Efficient elliptic curve exponentiation

Atsuko Miyaji[1], Takatoshi Ono[2] and Henri Cohen[3]

[1] Multimedia Development Center, Matsushita Electric Industrial Co., LTD.
[2] Matsushita Information Systems Research Laboratory Nagoya Co., Ltd.
[3] Université Bordeaux

Abstract. Elliptic curve cryptosystems, proposed by Koblitz([8]) and Miller([11]), can be constructed over a smaller definition field than the ElGamal cryptosystems([5]) or the RSA cryptosystems([16]). This is why elliptic curve cryptosystems have begun to attract notice. There are mainly two types in elliptic curve cryptosystems, elliptic curves E over $\mathbb{F}_{2^r}$ and E over $\mathbb{F}_p$. Some current systems based on ElGamal or RSA may often use modulo arithmetic over $\mathbb{F}_p$. Therefore it is convenient to construct fast elliptic curve cryptosystems over $\mathbb{F}_p$. In this paper, we investigate how to implement elliptic curve cryptosystems on $E/\mathbb{F}_p$.

1 Introduction

Koblitz ([8]) and Miller ([11]) proposed a method by which public key cryptosystems can be constructed on the group of points on an elliptic curve over a finite field instead of a finite field. If elliptic curve cryptosystems avoid the Menezes-Okamoto-Vanstone reduction ([13]), then the only known attacks are the Pollard $\rho-$method ([15]) and the Pohlig-Hellman method ([14]). So up to the present, we can construct elliptic curve cryptosystems over a smaller definition field than the discrete-logarithm-problem(DLP)-based cryptosystems like ElGamal cryptosystems([5]) or DSA([3]) and the RSA cryptosystems([16]). Elliptic curve cryptosystems with 160-bit key have the same security as both ElGamal cryptosystems and RSA with 1,024-bit key. This is why elliptic curve cryptosystems have been discussed in ISO/IEC CD 14883-3, ISO/IEC DIS 11770-3, ANSI ASC X.9, X.9.62, and IEEE P1363([7]). As standardization is advanced, fast implementation of elliptic curve cryptosystems has been reported([6, 20, 22]).

There are mainly two types in elliptic curve cryptosystems, elliptic curves over $\mathbb{F}_{2^r}$ and elliptic curves over $\mathbb{F}_p$. Up to the present, the study on implementation has been often aimed at elliptic curves over $\mathbb{F}_{2^r}$ since arithmetic in $\mathbb{F}_{2^r}$ has an advantage of good performance in hardware. Practically speaking, however, DLP-based cryptosystems or RSA cryptosystems, both of which use modular arithmetic over $\mathbb{F}_p$, have been widely used in many systems. Therefore it would be convenient to construct elliptic curve cryptosystems over $\mathbb{F}_p$ since we can offer both RSA and elliptic curve cryptosystems with one modular arithmetic.

Elliptic curve cryptosystems mainly consist of elliptic curve exponentiations. This paper studies efficient elliptic curve exponentiation, which aims at elliptic curves over $\mathbb{F}_p$ but can be applied to any elliptic curve. Studies on elliptic curve

exponentiations are mainly classified into three factors: the coordinate, an exponentiation for a fixed point, and an exponentiation for a random point. This paper investigates these three factors:
1. **The coordinate:** Elliptic curve exponentiation can be computed by repeating additions and doublings, where the repeated number of additions can be reduced by a suitable algorithm, but that of doublings can not be reduced especially in the case of exponentiation for a random point. On the other hand, we can define some coordinates on an elliptic curve, which give each different addition formula. So we investigate the efficiency of the addition formula in jacobian coordinates([2]) which is less familiar than projective coordinates. Jacobian coordinates offer a slower addition but a faster doubling, which should be suitable for elliptic curve exponentiation.
2. **Exponentiation for a fixed point:** In this case, the precomputation table method([1]) is useful, which computes an elliptic curve exponentiation by repeating only additions and no doubling. In order to make use of a feature of jacobian coordinates, we propose a new algorithm which requires more doublings but fewer additions. Total computation amount for kG in our algorithm is less than [1].
3. **Exponentiation for a random point:** In this case, the addition-subtraction method is usually mixed with the window method([10, 12, 9, 20]). In this approach, an interval between two windows mainly determines the computation amount: the longer the interval is, the less the computation amount is. Importantly, signed binary representation of k is not determined uniquely, while an interval differs for each signed binary representation. An average interval by mixing the signed binary representation in [12] and the window method is 4/3, that in [9] is 3/2. Here we present a new method for signed binary representation, which improves the average interval to 2 by mixing with the window method.

This paper is organized as follows. Section 2 discusses jacobian coordinates. Section 3 investigates each suitable algorithm for exponentiation of a fixed point and a random. Two implements of a 160-bit definition field and a 169-bit definition field are presented in appendices.

2 The coordinate

An elliptic curve can be represented by several sets of coordinates. The addition formula, which is defined by setting a point at infinity $\mathcal{O}$ to zero, differs for each coordinate: the computation amount of addition differs for each coordinate. Two coordinates, affine coordinates and projective coordinates, are well known([19]). Affine coordinate requires a division in every addition and every doubling but requires fewer multiplications than projective coordinate. On the other hand, projective coordinate does not require any division in either addition or doubling and does require a division only once in the final stage of the computation of elliptic curve exponentiation. In the case of $\mathbb{F}_p$ the computation of elliptic curve exponentiation in projective coordinates is faster than that in affine coordinates since the ratio of the computation amount of division in $\mathbb{F}_p$ to that of

multiplication in $\mathbb{F}_p$ is generally larger than 9.

Here we discuss another coordinate([2]), which is called jacobian coordinate in this paper. The addition formula in jacobian coordinates is similar to projective coordinate: it does not require any division modulo p in either addition or doubling and does require a division only once in the final stage of the computation of elliptic curve exponentiation. However, jacobian coordinates offer a doubling with less computation amount but an addition with more computation amount than projective coordinates. This feature should be suitable for elliptic curve exponentiation since the number of additions required in elliptic curve exponentiation can be reduced by a suitable algorithm, but that of doublings may not be reduced.

Here we presents the addition formula in jacobian coordinate which is a slightly revised version of ([2]). Let an elliptic curve over $\mathbb{F}_p (p > 3)$ be

$$E : y^2 = x^3 + ax + b \ \ (a, b \in \mathbb{F}_p, 4a^3 + 27b^2 \neq 0).$$

For the elliptic curve, the jacobian coordinate sets $x = X/Z^2$ and $y = Y/Z^3$ i.e.

$$E : Y^2 = X^3 + aXZ^4 + bZ^6.$$

The addition formulae in the jacobian coordinates are the following. Let $P = (X_1, Y_1, Z_1)$, $Q = (X_2, Y_2, Z_2)$ and $P + Q = R = (X_3, Y_3, Z_3)$.

• **Curve addition formula in jacobian coordinates ($P \neq \pm Q$)**

$$X_3 = -H^3 - 2U_1H^2 + r^2, Y_3 = -S_1H^3 + r(U_1H^2 - X_3), Z_3 = Z_1Z_2H,$$

where $U_1 = X_1Z_2^2, U_2 = X_2Z_1^2, S_1 = Y_1Z_2^3, S_2 = Y_2Z_1^3, H = U_2 - U_1, r = S_2 - S_1$;

• **Curve doubling formula in jacobian coordinates ($R = 2P$)**

$$X_3 = T, Y_3 = -8{Y_1}^4 + M(S - T), Z_3 = 2Y_1Z_1,$$

where $S = 4X_1Y_1^2, M = 3X_1^2 + aZ_1^4, T = -2S + M^2$.

Here we discuss the computation amount of addition formulae. We denote the computation amount for 1 multiplication(resp. division) in $\mathbb{F}_p$ by M(resp. D). For simplicity, we neglect addition, subtraction and multiplication by a small constant in $\mathbb{F}_p$ because they are much faster than multiplication and division in $\mathbb{F}_p$. Table 1 presents the number of multiplications in addition formula of jacobian coordinates and projective coordinates, where the addition formula in projective coordinates is presented in Appendix A. Table 1 includes the following special cases: in the computation of a fixed point, we may set Z_1 to one or both Z_1 and Z_2 to one in addition formula. Doubling formula depends on a coefficient a of an elliptic curve: in the case of $a = 0$ or $a = -3$(setting $w = 3X_1^2 - 3Z_1^4 = 3(X_1 - Z_1^2)(X_1 + Z_1^2)$), the computation amount is reduced. Note that addition formula does not depend on coefficients.

	addition			doubling		
	$Z_1, Z_2 \neq 1$	$Z_1 = 1$	$Z_1 = Z_2 = 1$	$a \neq 0, -3$	$a = 0$	$a = -3$
Projective coordinate	$14M$	$11M$	$7M$	$12M$	$10M$	$10M$
Jacobian coordinate	$16M$	$11M$	$6M$	$10M$	$7M$	$8M$

Table 1. Number of multiplications in addition formula

3 Elliptic curve exponentiation

This section discusses elliptic curve exponentiations for a fixed point and a random point. Both discussion depends neither on the size of definition field nor on the characteristic of definition field.

3.1 Exponentiation for a fixed point

For simplicity, here we assume a 160-bit definition field $\mathbb{F}_p$. As for a fixed point, the precomputation table method is useful([1]): it prepares a table of 40 points $16^i G$ for $i = 1, \cdots, 40$, each of which Z-coordinate is set to 1, and computes kG in about 44 additions/subtractions. Since this method does not require any doubling, projective coordinate is suitable. The computation amount sums up to $500M + D$, considering carefully three cases of addition in Table 1. Note that the computation amount does not depend on a coefficient a of an elliptic curve.

As we have seen in Section 2, the computation amount of doubling is less than that of addition. Therefore we would reduce the number of additions rather than that of doublings. Here we describe another method which requires more doublings but fewer additions than [1]. The tables consist of 62 points,

$$A[s] = \sum_{j=0}^{4} a_{s,j} 2^{32j} G \text{ and } B[s] = \sum_{j=0}^{4} a_{s,j} 2^{16+32j} G \ \ (1 \leq s \leq 31),$$

where $a_{s,0}, \cdots, a_{s,4}$ is a representation of s in radix 2, that is $s = \sum_{j=0}^{4} a_{s,j} 2^j$. The Z-coordinates of 62 points are also set to 1. Then the algorithm to compute kG is as follows, where k is set to $k = \sum_{j=0}^{159} k[j] 2^j$:

`Algorithm 1.`
`1. set` $u_j = \sum_{i=0}^{4} k[32i + j] 2^i$ `and` $v_j = \sum_{i=0}^{4} k[32i + 16 + j] 2^i$ `for` $0 \leq j \leq 15$
`2. set` $A[0] = \mathcal{O}, B[0] = \mathcal{O}$, `and` $T = \mathcal{O}$
`3. for` $i = 15$ `to 0 by` -1
$T = 2T$ `and` $T = T + A[u_i] + B[v_i]$
`4. output` $T = kG$.

The above algorithm computes kG by 30 additions and 15 doublings. Jacobian coordinate is suitable for this method. By using jacobian coordinate, the computation amount sums up to $479M + D$. If we set a coefficient $a = 0$ of elliptic curve, the computation amount is reduced to $434M + D$.

To sum up, our method with jacobian coordinate can reduce the computation amount of kG by 4% of [1]. Furthermore by selecting a suitable elliptic curve like a coefficient $a = 0$ it is reduced by 13% of [1].

3.2 Exponentiation for a random point

As for the computation of kP for a random point P, the addition-subtraction method is commonly mixed with the window method([10, 12, 9, 20]). In this approach, an interval between two windows mainly determines the computation amount for kP: the longer the interval is, the less the computation amount is. Importantly, signed binary representation of k is not determined uniquely,

while an interval differs for each signed binary representation. Here we present new signed binary representation, which can offer a longer interval by mixing with the window method. A feature of our method is that the signed binary representation depends on a width in the window method. On the other hand, known representation([10, 12, 9]) is independent of the window method.

Let $n = \lfloor log_2(k) \rfloor$, $k = \sum_{i=0}^{n} k[i]2^i$ $(k[i] = 0, 1)$ in binary representation, and w be a width in the window method. The following algorithm transforms k into $k' = \sum_{i=0} k'[i]2^i$ $(k'[i] = 0, \pm 1)$ in signed binary while setting windows $W[j](j = 0, 1, \cdots)$:

Algorithm 2.
1. set $i = 0$, $j = 0$ and $k[n+1] = 0$
2. while $i \leq n$ do:
  if $i + w - 1 \geq n - w$, then set $k'[i] = k[i], \cdots, k'[n] = k[n]$,
    set $W[j] = (k[i+w-1], \cdots, k[i+1], k[i])$ and goto 3.
  if $k[i] = 0$, then set $k'[i] = k[i]$, $i = i + 1$, and goto 2.
  if $k[i] = 1$, then set $t[j] = \sum_{t=0}^{w-1} k[i+t]2^t$
    if $k[i+w] = 0$, then set $k'[i] = k[i], \cdots, k'[i+w] = k[i+w]$,
      set $W[j] = (k[i+w-1], \cdots, k[i+1], k[i])$,
      set $j = j + 1$, $i = i + w + 1$ and goto 2.
    if $k[i+w] = 1$, then for first t satisfying $k[t] = 0$ $(t = i + w + 1, \cdots)$
      set $k'[i+w] = 0, \cdots, k'[t-1] = 0$, and $k[t] = 1$,
      set $t[j] = 2^w - t[j] = \sum_{t=0}^{w-1} k'[i+t]2^t$ (in binary representation),
      set $k'[i] = -k'[i], \cdots, k'[i+w-1] = -k'[i+w-1]$,
      set $W[j] = (k'[i+w-1], \cdots, k'[i+1], k'[i])$,
      set $i = t, j = j + 1$ and goto 2.
3. output $k' = \sum_{i=0} k'[i]2^i$ $(k'[i] = 0, \pm 1)$ and $W[i](i = 0, 1, \cdots)$.

Let the most significant window be $W[s-1] = (k'[i+w-1], \cdots, k'[i+1], k'[i])$. For convenience, set $W[s] = (k'[n], \cdots, k'[i+w])$, where $i + w \geq n - w + 1$ from Algorithm 2. Then k' is written as

$$k' = 2^{t_0}(2^{t_1}(\cdots 2^{t_{s-1}}(2^{t_s}W[s] + W[s-1])\cdots) + W[0]) \ (0 \leq t_i).$$

In order to decrease the number t_s of doublings, we revise $W[s]$ to a window with a length at most w from MSB and fit the most significant bit of $W[s-1]$ to the new $W[s]$, while leaving others as they are. Then we can compute kP from MSB to LSB by using the transformation of k after preparing points $P, 3P, \cdots, (2^w - 1)P$.

Example. Set $w = 4$. For a given $k = 101101100110110111$ in binary representation, the algorithm transforms k into:

$k' = \underline{101}\,\underline{1011}\,0\,\underline{0111}\,00\,\underline{\overline{1}00\overline{1}} = W[3]\,W[2]\,0\,W[1]\,00\,W[0]$,

revise k' to:

$k' = \underline{1011}\,\underline{011}\,0\,\underline{0111}\,00\,\underline{\overline{1}00\overline{1}} = W[3]\,W[2]\,0\,W[1]\,00\,W[0]$,

where $\overline{1}$ denotes -1 and each block of underlined digits represents one window like $W[0]$. Then kP can be computed by $2^6(2^5(2^3W[3] + W[2]) + W[1]) + W[0]$.

Let estimate the computation amount of exponentiation in this method. First we show that an average interval between two windows is 2:

Theorem 1. *Algorithm 2 constructs windows at an average interval of 2 bits.*

Proof. Let $W[j] = (k[i+w-1], \cdots, k[i+1], k[i])$ be a window. Then we show the next window $W[j+1]$ will start at $k[i+w+2]$ on the average. From Algorithm 2, the next window never starts at $k[i+w]$. The next window starts at $k[i+w+1]$ if and only if $(k[i+w+1], k[i+w]) = (0,1), (1,0)$. Therefore the probability of starting at $k[i+w+1]$ is $\frac{1}{2}$. The next window starts at $k[i+w+2]$ if and only if $(k[i+w+2], k[i+w+1], k[i+w]) = (1,0,0), (0,1,1)$. Therefore the probability of starting at $k[i+w+2]$ is $(\frac{1}{2})^2$. Thus, an average interval between $W[j]$ and $W[j+1]$ is computed in $\sum_{i=1} i * (\frac{1}{2})^i \simeq 2$. Therefore the next window will start at $k[i+w+2]$ on the average. □

From the above theorem, we obtain the following approximate multiplication count $T_w(n)$ for raising a point P to the n-th power by setting $u = \sum_{i=0}^{s} t_i$ and L to be the average length of k' and using jacobian coordinates

$$T_w(n) = (16 - \tfrac{5}{2^{w-1}})(\tfrac{L}{w+2}) + 10u + 16 * 2^{w-1} - 15.$$

Then it is easily shown that $u = (n + 2 - (\frac{1}{2})^w - w)$ and $L = n + \frac{3}{2}$. Therefore we obtain

$$T_w(n) = (16 - \tfrac{5}{2^{w-1}})(\tfrac{n+3/2}{w+2}) + 10(n + 2 - (\tfrac{1}{2})^w - w) + 16 * 2^{w-1} - 15.$$

Choosing w to make $T_w(n)$ minimal in the range of $150 < n < 170$, we get $w = 4$ is optimal since $T_3(n) > T_4(n) < T_5(n)$.

We discuss one case of $n = 159$, in which implementation is presented in Appendix B. Our algorithm computes kP in 33.7 additions and 157.9 doublings. The total computation amount of our algorithm with jacobian coordinates is $2098M + D$. On the other hand, the method of [9] requires $2384M + D$ in projective coordinates or $2141M + D$ in jacobian coordinates.

4 Conclusion

In this paper, we have been proposed efficient elliptic curve exponentiations for a fixed and a random point. As for a fixed point, our method with more doublings but fewer additions can compute kG with 160-bit k in 30 additions and 15 doublings. As for a random point, our method of mixing new signed representation with the window method can compute kP with 160-bit k in 33.7 additions and 157.9 doublings. The use of jacobian coordinate gives further improvement to the running time: elliptic curve exponentiation for a fixed point can be computed in $479M + D$ and that for a random point can be computed in $2098M + D$.

References

1. E. F. Brickell, D. M. Gordon, K. S. McCurley and D. B. Wilson, "Fast exponentiation with precomputation" *Advances in Cryptology-Proceedings of EUROCRYPT'92*, Lecture Notes in Computer Science, **658**(1993), Springer-Verlag, 200-207.

2. D. V. Chudnovsky and G. V. Chudnovsky "Sequences of numbers generated by addition in formal group and new primality and factorization tests" *Advances in Applied Math.*, **7**(1986), 385-434.
3. "Proposed federal information processing standard for digital signature standard (DSS)", *Federal Register*, v. 56, n. 169, 30 Aug 1991, 42980-42982.
4. W. Diffie and M. Hellman, "New directions in cryptography" *IEEE Trans. Inform. Theory*, Vol. IT-22 (1976), 644-654.
5. T. ElGamal, "A public key cryptosystem and a signature scheme based on discrete logarithms", *IEEE Trans. Inform. Theory*, Vol. IT-31 (1985), 469-472.
6. G. Harper, A. Menezes and S. Vanstone, "Public-key cryptosystems with very small key lengths", *Advances in Cryptology-Proceedings of Eurocrypt'92*, Lecture Notes in Computer Science, **658**(1993), Springer-Verlag, 163-173.
7. *IEEE P1363 Working Draft*, February 6, 1997.
8. N. Koblitz, "Elliptic curve cryptosystems", *Mathematics of Computation*, **48**(1987), 203-209.
9. K. Koyama and Y. Tsuruoka, "Speeding up elliptic cryptosystems by using a signed binary window method", *Abstract of proceedings of CRYPTO'92*, 1992.
10. D. E. Knuth, *The art of computer programming*, *vol. 2*, *Seminumerical Algorithms*, 2nd ed., Addison-Wesley, Reading, Mass. 1981.
11. V. S. Miller, "Use of elliptic curves in cryptography", *Advances in Cryptology-Proceedings of Crypto'85*, Lecture Notes in Computer Science, **218**(1986), Springer-Verlag, 417-426.
12. F. Morain and J. Olivos, "Speeding up the computations on an elliptic curve using addition-subtraction chains", Theoretical Informatics and Applications Vol.24, No.6 (1990), 531-544.
13. A. Menezes, T. Okamoto and S. Vanstone, "Reducing elliptic curve logarithms to logarithms in a finite field", *Proceedings of the 22nd Annual ACM Symposium on the Theory of Computing*, 80-89, 1991.
14. S. C. Pohlig and M. E. Hellman, "An improved algorithm for computing logarithm over $GF(p)$ and its cryptographic significance", *IEEE Trans. Inf. Theory*, IT-24(1978), 106-110.
15. J. Pollard, "Monte Carlo methods for index computation(mod p)", *Mathematics of Computation*, **32**(1978), 918-924.
16. R. Rivest, A. Shamir and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems", *Communications of the ACM*, vol.21, No.2(1978), 120-126.
17. B. Schneier *Applied cryptography*, II, John Wiley & Sons, Inc. 1996.
18. C. P. Schnorr, "Efficient identification and signatures for smart cards", *Advances in cryptology-Proceedings of Crypto'89*, Lecture Notes in Computer Science, **435**(1989), Springer-Verlag, 239-252.
19. J. H. Silverman, *The Arithmetic of Elliptic Curves*, GTM106, Springer-Verlag, New York, 1986.
20. R. Schroeppel, H. Orman, S. O'Malley and O. Spatscheck, "Fast key exchange with elliptic curve systems", *Advances in Cryptology-Proceedings of Crypto'95*, Lecture Notes in Computer Science, **963**(1995), Springer-Verlag, 43-56.
21. Torbjorn Granlund, The GNU MP LIBRARY, version 2.0.2, June 1996. ftp://prep.ai.mit.edu/pub/gnu/gmp-2.0.2.tar.gz
22. E. D. Win, A. Bosselaers and S. Vandenberghe "A fast software implementation for arithmetic operations in GF(2^n)", *Advances in Cryptology-Proceedings of Asiacrypt'95*, Lecture Notes in Computer Science, **1163**(1996), Springer-Verlag, 65-76.

A The addition formula in projective coordinate

Let an elliptic curve over $\mathbb{F}_p (p > 3)$ be

$$E : y^2 = x^3 + ax + b \ \ (a, b \in \mathbb{F}_p, 4a^3 + 27b^2 \neq 0).$$

For the elliptic curve, the projective coordinate sets $x = X/Z$ and $y = Y/Z$ i.e.

$$E : Y^2 Z = X^3 + aXZ^2 + bZ^3.$$

The addition formulae in projective coordinates are the following. Let $P = (X_1, Y_1, Z_1)$, $Q = (X_2, Y_2, Z_2)$ and $P + Q = R = (X_3, Y_3, Z_3)$.
• **Curve addition formula in projective coordinates ($P \neq \pm Q$)**

$$X_3 = vA, Y_3 = u(v^2 X_1 Z_2 - A) - v^3 Y_1 Z_2, Z_3 = v^3 Z_1 Z_2, \tag{1}$$

where $u = Y_2 Z_1 - Y_1 Z_2, v = X_2 Z_1 - X_1 Z_2, A = u^2 Z_1 Z_2 - v^3 - 2v^2 X_1 Z_2$;
• **Curve doubling formula in projective coordinates ($R = 2P$)**

$$X_3 = 2hs, Y_3 = w(4B - h) - 8{Y_1}^2 s^2, Z_3 = 8s^3, \tag{2}$$

where $w = a{Z_1}^2 + 3{X_1}^2, s = Y_1 Z_1, B = X_1 Y_1 s, h = w^2 - 8B$.

B Implementation of elliptic curve exponentiations

B.1 Elliptic curves

Elliptic curves $E/\mathbb{F}_p$ with order divisible by 160-bit or more prime is secure if it satisfies MOV-condition([7]). As we have seen in Section 2, the computation amount of doubling is reduced in the case of a coefficient $a = 0$ or -3 of elliptic curve. Here we implement two elliptic curves with a coefficient $a = 0$ in 160-bit and 169-bit key size.
1. 160-bit key size

- a definition field $\mathbb{F}_{p_1}$: $p_1 = 2^{160} - 2013$
- an elliptic curve E_1: $y^2 = x^3 + 4$, $\#E_1(\mathbb{F}_{p_1}) = 3 * 13 * q_1$, where q_1 is a prime
  $q_1 = 37\ 47440\ 09572\ 02638\ 92829\ 95765\ 50867\ 08565\ 09759\ 22411$
- a basepoint G_1: $(x_1, y_1) \in E_1(\mathbb{F}_{p_1})$ with order q_1, where
  $x_1 = 1312\ 01277\ 27149\ 38861\ 46561\ 78958\ 06449\ 61829\ 03474\ 73840$
  $y_1 = 1143\ 61120\ 94309\ 35596\ 62639\ 62368\ 56710\ 92306\ 44246\ 02993$

2. 169-bit key size

- a definition field $\mathbb{F}_{p_2}$: $p_2 = 2^{169} - 1825$
- an elliptic curve E_2: $y^2 = x^3 + 49$, $\#E_2(\mathbb{F}_{p_2}) = 3 * 67 * q_2$, where q_2 is a prime
  $q_2 = 3722\ 83004\ 13603\ 09920\ 99645\ 09743\ 01139\ 56489\ 04413\ 35543$
- a basepoint G_2: $(x_2, y_2) \in E_2(\mathbb{F}_{p_2})$ with order q_2, where
  $x_2 = 1\ 55608\ 20629\ 69890\ 07722\ 36926\ 87616\ 67589\ 98487\ 34687\ 95184$
  $y_2 = 55502\ 35686\ 97076\ 18367\ 46840\ 54359\ 62467\ 42560\ 87632\ 81833$

Here we discuss cryptographic differences between E_1 and E_2. From a security and efficiency point of view, these two elliptic curves give almost the same security, while the exponentiations in E_1 can be computed faster than that in E_2. However, from an application point of view, ElGamal encryption on E_2 can encrypt a 168-bit Triple-DES key([17]) but E_1 can not.

B.2 The running time

Here presents the running time of elliptic curve exponentiations over 160-bit and 169-bit definition fields in our methods. Our modulo arithmetic uses GNU MP Library GMP([21]) in order to make easy comparison possible since GMP might be most popular multiprecision library. The platform is SS-5(MicroSPARC 110 MHz/Solaris 2.4) with a 32 bit word size. Table 2 shows the running time.

ElGamal encryption on an elliptic curve mainly consists of one exponentiation of a fixed point and one exponentiation of a random point. Therefore we can estimate that Triple-DES key can be encrypted in about 42 msec.

	$E_1/\mathbb{F}_{p_1}(p_1 = 2^{160} - 2013)$	$E_2/\mathbb{F}_{p_2}(p_2 = 2^{169} - 1825)$
160 bit /169 bit addition	0.88 μsec	1.31 μsec
160 bit /169 bit multiply	9.66 μsec	12.04 μsec
160 bit /169 bit square	7.65 μsec	9.77 μsec
modular reduction	4.81 μsec	5.76 μsec
160 bit /169 bit division	0.91 msec	1.00 msec
addition(jacobian-coordinate)	0.23 msec	0.30 msec
doubling(jacobian-coordinate)	0.12 msec	0.13 msec
exponentiation of a fixed point	7.79 msec	9.31 msec
exponentiation of a random point	26.93 msec	32.54 msec

Table 2. Time of elliptic curve operations

This article was processed using the LaTeX macro package with LLNCS style