Speeding up Elliptic Cryptosystems by Using a Signed Binary Window Method

Kenji Koyama, Yukio Tsuruoka

NTT Communication Science Laboratories
Seikacho, Kyoto, 619-02 Japan

Abstract. The basic operation in elliptic cryptosystems is the computation of a multiple $d \cdot P$ of a point P on the elliptic curve modulo n. We propose a fast and systematic method of reducing the number of operations over elliptic curves. The proposed method is based on pre-computation to generate an adequate addition-subtraction chain for multiplier the d. By increasing the average length of zero runs in a signed binary representation of d, we can speed up the window method. Formulating the time complexity of the proposed method makes clear that the proposed method is faster than other methods. For example, for d with length 512 bits, the proposed method requires 602.6 multiplications on average. Finally, we point out that each addition/subtraction over the elliptic curve using homogeneous coordinates can be done in 3 multiplications if parallel processing is allowed.

1 Introduction

Elliptic curves over a finite field $\mathbf{F}_p$ or a ring $\mathbf{Z}_n$ can be applied to implement analogs [9] [11] [13] of the Diffie-Hellman scheme [4], ElGamal scheme [6] and RSA scheme [15], as well as primality testing [7] and integer factorization [12][13]. Cryptosystems based on elliptic curves, called *elliptic cryptosystems*, seem more secure than the original schemes. For example, it is conjectured that the low exponent attack on the RSA scheme cannot be analogously applied to the attack on the elliptic RSA scheme using a low multiplier [9]. The basic operation performed on an elliptic curve is the computation of a multiple $d \cdot P$ of a point P on the elliptic curve modulo n, which corresponds to the computation of x^d mod n. For a large n and d, the time complexity of elementary operations as well as the number of elementary operations are very high. Thus, reducing the number of such operations is important when implementing the above algorithms.

One solution is a so-called *binary method* [10] based on the *addition chain* [10] for multipliers d of $d \cdot P$ or exponents d of x^d. In general, an addition chain for a given d is a sequence of positive integers

$$a_0\ (= 1) - a_1 - a_2 - \cdots - a_r\ (= d),$$

where r is the number of additions, and $a_i = a_j + a_k$, for some $k \leq j < i$, for all $i = 1, 2, \ldots, r$. The binary method is a systematic algorithm based on an addition chain with elements that are powers of 2, i.e. a two-valued binary representation of d. To evaluate $d \cdot P$ or x^d, the ordinary binary method without pre-computation requires $\frac{3}{2} \lfloor \log_2 d \rfloor$ multiplications on average. The ordinary binary method does not always guarantee the minimum number of multiplications (the shortest addition chain). Obtaining the shortest addition chain is a NP-complete problem [5]. There have been many studies on the computation of x^d [1] [2] [3] [8] [16] and a few studies on the computation of $d \cdot P$ [14] to achieve fast and efficient computation. Among the variants of the binary method attempted to speed up the computation of x^d, Bos and Coster [1] proposed a heuristic window method based on an *addition sequence*. The addition sequence is a generalized addition chain including the given set of values. In their algorithm, the two-valued binary representation of d is split into pieces (windows), and the value of each window is computed in shorter addition sequence.

An addition chain can be extended to an ***addition-subtraction chain*** [2] [10] [14], with a rule $a_i = a_j \pm a_k$ in place of $a_i = a_j + a_k$. This idea corresponds to the evaluation of x^d using multiplication and division. For integers, division (or the computation of a multiplicative inverse modulo n) is a costly operation, and implementing this idea does not seem feasible. The reason why elliptic curves are so attractive is that the division in $\mathbf{Z}_n$ is replaced by a subtraction, which has the same cost as an addition. An addition (subtraction) formula on elliptic curves does not contain a division in $\mathbf{Z}_n$ particularly when homogeneous coordinates are used. Thus, the addition-subtraction chain can be effectively applied to computations over elliptic curves.

This paper proposes a fast and systematic method of computing a multiple $d \cdot P$ of a point P on the elliptic curve modulo n. By increasing the average length of zero runs in a signed binary representation of d, the window method can be speeded up. The organization of this paper is as follows. Section 2 describes a new signed binary window method, clarifying the difference between previous methods and the new method, and analyzes the number of operations for the proposed method. Section 3 shows that the proposed method is faster than other methods. Elliptic curves over a finite field and a ring and the addition formula over elliptic curves are briefly reviewed in Section 4. Then, serial/parallel computations implemented in homogeneous coordinates and affine coordinates are compared.

2 New Method

The proposed method is a window method based on an adequately chosen signed binary representation of d. The new method is described, clarifying the differences between the previous window method [1] based on ordinary binary representation and the new one in this section. The window method is an extension of the 2^k-ary method. For a given number d, the window method consists of four phases: (1) representation of d, (2) splitting the representation into segments (windows), (3) computing all the segments, and (4) concatenating all the segments.

2.1 Representation

For a given number d, the original window method uses an ordinary (two-valued) binary representation $B : (b_\lambda, b_{\lambda-1}, \ldots, b_0)$, where $b_i \in \{0,1\}$, $\lambda = \lfloor \log_2 d \rfloor$. The proposed method uses the signed (three-valued) binary representation $T : [t_{L-1}, \ldots, t_1, t_0]$, $t_{L-1} \neq 0$ for d satisfying $d = \sum_{i=0}^{L-1} t_i 2^i$, where $t_i \in \{\bar{1}, 0, 1\}$, and $\bar{1}$ denotes -1. Note that in ordinary binary representation B is uniquely determined for a given d, but T is not.

Morain-Olivos [14] and Jedwab-Mitchell [8] proposed algorithms to transform B into the equivalent T, for minimizing the weight (the number of non-zero digits) of T. Note that Morain-Olivos's method (MO method) is equivalent to Jedwab-Mitchell's method (JM method). We propose a new transform algorithm which increases the average length of zero runs in T, while minimizing the weight of T. The average length of the zero runs in specific T, denoted by $Z(T)$, is defined as follows.

$$Z(T) = \frac{1}{L}\sum_{i=0}^{L-1} z(i), \qquad z(i) = \begin{cases} 1 + z(i-1) & \text{if } t_i = 0 \\ 0 & \text{if } t_i \neq 0 \end{cases} \qquad (0 \le i \le L-1),$$

where $z(-1) = 0$.

Let B' be a subsequence of B, and let T' be a subsequence of T. A rule for transforming B' to equivalent T' is as follows.

Transformation Rule

$B' : (1 \cdots b_i \cdots 1)$ can be transformed into $T' : [10 \cdots t_i \cdots \bar{1}]$, where $t_i = b_i - 1$.

Let $\#_0(B')$ be a number of zeroes in B', and let $\#_1(B')$ be a number of non-zero digits in B'. The weight of T' is estimated as $\#_1(T') = 2 + \sum |t_i| = 2 + \sum |b_i - 1| = 2 + \#_0(B')$. Thus, the weight decreases by the transformation if $\#_1(B') - \#_0(B') > 2$

The proposed transform algorithm inputs B in LSB first order and counts the difference $D(B') \equiv \#_1(B') - \#_0(B')$, and applies the transformation rule repeatedly to appropriate B' with $D(B') \ge 3$. The main difference between the proposed method and other methods is a threshold value (such as 3) to apply the transformation rule. MO method applies the rule to B' with $D(B') \ge 2$. Further, the output of both MO method and JM method are *sparse*, which means no two adjacent digits are nonzero. However, the output of the proposed method is not sparse. MO method and the proposed method generate T with same the same weight. Thus, the average length of zero runs of output of the proposed method is greater than that of MO method.

The proposed transform algorithm is shown below.

```
algorithm transform (input $B$: array, output $T$: array)
  begin
      $M := 0; J := 0; Y := 0; X := 0; U := 0; V := 0; W := 0; Z := 0;$
      while $X < \lfloor \log_2 d \rfloor$ do begin
          if $B[X] = 1$ then $Y := Y + 1$ else $Y := Y - 1$;
          $X := X + 1$;
          if $M = 0$ then begin
              if $Y - Z \geq 3$ then begin
                  while $J < W$ do begin $T[J] := B[J]; J := J + 1$ end;
                  $T[J] := -1;\ J := J + 1;\ V := Y;\ U := X;\ M := 1$
              end else if $Y < Z$ then begin $Z := Y; W := X$ end
          end else begin
              if $V - Y \geq 3$ then begin
                  while $J < U$ do
                      begin $T[J] := B[J] - 1; J := J + 1$ end;
                  $T[J] := 1;\ J := J + 1;\ Z := Y;\ W := X;\ M := 0$
              end else if $Y > V$ then begin $V := Y;\ U := X$ end
          end
      end;
      if $M = 0 \vee (M = 1 \wedge V \leq Y)$ then begin
          while $J < X$ do begin $T[J] := B[J] - M; J := J + 1$ end;
          $T[J] := 1 - M; T[J + 1] := M$
      end else begin
          while $J < U$ do begin $T[J] := B[J] - 1; J := J + 1$ end;
          $T[J] := 1;\ J := J + 1;$
          while $J < X$ do begin $T[J] := B[J]; J := J + 1$ end;
          $T[J] := 1; T[J + 1] := 0$
      end
      return $T$
  end
```

[Example] For a given $d = 25722562047811804942$, the binary representation for d is:

B : $(1011001001111110001110010111100110000001001100010001011111100001110)$

MO method transforms B into:

T : $[10\bar{1}0\bar{1}001010000\bar{1}00100\bar{1}010\bar{1}000\bar{1}010\bar{1}0000001010\bar{1}00010010\bar{1}0000\bar{1}000100\bar{1}0]$

The proposed algorithm transforms B into:

T : $[1011001010000\bar{1}001000\bar{1}\bar{1}0\bar{1}0000\bar{1}\bar{1}0\bar{1}000000100110001000110000\bar{1}000100\bar{1}0]$

The average length of zero runs for MO method is 1.29 and that of the proposed algorithm is 1.42.

2.2 Splitting

The splitting phase is common to both ordinary binary representation B and signed binary representation T. Let w be the width of the window. B or T is split into segments with a length at most w. The following splitting procedure generates a list of all segments. For simplicity, the input array is represented by T.

```
procedure split (input $T$: array, $w$: integer, output $S$: array)
    Let segment list $S$ be empty
    while (length($T$) $\geq w$)
      begin
        Let $W$ be the left $w$ digits of $T$.
        Let $R$ be $T$ excluding $W$.
        Let $\widetilde{W}$ be $W$ excluding the right 0's.
        Let $\widetilde{R}$ be $R$ excluding the left 0's.
        Add new segment $\widetilde{W}$ to segment list $S$
        $T$:= $\widetilde{R}$.
      end
    Add last segment $T$ to segment list $S$
    return $S$
```

[Example] Assume T is the signed binary representation generated by the transform algorithm in the previous example. When $w = 4$, the splitting algorithm outputs the list of segments as

$$[\underline{1011}00\underline{101}0000\underline{\bar{1}001}000\underline{\bar{1}\bar{1}0\bar{1}}0000\underline{\bar{1}\bar{1}0\bar{1}}0000000\underline{1001}\ \underline{1}000\underline{1}000\underline{11}0000\underline{\bar{1}}000\underline{100\bar{1}}0]$$

where each block of underlined digits represents a segment. Note that the transform algorithm increases the run length of 0's in the segment gaps.

2.3 Computing the Segments

In B, the value of each segment is an odd positive integer up to $2^w - 1$. In T, if $w \geq 3$, the segment value never becomes $2^w - 1$ or $-(2^w - 1)$ because of the property of the transform algorithm. Thus, each segment value is an odd integer from $-(2^w - 3)$ to $2^w - 3$. The absolute values of all segments are obtained by the following simple addition sequence. (*i.e.* $1, 2, 3, 5, 7, \ldots, 2^w - 3$)

$$a_0 = 1, \quad a_1 = 2, \quad a_2 = 3, \quad a_i = a_{i-1} + 2 \qquad (3 \leq i \leq 2^{w-1} - 1)$$

Therefore, in T, all segment values can be computed by at most $2^{w-1} - 1$ additions. In B, all segment values can be computed by at most 2^{w-1} additions.

For the above example, segment values become $\{11, 5, 7, -13, -13, 9, 1, 1, 3, -1, 7\}$. Thus, all (absolute) segment values are computed by an addition sequence as $1, 2, 3, 5, \ldots, 13$ in 7 additions.

In reference [1], Bos and Coster computed all segments using a heuristic addition sequence. When the distribution of segment values is sparse, the heuristic method may be effective. However, if the distribution is dense a systematic method may be more effective. When $\lambda = 511$ and $w = 5$, the distribution becomes dense and consequently the proposed systematic method is more effective.

2.4 Concatenating and The Number of Operations

Concatenation requires doublings and non-doubling additions.
For example, for the split T in the above example, concatenation is achieved by:
$dP = ((\cdots(((11P \cdot 2^{2+3} + 5P) \cdot 2^{4+4} - 7P) \cdot 2^{3+4} - 13P)\cdots) \cdot 2^{4+1} - P) \cdot 2^{3+4} + 7P) \cdot 2^{1}$.
The inner most $11P$ corresponds to the most significant segment, and the exponent $2+3$ corresponds to the sum of the length 2 of the following window gap and the length 3 of the next segment.

Let L be the length of B or T. Note that L is $\lambda + 1$ for B and L is $\lambda + 1$ or $\lambda + 2$ for T. Let Z' be the average length of zero runs in the most significant windows for B or T. In other words, Z' is the average number of 0's deleted in W' by the splitting algorithm in the beginning. Let Z'' be the average length of zero runs deleted in R by the splitting algorithm for B or T. The average length of the most significant segment is $w - Z'$. The number of doublings in concatenation is same as the length of T (or B) except for the most significant segment. Thus, the number of doublings in concatenation is $L - (w - Z')$ for B and T. The average number of segments becomes $L/(w + Z'')$, which corresponds to the number of non-doubling additions in concatenation.
Thus, on average, the window method requires R operations:

$$R = (L + Z' - w) + \frac{L}{w + Z''} + C,$$

where $C = 2^{w-1}$ for B, and $C = 2^{w-1} - 1$ for T.

2.5 Analysis of the number of operations

In this subsection, parameters L, Z', Z'' and w in the above expression R are analyzed.
The length of T is either $(\lambda + 1)$ or $(\lambda + 2)$. The transform algorithm outputs T of length $\lambda + 2$ with probability 1/4. Thus, the average length of T, denoted by $\overline{L}$, is expressed by $\overline{L} = \frac{1}{4} \cdot (\lambda + 2) + \frac{3}{4} \cdot (\lambda + 1) = \lambda + 5/4$.

Let p be the probability that 0 occurs in B. If each digit in B is independent, a straight analysis results in $Z' = p(1 - p^{w})/(1 - p)$ and $Z'' = p(1 - p^{(L-w)})/(1 - p)$ for B. If $p = 0.5$, then $Z' = 1 - 2^{-w}$ and $Z'' = 1 - 2^{(w-L)}$. If w is significant, then $Z'' \approx Z' \approx 1$ for B. For simplicity, let Z_B represent Z' and Z'' for B.

The expected value of $Z(T)$ for all possible T, denoted by Z_T, is analyzed as follows. The essence of the transform algorithm is represented by the automaton in Figure 1. The automaton inputs a sequence of bits$\{0, 1\}$ of B in LSB first order, and outputs $\{\overline{1}, 0, 1\}^*$.

In Figure 1, each arc is labeled by an input digit $b \in \{0, 1\}$. All output digits are determined by one of the following two functions.

$$f(b) = \begin{cases} b-1 & \text{if } s_3 < s_0, \\ b & \text{otherwise}, \end{cases} \qquad g(b) = \begin{cases} \bar{1} & \text{if } s_3 < s_0, \\ 1 & \text{otherwise}, \end{cases}$$

where the condition denoted by $s_3 < s_0$ means that s_3 is visited before s_0 by forthcoming transition. Solid arc corresponds to $f(b)$, and dotted arc corresponds to $g(b)$.
Assume input (*i.e.* B) comes from a memoryless binary information source with $p = 0.5$. Let z_i be an average length of zero runs at state s_i. Each value of z_i is obtained by solving the following equations.

$$\begin{cases} z_0 = (1/2)(1+z_0), \\ z_1 = (1/2)(1+z_0) + (1/2)Prob(s_3 < s_0|s_2)(1+z_2), \\ z_2 = (1/2)(1+z_3) + (1/2)Prob(s_0 < s_3|s_1)(1+z_1), \\ z_3 = (1/2)(1+z_3), \\ z_4 = (1/2)(1+z_3) + (1/2)Prob(s_0 < s_3|s_5)(1+z_5), \\ z_5 = (1/2)(1+z_0) + (1/2)Prob(s_3 < s_0|s_4)(1+z_4), \end{cases}$$

where $Prob(s_i < s_j|s_k)$ means the probability of the case of $(s_i < s_j)$ from state s_k. In the above equations, $Prob(s_3 < s_0|s_2) = Prob(s_0 < s_3|s_1) = Prob(s_0 < s_3|s_5) = Prob(s_3 < s_0|s_4) = (1/2) + (1/2)(1/4) + (1/2)(1/4^2) + \cdots \approx 2/3$. Thus, $z_0 = 1,\ z_1 = 2,\ z_2 = 2,\ z_3 = 1,\ z_4 = 2,\ z_5 = 2$.
Let p_i be a stationary probability of state s_i. All p_i are calculated by solving the equation $V = M \cdot V$ where V is the vector of all p_i and M is the given transition matrix. The result is $p_0 = 1/4,\ p_1 = 1/6,\ p_2 = 1/12,\ p_3 = 1/4,\ p_4 = 1/6,\ p_5 = 1/12$. Therefore, $Z_T = \sum_{i=0}^{5} p_i \cdot z_i = 3/2$ for the proposed method. Note that, $Z_T = 4/3$ for MO method and JM method.
In summary, using $\overline{L} = \lambda + 5/4$ and $Z_T = 3/2$, the average number of operations R for T, or R_T, is rewritten as:

$$R_T = (\lambda + \frac{11}{4} - w) + \frac{\lambda + \frac{5}{4}}{w + \frac{3}{2}} + 2^{w-1} - 1$$

The optimal value of window size w depends on the size of d. It is obtained by solving $\frac{\partial}{\partial w} R_T = 0$. For d with $\lambda = 511$, $w = 5$ is the optimal window size.

3 Comparison with other methods

Brickell [2] proposed a fast hardware implementation of computing $x^d \bmod n$ using the precomputation of the multiplicative inverse $x^{-1} \bmod n$.
Morain and Olivos [14] proposed an addition-subtraction chain algorithm based on a binary method. Their method obtains d_+, and d_- for $d(d = d_+ - d_-)$, and computes $d \cdot P$ as $(d_+ \cdot P) - (d_- \cdot P)$. In MO method, $d_+ \cdot P$ (and $d_- \cdot P$) are computed using the ordinary binary method. The average number of operations for MO method is $\frac{4}{3}\lambda + O(1)$.

Yacobi [16] applied the idea of data compression (Lempel-Ziv's incremental parsing algorithm) to splitting binary representation. The average number of operations in Yacobi's method is $\lambda + (\log(\lambda) - \log\log(\lambda))/2 + 1.5\lambda/\log(\lambda)$, where $\lambda = \lfloor \log_2 d \rfloor$. In his method, the segment size is initially small, and increases by parsing B. This method is inefficient for small d such as $\lambda = 511$

Bos and Coster [1] proposed a heuristics for an addition sequence and used a bigger window such as $w = 10$. This method requires an average of 605 operations for $\lambda = 511$. However, their method is based on heuristics.

A comparison of several addition(-subtraction) chain algorithms is shown in Table.1. From Table.1, the proposed method is seen to be faster than the other methods.

Table 1. The number of operations for d of 512 bit length

Method	Chain	Av.	Worst
Binary Method [10]	A	766.5	1022
Signed Binary Method [2] [8] [14]	A/S	681.7	768
Yacobi's Method [16]	A	635.1	—
Window Method ($w = 5$) [1]	A	609.3	630
Bos-Coster's Method [1]	A	605	—
Signed Binary Window Method ($w = 5$)	A/S	602.6	629

4 The Speed of Each Addition over Elliptic Curves

4.1 Elliptic Curves over a Finite Field and a Ring

Let K be a field of characteristic $\neq 2, 3$, and let $a, b \in K$ be two parameters satisfying $4a^3 + 27b^2 \neq 0$. An elliptic curve over K with parameters a and b is defined as the set of points (x, y) with $x, y \in K$ satisfying this equation on the affine plane

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

together with a special element denoted O and called the point at infinity [11]. Elliptic curves over the finite field $\mathbf{F}_p$ with p elements, for some prime p, are denoted by E_p. What makes elliptic curves interesting in cryptography and number-theoretic applications is the fact that an *addition operation* on the points of an elliptic curve E_p can be defined to make it an abelian group.

Elliptic curves over the ring $\mathbf{Z}_n$, where n is an odd composite square-free integer, can be defined in a similar way to E_p. For simplicity, let n be the product of two distinct large primes p and q as in the RSA scheme[15] and the KMOV scheme[9]. Addition on E_n, whenever it is defined, is equivalent to the group operation (defined by component) on $E_p \times E_q$. Thus, every point $P = (x, y)$ on E_n can be represented uniquely as a pair $[P_p, P_q] = [(x_p, y_p), (x_q, y_q)]$ where $P_p \in E_p$ and $P_q \in E_q$. Note that addition on E_n is undefined if and only if exactly one of the points P_p and P_q is the point at infinity. It is important to note that when all prime factors of n are large, it is extremely unlikely that the sum of two points on E_n is undefined.

4.2 Addition Formulae over Elliptic Curves

Let $P_1 = (x_1, y_1)$ and $P_2 = (x_2, y_2)$ be two points on the elliptic curve E_p. The point $P_3 = P_1 + P_2 = (x_3, y_3)$ is defined according to the following rules. If $P_1 = \mathcal{O}$, then $P_3 = P_1 + P_2 = P_2$. If $P_1 = -P_2$, that is, $x_1 = x_2$ and $y_1 = -y_2$, then $P_3 = P_1 + P_2 = \mathcal{O}$. When $P_1, P_2 \neq \mathcal{O}$, and $P_1 \neq -P_2$, an *addition formula* to find $P_3 = P_1 + P_2 = (x_3, y_3)$ is given below according to two cases: a non-doubling addition formula where $P_1 \neq P_2$ and a doubling formula $P_1 = P_2$ [11].

Non-doubling Addition Formula in Affine Coordinates

$$\begin{cases} x_3 = \lambda^2 - x_1 - x_2 \\ y_3 = \lambda(x_1 - x_3) - y_1, \end{cases} \tag{1}$$

where $\lambda = (y_2 - y_1)/(x_2 - x_1)$.

Doubling Formula in Affine Coordinates

$$\begin{cases} x_3 = \lambda^2 - 2x_1 \\ y_3 = \lambda(x_1 - x_3) - y_1, \end{cases} \tag{2}$$

where $\lambda = (3x_1^2 + a)/2y_1$.

Note that a subtraction to find $P_3 = P_1 - P_2$ is defined by changing the sign of y_2 in the addition formula $P_3 = P_1 + P_2$.

A point (x, y) on the affine plane is equivalent to a point (X, Y, Z) on the projective plane, where $x = X/Z$, $y = Y/Z$. That is, an elliptic curve is also defined as the set of points (X, Y, Z) in homogeneous coordinates satisfying the equation

$$ZY^2 = X^3 + aXZ^2 + bZ^3,$$

together with the point at infinity (0,1,0). The non-doubling addition formula (1) and the doubling formula (2) in affine coordinates can be rewritten in homogeneous coordinates. Replace x_i with X_i/Z_i and y_i with Y_i/Z_i $(i = 1, 2)$ and reduce the fractions of x_3 and y_3 to a common denominator. Then, the resulting numerators of x_3 and y_3 become X_3 and Y_3, and the common denominator becomes Z_3. Let $P_1 = (X_1, Y_1, Z_1) \in E_p$, $P_2 = (X_2, Y_2, Z_2) \in E_p$, and $P_3 = (X_3, Y_3, Z_3) \in E_p$. The addition formulae to find $P_3 = P_1 + P_2$ in homogeneous coordinates are expressed as follows.

Non-doubling Addition Formula in Homogeneous Coordinates

$$\begin{cases} X_3 = X_1^4Z_2^4 - 2X_1^3X_2Z_1Z_2^3 + 2X_1X_2^3Z_1^3Z_2 - X_1Y_1^2Z_1Z_2^4 + 2X_1Y_1Y_2Z_1^2Z_2^3 \\ \quad -X_1Y_2^2Z_1^3Z_2^2 - X_2^4Z_1^4 + X_2Y_1^2Z_1^2Z_2^3 - 2X_2Y_1Y_2Z_1^3Z_2^2 + X_2Y_2^2Z_1^4Z_2, \\ Y_3 = X_2^3Y_2Z_1^4 - 2X_2^3Y_1Z_1^3Z_2 - Y_2^3Z_1^4Z_2 + 3X_1X_2^2Y_1Z_1^2Z_2^2 - 3X_1^2X_2Y_2Z_1^2Z_2^2 + \\ \quad 3Y_1Y_2^2Z_1^3Z_2^2 + 2X_1^3Y_2Z_1Z_2^3 - 3Y_1^2Y_2Z_1^2Z_2^3 - X_1^3Y_1Z_2^4 + Y_1^3Z_1Z_2^4, \\ Z_3 = -X_1^3Z_1Z_2^4 + 3X_1^2X_2Z_1^2Z_2^3 - 3X_1X_2^2Z_1^3Z_2^2 + X_2^3Z_1^4Z_2. \end{cases} \tag{3}$$

Doubling formula in Homogeneous Coordinates

$$\begin{cases} X_3 = 2Y_1Z_1(a^2Z_1^4 + 6aX_1^2Z_1^2 + 9X_1^4 - 8X_1Y_1^2Z_1), \\ Y_3 = -a^3Z_1^6 - 9a^2X_1^2Z_1^4 - 27aX_1^4Z_1^2 + 12aX_1Y_1^2Z_1^3 - 27X_1^6 + 36X_1^3Y_1^2Z_1 - 8Y_1^4Z_1^2, \\ Z_3 = 8Y_1^3Z_1^3. \end{cases} \tag{4}$$

By introducing moderate intermediate variables that are more moderate than ones in [9], addition formulae (3) and (4) can be revised to minimize the number of multiplications in serial processing. The revised addition formulae in homogeneous coordinates are:

Revised Non-doubling Addition Formula in Homogeneous Coordinates

$$\begin{cases} X_3 = VA \\ Y_3 = U(V^2X_1Z_2 - A) - V^3Y_1Z_2, \\ Z_3 = V^3Z_1Z_2, \end{cases} \tag{5}$$

where $U = Y_2Z_1 - Y_1Z_2$, $V = X_2Z_1 - X_1Z_2$, $A = U^2Z_1Z_2 - V^2T$, $T = X_2Z_1 + X_1Z_2$.

Revised Doubling Formula in Homogeneous Coordinates

$$\begin{cases} X_3 = 2SH, \\ Y_3 = W(4F - H) - 8E^2, \\ Z_3 = 8S^3, \end{cases} \tag{6}$$

where $S = Y_1Z_1$, $W = 3X_1^2 + aZ_1^2$, $E = Y_1S$, $F = X_1E$, $H = W^2 - 8F$.

Note that all of the above computations are modulo p or modulo n.

4.3 Performance Evaluation of Addition Formulae

Computations of the multiples of a point on the elliptic curves E_n can be performed in affine coordinates or homogeneous coordinates. The time complexity of the addition formulae implemented in these coordinates was compared. Each elementary addition over E_n was calculated using addition, subtraction, multiplication and division in $\mathbf{Z}_n$. For simplicity, addition, subtraction and special multiplication by a small constant such as $2y_1$ and $3(x_1^2)$ ware neglected because they are much faster than multiplication and division in $\mathbf{Z}_n$. In addition formulae (3)-(4) and (5)-(6) in homogeneous coordinates, contrary to addition formulae (1)-(2) in affine coordinates, the divisions in $\mathbf{Z}_n$ in each addition over E_n can be avoided. Computation in homogeneous coordinates requires 1 division ($1/Z_3$) in $\mathbf{Z}_n$ to obtain both $x_3 = X_3/Z_3$ and $y_3 = Y_3/Z_3$ in the final stage of the chain. Note that division in $\mathbf{Z}_n$ can be implemented using the generalized Euclidean algorithm for computing the greatest common divisor.

A serial computation of non-doubling addition formula (1) requires two multiplications and one division in $\mathbf{Z}_n$. A serial computation of doubling formula (2) requires three multiplications and one division in $\mathbf{Z}_n$. A serial computation of non-doubling addition formula (5) required 15 multiplications in $\mathbf{Z}_n$. For the KMOV elliptic cryptosystem with $a = 0$, the computation of W in the doubling formula (6) can be simplified as $W = 3X_1^2$. Thus, a serial computation of doubling formula (6) requires 10 multiplications in $\mathbf{Z}_n$.

Assume that parallel processing of each addition over E_n is allowed in special hardware. For simplicity, the time for communication among processors is neglected. In affine coordinates, parallel processings of non-doubling addition and doubling require the same computational complexity as those in serial processing. Consider parallel processing of the addition formula in homogeneous coordinates. In general, parallel multiplication permits any polynomial of degree $2k$ to be computed in one step from the set of polynomials of degree k, where each step requires the time of one multiplication in $\mathbf{Z}_n$. The non-doubling addition formula (3) consists of polynomials of degree 8 with 6 variables, therefore, the values of X_3, Y_3, Z_3 can be obtained in 3 steps. The doubling formula (4) consists of polynomials of degree 6 with 4 variables (including a), therefore, the values of X_3, Y_3, Z_3 can be similarly obtained in 3 steps. That is, the related terms of degree 2 are computed in the first step, the related terms of degree 4 in the second step, and every term of degree 8 or 6 in the target polynomials in the third step.

Denote c be the ratio of the computation amount of division in $\mathbf{Z}_n$ to that of multiplication in $\mathbf{Z}_n$. Note that $c > 1$. Let R be the number of operations of addition formula in addition-subtraction chain. Assume that non-doubling additions occur with probability p_n and doublings occur with probability $(1 - p_n)$. For the proposed signed binary window method, we have $p_n \approx 1/6$ and $R \approx 602.6$ for $\lambda = 511$ as described in Section 2. Table 2 shows the numbers of multiplications in $\mathbf{Z}_n$ in serial/parallel processings in affine coordinates and homogeneous coordinates. From Table 2, we can observe that serial computation in homogeneous coordinates is faster than that in affine coordinates if $c > 8$

and $\lambda = 511$. Moreover, when $\lambda = 511$, parallel computation in homogeneous coordinates is always faster than that in affine coordinates.

Table 2. The number of mult./div. in $\mathbf{Z}_n$ in the total chain

Processing	Coordinates	mult.	div.
Serial	Affine	$(2p + 3(1-p))R$ $\approx 2.83 \cdot R$	R
	Homogeneous	$(15p + 10(1-p))R$ $\approx 10.83 \cdot R$	1
Parallel	Homogeneous	$3 \cdot R$	1

When the multiplication chain is carried out based on alphabetically ordered factoring in formula (3), 17 processors are needed in the first step, 29 processors in the second step, and 24 processors in the third step. Since each processor can be used repeatedly, this multiplication system (or addition formula engine) requires 29 processors. Note that parallel computation of formula (4) requires less than 29 processors. As a result, each addition over the elliptic curve can be done in 3 multiplications if 29 parallel processors are used.

5 Conclusion

We have proposed a fast and systematic method of computing a multiple $d \cdot P$ over elliptic curves. This speeding up method is also applicable to computation in the group where the inverse operation is as fast as an ordinary operation. Furthermore, we pointed out that if parallel processing is allowed, each addition over the curve using homogeneous coordinates can be done in 3 multiplications.

References

1. Bos, J. and Coster, M: "Addition chain heuristics" *Proc. of CRYPTO'89* (1989).
2. Brickell, E.F.:"A fast modular multiplication algorithm with application to two key cryptography" *Proc. of CRYPTO'82* (1982).
3. Brickell, E.F., Gordon, D.M., McCurley, K.S., and Wilson, D.: "Fast exponentiation with precomputation " *Proc. of EUROCRYPT'92* (1992).
4. Diffie, W. and Hellman, M.E.: "New directions in cryptography", *IEEE Transactions on Information Theory*, Vol. 22, No. 6, (1976), pp. 644–654.
5. Downey, P. Leony, B. and Sethi, R:"Computing sequences with addition chains", *Siam J. Comput. 3* (1981) pp. 638–696.
6. ElGamal, T.:"A public key cryptosystem and a signature scheme based on the discrete logarithm", *IEEE Transactions on Information Theory*, Vol. 31, No. 4, (1985), pp. 469–472.

7. Goldwasser, S. and Killian, J.:"Almost all primes can be quickly certified", *Proc. 18th STOC*. Berkeley, (1986), pp. 316–329.
8. Jedwab, J. and Mitchell, C, J.:"Minimum weight modified signed-digit representations and fast exponentiation", *Electronics Letters* Vol. 25, No. 17, (1989), pp. 1171–1172.
9. Koyama, K. Maurer, U. Okamoto, T and Vanstone, S, A: "New public-key schemes based on elliptic curves over the ring $\mathbf{Z}_n$", *Proc. of CRYPTO'91* (1991).
10. Knuth, D.E.: "Seminumerical algorithm (arithmetic)" The Art of Computer Programming Vol.2, Addison Wesley, (1969).
11. Koblitz, N.:*A course in number theory and cryptography*, Berlin: Springer-Verlag, (1987).
12. Lenstra, H. W. Jr.: "Factoring integers with elliptic curves", *Ann. of Math. 126* (1987), pp. 649–673.
13. Montgomery, P.L.:"Speeding the Pollard and elliptic curve methods of factorization", Math. Comp. 48, (1987), pp. 243–264.
14. Morain, F. and Olivos, J.: "Speeding up the computations on an elliptic curve using addition-subtraction chains" Theoretical Informatics and Applications Vol. 24, No. 6 (1990) pp. 531–544.
15. Rivest, R.L. Shamir, A. and Adleman, L.:"A method for obtaining digital signatures and public-key cryptosystems", *Communications of the ACM*, Vol. 21, No. 2, (1978), pp. 120–126.
16. Yacobi, Y.: "Exponentiating faster with addition chains" *Proc. of EURO-CRYPT'90* (1990).

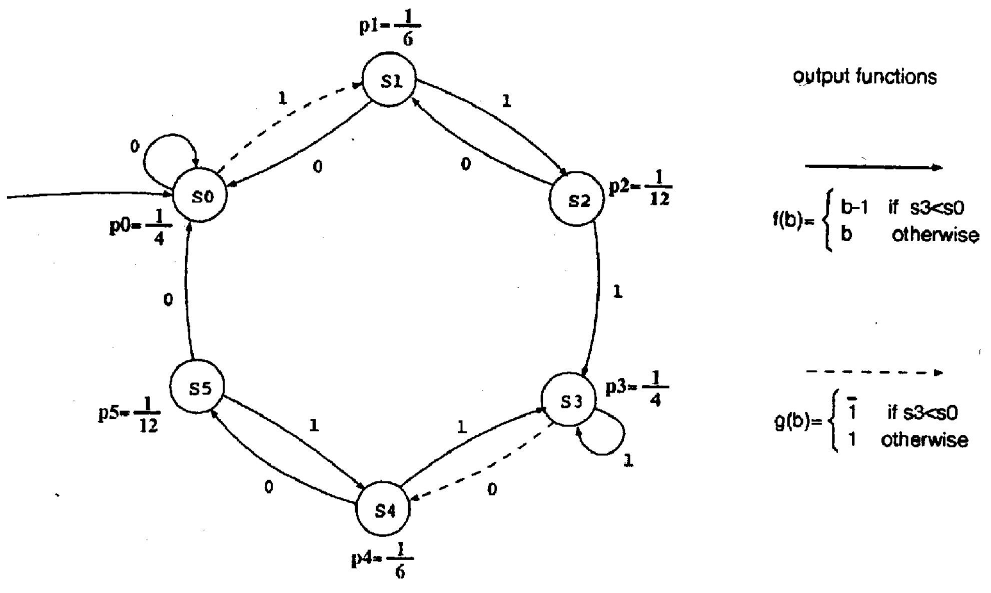

Figure.1 Automaton for the transform algorithm