Corollary 2. Let ω be a primitive element for $GF(p)$. then $x^2 - \omega$ is irreducible over $GF(p)$

3.7. Optimal Extension Fields

In the following, we define a new class of finite field, which we call an Optimal Extension Field (OEF). To simplify matters, we introduce a new name for a class of prime numbers:

Definition 1. A pseudo-Mersenne prime is a prime number of the form $2^n + c$, $\log_2 c \leq \frac{1}{2} n$. $p \leq 2^{64} - 59$ as the field characteristic on this computer. To this end, we recommend the use of Galois fields with subfields as large as possible while still within single-precision limits of our host CPU.

It is well known that fast modular reduction is possible with modulo of the form $2^n \pm c$, where c is a "small" integer. Integers of this form allow modular reduction without division. We present a form of such a modular reduction algorithm, adapted from [12]. In this paper we consider only primes of the form $2^n - c$, although a trivial change to the following algorithm allows the use of primes $2^n + c$. The operators << and >> are taken to mean "left shift" and "right shift" respectively.

Algorithm 1 Fast subfield Modular Reduction

Require: $p = 2^n - c$, $\log_2 c \leq \frac{n}{2}$, $x < b^2$ is the integer to reduce

Ensure: $r \equiv x \bmod p$

$q_0 \leftarrow x >> n$

$r_0 \leftarrow x - q_0 2^n$

$r \leftarrow r - q_0$

$i \leftarrow 1 + r_1$

while $q_i \geq 0$ do

$q_{i+1} \leftarrow q_i c >> n$

$r_{i+1} \leftarrow q_i c - (q_{i+1} << n)$

$i \leftarrow i + 1$

$r \leftarrow r + r_i$

end while

while $r \geq p$ do

$r \leftarrow r - p$

end while

Definition 2. An Optimal Extension Field is a finite field $GF(p^m)$ such that:

1. p is a pseudo-Mersenne prime,
2. An irreducible binomial $P(x) = x^m - \omega$ over $GF(p)$.

We observe that there are two special cases of OEF which yield additional arithmetic advantages, which we call Type I and Type II.

Definition 3. A Type I OEF has $p = 2^n \pm 1$.

A Type I OEF allows for subfield modular reduction with very low complexity, as described in Section 5.

Definition 4. A Type II OEF has an irreducible binomial $x^m - 2$.

A Type II OEF allows for speedups in extension field modular reduction since the multiplications by ω in Theorem 1 can be implemented using shifts instead of explicit multiplications.

The choice of m depends on the factorization of $p - 1$ due to Theorem 2 and Corollary 1. In the following we describe an efficient construction method for OEFs. From a very high level, this method consists of three main steps: We choose a pseudo-Mersenne prime p first, then factor $p - 1$, and then finally select an extension degree m. Since $p \leq 2^{64}$ due to current common processor word lengths, it is sufficient to use trial division to quickly factor $p - 1$. This procedure does not exhaustively list all OEFs, rather it is designed to quickly locate a Type II OEF for a desired field order and machine word size. Further, this procedure considers only those primes $2^n - c$, although a prime $2^n \pm c$ is a valid choice for OEFs.

4. Normal Bases

Some groups have added structure that allow much faster exponentiation. In $GF(p^n)$, normal bases allow pth powers to be calculated with just a cyclic shift, greatly speeding the p-ary method.

The most common use of this is in $GF(2^n)$, where the use of a normal basis allows squarings to be done with just a shift. The 2^k-ary method then takes only $\lfloor n/k \rfloor \pm 2^{k-1} - 2$ multiplications, since only odd powers up to $2^k - 1$ need to be computed.

Compute $g^3, g^5, \ldots, g^{2^k-1}$.
$a \leftarrow 1$
for $d = 2^k - 1$ to 1 by -2
  for each i such that $c_i = d$
    $a \leftarrow a * (g^d)^{2^{k_i}}$

return a.

5. Window method

5.1. Binary Method

This method is also known as the "square and multiply" method. It is over 2000 years old; Knuth[13] discusses its history and gives references. The basic idea is to compute g^r using the binary expansion of r. Let

$$r = \sum_{i=0}^{l} c_i 2^i.$$

Then the following algorithm will compute g^r:

$a \leftarrow 1$
for $d = l$ to 0 by -1
  $a \leftarrow a * a$
  if $c_d = 1$ then $a \leftarrow a * g$
return a.

5.2. m-ary method

The above meyhod has an obvious generalization : use a base lager than two. Let

$$r = \sum_{i=0}^{l} c_i m^i.$$

The m-ary method computes g^r using this representation:

Compute $g^2, g^3, \ldots, g^{m-1}$.
$a \leftarrow 1$
for $d = l$ to 0 by -1
  $a \leftarrow a^m$
  $a \leftarrow a * g^{c_d}$
return a.

5.3. Window Method

The 2^k-ary method may be thought of as taking k-bit *windows* in the binary representation of r, calculating the powers in the windows one by one, squaring them k times to shift them over, and then multiplying by the power in

the next window.

This leads to several different generalizations. One obvious one is that there is no reason to force the windows to be next to each other. Strings of zeros do not need to be calculated, and may be skipped. Moreover, only odd powers of g need to be computed in the first step.

References

1. Daniel V. Bailey. Optimal extension fields. Major Qualifying Project(Senior thesis), 1998. Computer Science Department, Worcester Polytechnic Institute, Worcester, MA, USA.

2. Richard E. Crandall. Method and apparatus for public key exchange in cryptographic system. US Patent 5463690, 1995.

3. Jorge Guajardo and Christof paar. Efficient algorithms for elliptic curve cryptosystems. In Advances in Cryptology - Ctypto '97, pages 342-356. Springer Lecture Notes in Computer Science, August 1997.

4. G. harper, A. Menezes, and S. Vanstone. Public-key cryptosystems with very small key lengths. In Advances in Cryptology - EUROCRYPT '92, pages 163-173, May 1992.

5. D. Jungnickel. Finite Fields. B.I.-Wissenschaftsverlag, Mannheim, Leipzig, Wien, Zurich, 1993.

6. D.E. Knuth. The Art of computer Programming. Volume 2: Seminumerical Algorithms. Addison-Wesley, Reading, Massachusetts, 2nd edition, 1981.

7. N. Koblitz. Elliptic curve cryptosystems. Mathematics of Computation, 48:203-209, 1987.

8. N. Koblitz. Hyperelliptic crytosystems. Journal of Cryptology, 1(3):129-150, 1989.

9. J. Koeller, A. Menezes, M. Qu,,and S. Vanstone. Elliptic Curve Systems. Draft 8, IEEE P 1363 Standard for RSA, Diffie-Hellman and Related Public-Key Cryptography, May 1996. Working document.

10. Kenji Koyama and Yukio Tsuruoka. Speeding up elliptic cryptosystems by using a signed binary window method. In Crypto '92. Springer Lecture Notes in Computer Science, 1992.

11. R. Lidl and H. Niederreiter. Finite Fields, volume 20 of Encyclopedia of Mathematics and its Applications. Addison-Wesley, Reading, Massachusetts, 1983.

12. A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone. Handbook of Applied Cryptography. CRC Press, 1997.

13. V. Miller. Uses of elliptic curves in cryptography. In Lecture Notes in Computer Science 218: Advances in Cryptology - CRYPTO '85, pages 417-426. Springer-Verlag, Berlin, 1986.

14. Atsuko Miyaji and Makoto Tatebayashi. Method for generating and verifying electronic signatures and privacy communication using elliptic curves. US Patent 5442707, 1995.

15. S. Paulus. Ein Algorithmus zur Berechnung der Klassengruppe quadratischer Ordnungen uber

Hauptidealringen. PhD thesis, Institute for Experimental Mathematics, University of Essen, Essen, Germany, June 1996.

16. R. Schroeppel, H. Orman, S. O'Malley, and O. Spatscheck. Fast key exchange with elliptic curve systems. Advances in Cryptology - CRYPTO '95, pages 43-56, 1995.

17. E. De Win, A. Bosselaers, S. Vandenberghe, P. De Gersem, and J. Vandewalle. A fast software implementation for arithmetic operations in $GF(2^n)$. In Asiacrypt '96. Springer Lecture Notes in Computer Science, 1996.

18. ANSI X9.62-199x. The Elliptic Curve Digital Signature Algorithm. Draft, January 1998. working document.

19. ANSI X9.63-199x. Elliptic Curve Key Agreement and Key Transport Protocols. Draft, January 1998. working document.