FAST ELLIPTIC CURVE CRYPTOGRAPHY USING OPTIMAL DOUBLE-BASE CHAINS

Vorapong Suppakitpaisarn, Hiroshi Imai
Graduate School of Information
Science and Technology,
The University of Tokyo
Tokyo, Japan 113-0022
mr_t_dtone@is.s.u-tokyo.ac.jp
imai@is.s.u-tokyo.ac.jp

Masato Edahiro
Graduate School of
Information Science
Nagoya University
Nagoya, Japan 464-8601
eda@ertl.jp

ABSTRACT

In this work, we propose an algorithm to produce the double-base chain that optimizes the time used for computing an elliptic curve scalar multiplication, i.e. the bottleneck operation of the elliptic curve cryptosystem. The double-base number system and its subclass, double-base chain, are the representation that combines the binary and ternary representations. The time is measured as the weighted sum in terms of the point double, triple, and addition, as used in evaluating the performance of existing greedy-type algorithms, and our algorithm is the first to attain the minimum time by means of dynamic programming. Compared with greedy-type algorithm, the experiments show that our algorithm reduces the time for computing the scalar multiplication by 3.88-3.95% with almost the same average running time for the method itself. The proposed algorithm is also better than the general algorithm for the double-base number system using Yao's algorithm when the point triple is comparatively fast to the point addition.

KEYWORDS

Internet Security, Cryptography, Elliptic Curve Cryptography, Minimal Weight Conversion, Digit Set Expansion, Double-Base Number System, Double-Base Chain

1. INTRODUCTION

Scalar multiplication is the bottleneck operation of the elliptic curve cryptography. It is to compute

$$Q = rS$$

when S, Q are points on the elliptic curve and r is a positive integer. The computation time of the operation strongly depends on the representation of r. The most common way to represent r is to use the binary expansion,

$$r = \sum_{t=0}^{n-1} r_t 2^t,$$

where r_t is a member of a finite *digit set* D_S. We call

$$R = \langle r_0, \ldots, r_{n-1} \rangle$$

as the *binary expansion* of r. If $D_S = \{0, 1\}$, we can represent each integer r by a unique binary expansion. However, we can represent some integers by more than one binary expansion if $\{0, 1\} \subsetneq D_S$. For example, $r = 15 = 2^0 + 2^1 + 2^2 + 2^3 = -2^0 + 2^4$ can be represented by $R_1 = \langle 1, 1, 1, 1, 0 \rangle$, $R_2 = \langle -1, 0, 0, 0, 1 \rangle$, and many other ways. Shown in Section 2, the computation of scalar multiplication based on the binary expansion R_2 makes the operation faster than using the binary expansion R_1. The algorithm to find the optimal binary expansion of each integer has been studied extensively in many works [1], [2].

The representation of r is not limited only to binary expansion. Takagi et al. [3] have studied about the representation in a larger radix, and discuss about its application in pairing-based cryptosystem. The efficiency of representing the number by ternary expansion is discussed in the paper.

Some numbers have better efficiency in binary expansion, and some are better in ternary expansion. Then, it is believed that double-base number system (DBNS) [4], [5] can improve the efficiency of the scalar multiplication. The double-base number system of r is defined by $C[r] = \langle R, X, Y \rangle$ when $R = \langle r_0, \ldots, r_{m-1} \rangle$ for $r_i \in D_S - \{0\}, X = \langle x_0, \ldots, x_{m-1} \rangle, Y = \langle y_0, \ldots, y_{m-1} \rangle$ for $x_i, y_i \in \mathbb{Z}$, and

$$r = \sum_{t=0}^{m-1} r_t 2^{x_t} 3^{y_t}.$$

The representation of each integer in double-base number system is not unique. For example, $14 = 2^3 3^0 + 2^1 3^1 = 2^1 3^0 + 2^2 3^1$ can be represented as $C_1[14] = \langle \langle 1, 1 \rangle, \langle 3, 1 \rangle, \langle 0, 1 \rangle \rangle$, and $C_2[14] = \langle \langle 1, 1 \rangle, \langle 1, 2 \rangle, \langle 0, 1 \rangle \rangle$.

Meloni and Hasan [6] used double-base number system with Yao's algorithm to improve the computation time of scalar multiplication. However, the implementation is complicated to analyze, and it needs more memory to store many points on the elliptic curve. In other words, the implementation of scalar multiplication based on double-base number system is difficult. To cope with the problem, Dimitrov et al. proposed to used double-base chains, double-base number system with more restrictions. Double-base chains $C[r] = \langle \langle r_t \rangle_{t=0}^{m-1}, \langle x_t \rangle_{t=0}^{m-1}, \langle y_t \rangle_{t=0}^{m-1} \rangle$ is similar to double-base number system, but double-base chains require x_i and y_i to be monotone, i.e. $x_0 \leq \cdots \leq x_{m-1}, y_0 \leq \cdots \leq y_{m-1}$. Concerning $C_1[14]$, $C_2[14]$ on the previous paragraph, $C_1[14]$ is not a double-base chain, while $C_2[14]$ is.

Like binary expansion and double-base number system, some integers have more than one double-base chains, and the efficiency of elliptic curve cryptography strongly depends on which chain we use. The algorithm to select efficient double-base chains is very important. The problem has been studied in the literature [7], [8], [9]. However, they proposed greedy algorithms which cannot guarantee the optimal chain. On the other hand, we adapted our previous works [10], where the dynamic programming algorithm is devised to find the optimal binary expansion. The paper presents efficient dynamic programming algorithms which output the chains with optimal cost. Given the cost of elementary operations formulated as in [7], [11], we find the best combination of these elementary operations in the framework of double-base chains. By the experiment, we show that the optimal double-base chains are better than the best greedy algorithm proposed on double-base chain [9] by 3.9% when $D_S = \{0, \pm 1\}$. The experimental results show that the average results of the algorithm are better than the algorithm using double-base number system with Yao's algorithm [6] in the case when point triples is comparatively fast to point additions.

Even though our algorithm is complicated than the greedy-type algorithm, both algorithms have the same time complexity, $O(\lg^2 n)$. Also, the average running time of our method for 448-bit inputs is 30ms when we implement the algorithm using Java in Windows Vista, AMD Athlon(tm) 64X2 Dual Core Processor 4600+ 2.40GHz, while the average running time of the algorithm in [7] implemented in the same computation environment is 29ms. The difference between the average running time of our algorithm and the existing one is negligible, as the average computation time of scalar multiplication in Java is shown to be between 400-650ms [12].

In 2010, Imbert and Philippe [13] proposed an algorithm which can output the shortest chains when $D_S = \{0, 1\}$. Their works can be considered as a specific case of our works as our algorithm can be applied to any finite digit sets. Adjusting the parameters of our algorithm, we can also output the shortest chains for double-base chains.

The paper is organized as follows: we describe the double-and-add scheme, and how we utilize the double-base chain to elliptic curve cryptography in Section 2. In Section 3, we show our algorithm which outputs the optimal double-base chain. Next, we present the experimental results comparing to the existing works in Section 4. Last, we conclude the paper in Section 5.

2. COMPUTATION TIME FOR DOUBLE-BASE CHAINS

Using the binary expansion $R = \langle r_t \rangle_{t=0}^{n-1}$, where $r = \sum_{t=0}^{n-1} r_t 2^t$ explained in Section 1, we compute the scalar multiplication $Q = rS$ by double-and-add scheme as shown in Algorithm 1. For example, we compute $Q = 127S$ when the binary expansion of 127 is $R = \langle 1, 1, 1, 1, 1, 1, 1 \rangle$ as follows:

$$Q = 2(2(2(2(2(2S + S) + S) + S) + S) + S) + S.$$

Above, we need two elementary operations, that are point doubles ($S + S$, $2S$) and point additions ($S + Q$ when $S \neq Q$). These two operations look similar, but they are computationally different in many cases. In this example, we need six point doubles and six point additions. Generally, we need n point doubles and n point additions. However, Q is initialized to O, and we need not the point addition on the first iteration. Also, $r_t S = 0$ if $r_t = 0$, and we need not the point addition in this case. Hence, the number of the point addition is $W(R) - 1$, where $W(R)$ is the Hamming weight of the expansion defined as:

$$W(R) = \sum_{t=0}^{n-1} W(r_t),$$

where $W(r_t) = 0$ when $r_t = 0$ and $W(r_t) = 1$ otherwise. In this case, $W(R) = 7$.

The Hamming weight tends to be less if the digit set D_S is larger. However, using big D_S makes cost for the precomputation higher as we need to precompute $r_t S$ for all $r_t \in D_S$.

Algorithm 1: Double-and-add method

```
input : A point on elliptic curve $S$, the
        positive integer $r$ with the binary
        expansion $\langle r_0, \ldots, r_{n-1} \rangle$
output: $Q = rS$

1 $Q \leftarrow O$
2 for $t \leftarrow n - 1$ to $0$ do
3     $Q \leftarrow Q + r_t S$
4     if $t \neq 0$ then $Q \leftarrow 2Q$
5 end
```

In Algorithm 2, we show how to apply the double-base chain $C[r] = \langle R, X, Y \rangle$, when $R = \langle r_t \rangle_{t=0}^{m-1}$, $X = \langle x_t \rangle_{t=0}^{m-1}$, $Y = \langle y_t \rangle_{t=0}^{m-1}$ to compute scalar multiplication. For example, one of the double-base chain of $127 = 2^0 3^0 + 2^1 3^2 + 2^2 3^3$ is $C[127] = \langle R, X, Y \rangle$, where $R = \langle 1, 1, 1 \rangle$, $X = \langle 0, 1, 2 \rangle$, $Y = \langle 0, 1, 3 \rangle$. Hence, we can compute $Q = 127S$ as follows:

$$Q = 2^1 3^2 (2^1 3^1 S + S) + S.$$

In addition to point doubles and point additions required in the binary expansion, we also need point point triples ($3S$).

In this example, we need two point additions, two point doubles, and three point triples. In general, the number of point additions is $W(C) - 1$

when $W(C) = m$ is the number of terms in the chain C. On the other hand, the number of point doubles and point triples are x_{m-1} and y_{m-1} respectively.

Algorithm 2: Using the double-base chain to compute scalar multiplication

```
input : A point on elliptic curve $S$, the
        positive integer $r$ with the
        double-base chains
        $C[r] = \langle R, X, Y \rangle$, where
        $R = \langle r_t \rangle_{t=0}^{m-1}$, $X = \langle x_t \rangle_{t=0}^{m-1}$,
        $Y = \langle y_t \rangle_{t=0}^{m-1}$
output: $Q = rS$

1 $Q \leftarrow O$
2 for $t \leftarrow m-1$ to $0$ do
3     $Q \leftarrow Q + r_t S$
4     if $t \neq 0$ then $Q \leftarrow 2^{(x_{t-1}-x_t)} 3^{(y_{t-1}-y_t)} Q$
5     else $Q \leftarrow 2^{x_0} 3^{y_0} Q$
6 end
```

In the double-and-add method, the number of point doubles required is proved to be constantly equal to $n - 1 = \lfloor \lg r \rfloor - 1$. Then, the efficiency of the binary expansion strongly depends on the number of point additions or the Hamming weight. In double-base chain, the number of point doubles and point triples are not constant. Hence, we need to optimize the value

$$x_{m-1} \cdot P_{dbl} + y_{m-1} \cdot P_{tpl} + (W(C[r]) - 1) \cdot P_{add},$$

when $P_{dbl}, P_{tpl}, P_{add}$ are the cost for point double, point triple, and point addition respectively. This is difference from the literature [13] where only the Hamming weight is considered.

3. ALGORITHM FOR OPTIMAL DOUBLE-BASE CHAINS

3.1. Algorithm for Digit Set {0,1}

Define the cost to compute r using the chain $C[r] = \langle R, X, Y \rangle$ as $P(C[r]) = x_{m-1} \cdot P_{dbl} + y_{m-1} \cdot P_{tpl} + (W(C[r]) - 1) \cdot P_{add}$, when $C[r] \neq \langle \langle\rangle, \langle\rangle, \langle\rangle \rangle$, and $P(C[r]) = 0$ otherwise. Our algorithm is to find the double-base chain of r, $C[r] = \langle R, X, Y \rangle$ such that for all double-base chain of r, $Ce[r] = \langle Re, Xe, Ye \rangle$, $P(Ce[r]) \geq P(C[r])$. To explain the algorithm, we start with a small example explained in Example 1 and Figure 1.

Example 1 Find the optimal chain $C[7]$ given $D_S = \{0, 1\}$, $P_{tpl} = 1.5$, $P_{dbl} = 1$, $P_{add} = 1$.

Assume that we are given the optimal chain $C[3] = \langle R[3], X[3], Y[3] \rangle$ ($3 = \lfloor \frac{7}{2} \rfloor$) and $C[2] = \langle R[2], X[2], Y[2] \rangle$ ($2 = \lfloor \frac{7}{3} \rfloor$). We want to express 7 in term of $\sum_{t=0}^{m-1} r_t 2^{x_t} 3^{y_t}$, when $r_t \in D_S - \{0\} = \{1\}$. As $2 \nmid 7$ and $3 \nmid 7$, the smallest term much be $1 = 2^0 3^0$. Hence, $x_0 = 0$ and $y_0 = 0$. Then, $7 = \sum_{t=1}^{m-1} 2^{x_t} 3^{y_t} + 1$. By this equation, there are only two ways to compute the scalar multiplication $Q = 7S$ with Algorithm 2. The first way is to compute $3S$, do point double to $6S$ and point addition to $7S$. As we know the the optimal chain for 3, the cost using this way is $P(C[3]) + P_{dbl} + P_{add}$. The other way is to compute $2S$, do point triple to $6S$ and point addition to $7S$. In this case, the cost is $P(C[2]) + P_{tpl} + P_{add}$. The optimal way is to select one of these two ways. We will show later that $P(C[3]) = 1.5$ and $P(C[2]) = 1$. Then,

$$P(C[3]) + P_{dbl} + P_{add} = 1.5 + 1 + 1 = 3.5,$$

$$P(C[2]) + P_{tpl} + P_{add} = 1 + 1.5 + 1 = 3.5.$$

Both of them have the same amount of computation time $P(C[7]) = 3.5$, and we can choose any of them. Suppose that we select the first choice, and $C[3] = \langle R[3], \langle x[3]_t \rangle_{t=0}^{m-2}, \langle y[3]_t \rangle_{t=0}^{m-2} \rangle$. The optimal double-base chain of 7 is $C[7] = \langle R, X, Y \rangle$ when

$$R = \langle 1, R[3] \rangle.$$

$$X = \langle x_0, \ldots, x_{m-1} \rangle,$$

where $x_0 = 0$ and $x_t = x[3]_{t-1} + 1$ for $1 \leq t \leq m-1$.

$$Y = \langle y_0, \ldots, y_{m-1} \rangle,$$

where $y_0 = 0$ and $y_t = y[3]_{t-1}$ for $1 \leq t \leq m-1$.

Next, we find $C[3]$ that is the optimal double-base chain of $\left\lfloor \frac{7}{2} \right\rfloor = 3$. Similar to $7S$, we can compute $3S$ by two ways. The first way is to triple the point S. Using this way, we need one point triple, which costs $P_{tpl} = 1.5$. The double-base chain in this case will be

$$\langle \langle 1 \rangle, \langle 0 \rangle, \langle 1 \rangle \rangle.$$

The other way is that we double point S to $2S$, then add $2S$ with S to get $3S$. The cost is $P_{dbl} + P_{add} = 1 + 1 = 2$. In this case, the double-base chain is

$$\langle \langle 1, 1 \rangle, \langle 0, 1 \rangle, \langle 0, 0 \rangle \rangle.$$

We select the better double-base chain that is

$$C[3] = \langle \langle 1 \rangle, \langle 0 \rangle, \langle 1 \rangle \rangle.$$

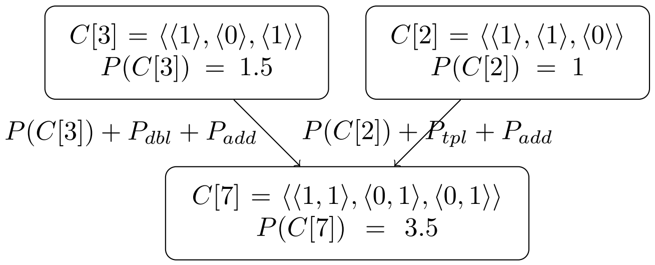
Last, we find $C[2]$, the optimal double-base chain of $\left\lfloor \frac{7}{3} \right\rfloor = 2$. The interesting point to note is that there are only one choice to consider in this case. This is because the fact that we cannot rewrite 2 by $3A + B$ when $A \in \mathbb{Z}$ and $B \in D_S$ if $r \equiv 2 \bmod 3$. Then, the only choice left is to double the point S, which costs 1, and the double-base chain is

$$C[2] = \langle \langle 1 \rangle, \langle 1 \rangle, \langle 0 \rangle \rangle.$$

To conclude, the optimal double-base chain for 7 in this case is

$$C[7] = \langle \langle 1, 1 \rangle, \langle 0, 1 \rangle, \langle 0, 1 \rangle \rangle.$$

In Example 1, we consider the computation as a top-down algorithm. However, bottom-up algorithm is a better way to implement the idea. We begin the algorithm by computing the double-base chain of $\left\lfloor \frac{r}{2^x 3^y} \right\rfloor$ for all $x, y \in \mathbb{Z}^+$ such that $x + y = q$ where $2^q \leq r < 2^{q+1}$. Then, we move



$C[3] = \langle\langle 1 \rangle, \langle 0 \rangle, \langle 1 \rangle\rangle$
$P(C[3]) = 1.5$
$C[2] = \langle\langle 1 \rangle, \langle 1 \rangle, \langle 0 \rangle\rangle$
$P(C[2]) = 1$
$P(C[3]) + P_{dbl} + P_{add}$
$P(C[2]) + P_{tpl} + P_{add}$
$C[7] = \langle\langle 1, 1 \rangle, \langle 0, 1 \rangle, \langle 0, 1 \rangle\rangle$
$P(C[7]) = 3.5$

Figure 1. We can compute $C[7]$ by two ways. The first way is to compute $C[3]$, and perform a point double and a point addition. The cost in this way is $P(C[3]) + P_{dbl} + P_{add}$. The second way is to compute $C[2]$, and perform a point triple and a point addition, where the cost is $P(C[2]) + P_{tpl} + P_{add}$. The amount of computation time in both ways are similar, and we can choose any of them.

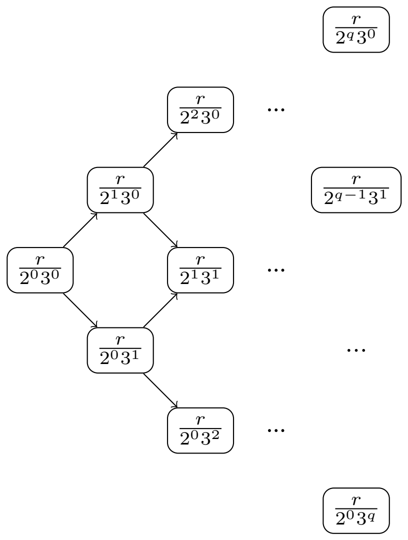


Figure 2. Bottom-up algorithm to find the optimal double-base chain of r

to compute the double-base chain of $\left\lfloor \frac{r}{2^x 3^y} \right\rfloor$ for all $x, y \in \mathbb{Z}^+$ such that $x + y = q - 1$ by referring to the double-base chain of $\left\lfloor \frac{r}{2^x 3^y} \right\rfloor$ when $x + y = q$. We decrease the number $x + y$ until $x + y = 0$, and we get the chain of $r = \left\lfloor \frac{r}{2^0 3^0} \right\rfloor$. We illustrate this idea in Figure 2 and Example 2.

Example 2 Find the optimal double-base chain of 10 when

$$P_{add} = P_{dbl} = P_{tpl} = 1$$

given $D_S = \{0, 1\}$.

In this case, the value q is initialized to $\lfloor \lg 10 \rfloor = 3$.

- On the first step when $q \leftarrow 3$, the possible (x, y) are $(3,0),(2,1),(1,2),(0,3)$, and the double-base chain we find $v = \left\lfloor \frac{r}{2^x 3^y} \right\rfloor$ are

$$\left\lfloor \frac{10}{2^3 3^0} \right\rfloor = 1,$$

$$\left\lfloor \frac{10}{2^2 3^1} \right\rfloor = \left\lfloor \frac{10}{2^1 3^2} \right\rfloor = \left\lfloor \frac{10}{2^0 3^3} \right\rfloor = 0,$$

  The optimal expansion of 0 is $\langle\langle\rangle, \langle\rangle, \langle\rangle\rangle$, and the optimal expansion of 1 is $\langle\langle 1\rangle, \langle 0\rangle, \langle 0\rangle\rangle$.

- We move to the second step when $q \leftarrow 2$. The possible (x, y) are $(2,0),(1,1),(0,2)$. In this case, we find the optimal double-base chains of

$$\left\lfloor \frac{10}{2^2 3^0} \right\rfloor = 2,$$

$$\left\lfloor \frac{10}{2^1 3^1} \right\rfloor = \left\lfloor \frac{10}{2^0 3^2} \right\rfloor = 1.$$

  From the first step,

$$C[1] = \langle\langle 1\rangle, \langle 0\rangle, \langle 0\rangle\rangle,$$

  with $P(C[1]) = 0$. The only way to compute $2S$ is to double the point S. Hence,

$$C[2] = \langle\langle 1\rangle, \langle 1\rangle, \langle 0\rangle\rangle,$$

  and

$$P(C[2]) = P(C[1]) + P_{dbl} = 0 + 1 = 1.$$

- The third step is when $q \leftarrow 1$. The possible (x, y) are $(1,0)$ and $(0,1)$, and we find the optimal double-base chains of

$$\left\lfloor \frac{10}{2^1 3^0} \right\rfloor = 5,$$

$$\left\lfloor \frac{10}{2^0 3^1} \right\rfloor = 3.$$

  The only way to compute $5S$ is to double the point $2S$ and add the result with S, i.e.

$$5S = 2(2S) + S.$$

  Then, we edit $C[2] = \langle\langle 1\rangle, \langle 1\rangle, \langle 0\rangle\rangle$ to

$$C[5] = \langle\langle 1, 1\rangle, \langle 0, 2\rangle, \langle 0, 0\rangle\rangle,$$

  and

$$\begin{aligned} P(C[5]) &= P(C[2]) + P_{dbl} + P_{add} \\ &= 1 + 1 + 1 = 3. \end{aligned}$$

  On the other hand, there are two choices for the optimal double-base chain of $3S$, $C[3]$. The first choice is to triple S. In this case, the computation cost will be

$$C[1] + P_{tpl} = 0 + 1 = 1.$$

  The other choice is to double the point S to $2S$, and add the result with S. The computation cost in this case is

$$C[1] + P_{dbl} + P_{add} = 0 + 1 + 1 = 2.$$

  Then, the optimal case is the first case, and we edit $C[1] = \langle\langle 1\rangle, \langle 0\rangle, \langle 0\rangle\rangle$ to

$$C[3] = \langle\langle 1\rangle, \langle 0\rangle, \langle 1\rangle\rangle.$$

- In the last step, $q \leftarrow 10$, and $(x, y) = (0, 0)$. The only optimal double-base chain we need to find in this step is our output, $C[10]$. There are two ways to compute $10S$ using double-base chain. The first way is to double a point $5S$. The computation cost in this case is

$$P(C[5]) + P_{dbl} = 3 + 1 = 4.$$

  The other choice is to triple the point $3S$ to $9S$, and add the result with S. The computation cost in this case is

$$P(C[3]) + P_{tpl} = 1 + 1 = 2.$$

  Then, the optimal case is the second case, and we edit $C[3] = \langle\langle 1\rangle, \langle 0\rangle, \langle 1\rangle\rangle$ to

$$C[10] = \langle\langle 1, 1\rangle, \langle 0, 0\rangle, \langle 0, 2\rangle\rangle.$$

Algorithm 3: The algorithm finding the optimal double-base chain for single integer for any D_S

```
input  : The positive integer $r$, the finite digit set $D_S$, and the carry set $G$
output : The optimal double-base chain of $r$, $C[r] = \langle R, X, Y \rangle$

1  $q \leftarrow \lfloor \lg r \rfloor$
2  while $q \geq 0$ do
3      forall the $x, y \in \mathbb{Z}$ such that $x + y = q$ do
4          $v \leftarrow \left\lfloor \frac{r}{2^x 3^y} \right\rfloor$
5          forall the $g_v \in G$ do
6              $va \leftarrow v + g_v$
7              if $va = 0$ then $C[0] \leftarrow \langle \langle \rangle, \langle \rangle, \langle \rangle \rangle$
8              if $va \in D_S$ then $C[va] \leftarrow \langle \langle va \rangle, \langle 0 \rangle, \langle 0 \rangle \rangle$
9              else $C[va] \leftarrow FO(va, C[G + \lfloor \frac{v}{2} \rfloor], C[G + \lfloor \frac{v}{3} \rfloor])$
10         end
11     end
12     $q \leftarrow q - 1$
13 end
```

3.2. Generalized Algorithm for Any Digit Sets

When $D_S = \{0, 1\}$, we usually have two choices to compute $C[r]$. One is to perform a point double, and use the subsolution $C[\lfloor \frac{r}{2} \rfloor]$. The other is to perform a point triple, and use the subsolution of $C[\lfloor \frac{r}{3} \rfloor]$. However, we have more choices when we deploy larger digit set. For example, when $D_S = \{0, \pm 1\}$

$$5 = 2 \times 2 + 1 = 3 \times 2 - 1 = 2 \times 3 - 1,$$

the number of cases increase from one in the previous subsection to three. Also, we need more optimal subsolution in this case. Even for point double, we need $C[\lfloor \frac{r}{2} \rfloor] = C[\lfloor \frac{5}{2} \rfloor] = C[2]$ and $C[\lfloor \frac{r}{2} \rfloor + 1] = C[\lfloor \frac{5}{2} \rfloor + 1] = C[3]$. We call $\lfloor \frac{r}{2} \rfloor = 2$ as *standard*, and the additional term g_r as *carry*. For example, carry $g_2 = 1$ when we com-

Algorithm 4: Function FO (referred in Algorithm 3)

```
input  : The positive integer $r$,
         the optimal double-base chain of $g + \lfloor \frac{r}{2} \rfloor$ for $g \in G$, the optimal double-base chain of $g + \lfloor \frac{r}{3} \rfloor$ for $g \in G$
output : The optimal double-base chain of $r$, $C[r] = \langle R, X, Y \rangle$

1  $c_{2,u}, c_{3,u} \leftarrow \infty$ for all $u \in D_S$
2  forall the $u \in D_S$ such that $r \equiv u \bmod 2$ do
3      $c_{2,u} \leftarrow P(C[\frac{r-u}{2}]) + P_{dbl}$
4      if $u \neq 0$ then $c_{2,u} \leftarrow c_{2,u} + P_{add}$
5  end
6  $c_2 \leftarrow \min c_{2,u}$, $u_2 \leftarrow$ minarg $c_{2,u}$, $v_2 \leftarrow \frac{r-u}{2}$
7  forall the $u \in D_S$ such that $r \equiv u \bmod 3$ do
8      $c_{3,u} \leftarrow P(C[\frac{r-u}{3}]) + P_{tpl}$
9      if $u \neq 0$ then $c_{3,u} \leftarrow c_{3,u} + P_{add}$
10 end
11 $c_3 \leftarrow \min c_{3,u}$, $u_3 \leftarrow$ minarg $c_{3,u}$, $v_3 \leftarrow \frac{r-u}{3}$
12 if $c_2 \leq c_3$ then
13     Let $C[v_2] = \langle \langle r'_t \rangle_{t=0}^{m-2}, \langle x'_t \rangle_{t=0}^{m-2}, \langle y'_t \rangle_{t=0}^{m-2} \rangle$
14     if $u_2 = 0$ then
15         $r_t = r'_t$, $x_t = x'_t + 1$, $y_t = y'_t$
16     else
17         $r_0 = u_2$, $x_0 = 0$, $y_0 = 0$, $r_t = r'_{t-1}$, $x_t = x'_{t-1} + 1$, $y_t = y'_{t-1}$
18
19 end
20 else
21     Let $C[v_3] = \langle \langle r'_t \rangle_{t=0}^{m-2}, \langle x'_t \rangle_{t=0}^{m-2}, \langle y'_t \rangle_{t=0}^{m-2} \rangle$
22     if $u_3 = 0$ then
23         $r_t = r'_t$, $x_t = x'_t$, $y_t = y'_t + 1$
24     else
25         $r_0 = u_3$, $x_0 = 0$, $y_0 = 0$, $r_t = r'_{t-1}$, $x_t = x'_{t-1}$, $y_t = y'_{t-1} + 1$
26
27 end
28 if u = 0 then $C[r] \leftarrow \langle \langle r_t \rangle_{t=0}^{m-2}, \langle x_t \rangle_{t=0}^{m-2}, \langle y_t \rangle_{t=0}^{m-2} \rangle$
29 else $C[r] \leftarrow \langle \langle r_t \rangle_{t=0}^{m-1}, \langle x_t \rangle_{t=0}^{m-1}, \langle y_t \rangle_{t=0}^{m-1} \rangle$
```

pute $C[3]$, and carry $g_2 = 0$ when we compute $C[2]$.

Suppose that we are now consider a standard r with a carry g_r. Assume that the last two steps to compute $(r + g_r)P$ are point double and point addition with $u \in D_S$. We get a relation

$$r + g_r = 2U + u,$$

when U can be written as a summation of a standard $\lfloor \frac{r}{2} \rfloor$ and a carry $g_{\lfloor \frac{r}{2} \rfloor}$. Hence,

$$r + g_r = 2(\left\lfloor \frac{r}{2} \right\rfloor + g_{\lfloor \frac{r}{2} \rfloor}) + u,$$

$$r - 2 \cdot \left\lfloor \frac{r}{2} \right\rfloor + g_r = 2 \cdot g_{\lfloor \frac{r}{2} \rfloor} + u,$$

$$(r \bmod 2) + g_r = 2 \cdot g_{\lfloor \frac{r}{2} \rfloor} + u.$$

Let

$$C[\left\lfloor \frac{r}{2} \right\rfloor + g_{\lfloor \frac{r}{2} \rfloor}] = \langle \langle r'_t \rangle_{t=0}^{m-2}, \langle x'_t \rangle_{t=0}^{m-2}, \langle y'_t \rangle_{t=0}^{m-2} \rangle$$

be the optimal solution of $\lfloor \frac{r}{2} \rfloor + g_{\lfloor \frac{r}{2} \rfloor}$. The optimal solution

$$C_{2,u}[r + g_r] = \langle \langle r_t \rangle_{t=0}^{m-1}, \langle x_t \rangle_{t=0}^{m-1}, \langle y_t \rangle_{t=0}^{m-1} \rangle,$$

where

$$r_0 = u,$$

$$r_t = r'_{t-1}$$

for $1 \leq t \leq m - 1$,

$$x_0 = 0,$$

$$x_t = x'_{t-1} + 1$$

for $1 \leq t \leq m - 1$,

$$y_0 = 0,$$

$$y_t = y'_{t-1}$$

for $1 \leq t \leq m - 1$.

If the last two steps to compute $(r + g_r)P$ are point triple and point addition with $u \in D_S$. We get a relation

$$r + g_r = 3(\left\lfloor \frac{r}{3} \right\rfloor + g_{\lfloor \frac{r}{3} \rfloor}) + u,$$

when $u \in D_S$. Same as the case for point double, we get a relation

$$(r \bmod 3) + g_r = 3 \cdot g_{\lfloor \frac{r}{3} \rfloor} + u.$$

In this case, the optimal solution

$$C_{3,u}[r + g_r] = \langle \langle r_t \rangle_{t=0}^{m-1}, \langle x_t \rangle_{t=0}^{m-1}, \langle y_t \rangle_{t=0}^{m-1} \rangle,$$

where

$$r_0 = u,$$

$$r_t = r'_{t-1}$$

for $1 \leq t \leq m - 1$,

$$x_0 = 0,$$

$$x_t = x'_{t-1}$$

for $1 \leq t \leq m - 1$,

$$y_0 = 0,$$

$$y_t = y'_{t-1} + 1$$

for $1 \leq t \leq m - 1$ if

$$C[\left\lfloor \frac{r}{3} \right\rfloor + g_{\lfloor \frac{r}{3} \rfloor}] = \langle \langle r'_t \rangle_{t=0}^{m-2}, \langle x'_t \rangle_{t=0}^{m-2}, \langle y'_t \rangle_{t=0}^{m-2} \rangle$$

be the optimal solution of $\lfloor \frac{r}{3} \rfloor + g_{\lfloor \frac{r}{3} \rfloor}$.

In our algorithm, we compute $C_{2,u}[r + g_r], C_{3,u}[r + g_r]$ for all $u \in D_S$, and the chain with the smallest cost $P(C_{2,u}[r + g_r]), P(C_{3,u}[r + g_r])$ is chosen to be the optimal solution $C[r + g_r]$.

Suppose that the input of the algorithm is r, and we are computing $C[r]$. Our algorithm needs an optimal chain of $\lfloor \frac{r}{2} \rfloor + g_{\lfloor \frac{r}{2} \rfloor}$ and $\lfloor \frac{r}{3} \rfloor + g_{\lfloor \frac{r}{3} \rfloor}$. Then, our algorithm requires an optimal chain of $\lfloor \frac{r}{4} \rfloor + g_{\lfloor \frac{r}{4} \rfloor}$, $\lfloor \frac{r}{6} \rfloor + g_{\lfloor \frac{r}{6} \rfloor}$, and $\lfloor \frac{r}{9} \rfloor + g_{\lfloor \frac{r}{9} \rfloor}$ to compute $C[\lfloor \frac{r}{2} \rfloor + g_{\lfloor \frac{r}{2} \rfloor}]$ and $C[\lfloor \frac{r}{3} \rfloor + g_{\lfloor \frac{r}{3} \rfloor}]$. Let the set of possible g_k be G_k, i.e. $G_r = \{0\}$ when r is an input of the algorithm. Define

$$G = \bigcup_{x,y \in \mathbb{Z}} G_{\left\lfloor \frac{r}{2^x 3^y} \right\rfloor}.$$

We show in Appendix A that G is a finite set if D_S is finite. This infers that it is enough to compute $C[\left\lfloor \frac{r}{2^x 3^y} \right\rfloor + g]$ for each $g \in G$ when we consider a standard $\left\lfloor \frac{r}{2^x 3^y} \right\rfloor$. We illustrate the idea in Example 3 and Figure 3.

Example 3 Compute the optimal double-base chain of 5 when $P_{add} = P_{dbl} = P_{tpl} = 1$ and $D_S = \{0, \pm 1\}$.

When $D_S = \{0, \pm 1\}$, we can compute the carry set $G = \{0, 1\}$ using algorithm proposed in Appendix A.

We want to compute $C[5] = \langle R, X, Y \rangle$ such that $r_i \in D_S$ and $x_i, y_i \in \mathbb{Z}$, $x_i \leq x_{i+1}$, $y_i \leq y_{i+1}$. 5 can be rewritten as follows:

$$5 = 2 \times 2 + 1 = (2+1) \times 2 - 1 = (1+1) \times 3 - 1.$$

We need $C[2]$ ($\left\lfloor \frac{5}{2} \right\rfloor = 2$, $g_2 = 0$ or $\left\lfloor \frac{5}{3} \right\rfloor = 1$, $g_1 = 1$) and $C[3]$ ($\left\lfloor \frac{5}{2} \right\rfloor = 2$, $g_2 = 1$).

It is easy to see that the optimal chain $C[2] = \langle \langle 1 \rangle, \langle 1 \rangle, \langle 0 \rangle \rangle$ and $C[3] = \langle \langle 1 \rangle, \langle 0 \rangle, \langle 1 \rangle \rangle$. $P(C[2]) = P(C[3]) = 1$.

We choose the best choice among $5 = 2 \times 2 + 1$, $5 = 3 \times 2 - 1$, $5 = 2 \times 3 - 1$. By the first choice, we get the chain

$$C_{2,1}[5] = \langle \langle 1, 1 \rangle, \langle 0, 0 \rangle, \langle 0, 2 \rangle \rangle.$$

The second choice and the third choice is

$$C_{2,-1}[5] = C_{3,-1}[5] = \langle \langle -1, 1 \rangle, \langle 0, 1 \rangle, \langle 0, 1 \rangle \rangle.$$

We get

$$P(C_{2,1}[5]) = P(C_{2,-1}[5]) = P(C_{3,-1}[5]) = 3,$$

and all of them can be the optimal chain.

Using the idea explained in this subsection, we propose Algorithms 3,4.

4. EXPERIMENTAL RESULTS

To evaluate our algorithm, we show some experimental results in this section. We perform the experiment on each implementation environment such as the scalar multiplication defined on the binary field ($\mathbb{F}_{2^q}$) and the scalar multiplication defined on the prime field ($\mathbb{F}_p$). In this section, we will consider the computation time of point addition, point double, and point triple defined in Section 1 as the number of the operations in lower layer, field inversion ($[i]$), field squaring ($[s]$), and field multiplication ($[m]$), i.e. we show the average computation time of scalar multiplication in terms of $\alpha[i] + \beta[s] + \xi[m]$. Then, we approximate the computation time of field squaring $[s]$ and field inversion $[i]$ in terms of multiplicative factors of field multiplication $[m]$, and compare our algorithm with existing algorithms.

4.1. Scalar Multiplication on Binary Field

In the binary field, the field squaring is very fast, i.e. $[s] \approx 0$. Normally,

$$3 \leq [i]/[m] \leq 10.$$

Basically,

$$P_{dbl} = P_{add} = [i] + [s] + 2[m],$$

and there are many researches working on optimizing more complicated operation such as point triple and point quadruple [14] [15]. Moreover, when point addition is chosen to perform just after the point double, we can use some intermediate results of point double to reduce the computation time of point addition. Then, it is more effective to consider point double and point addition together as one basic operation. We call the operation as point double-and-add, with the computation time

$$P_{dbl+add} < P_{dbl} + P_{add}.$$

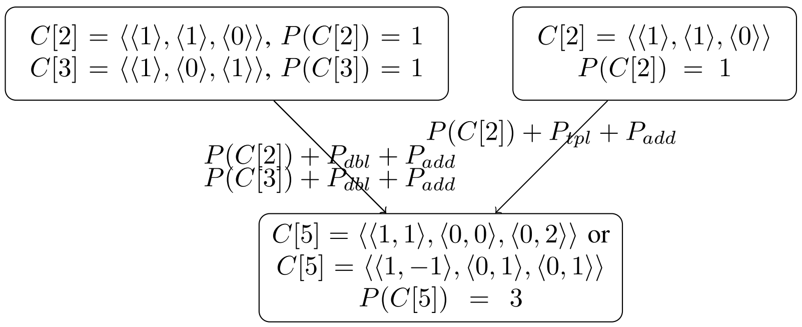

$C[2] = \langle\langle 1\rangle, \langle 1\rangle, \langle 0\rangle\rangle$, $P(C[2]) = 1$
$C[3] = \langle\langle 1\rangle, \langle 0\rangle, \langle 1\rangle\rangle$, $P(C[3]) = 1$
$C[2] = \langle\langle 1\rangle, \langle 1\rangle, \langle 0\rangle\rangle$
$P(C[2]) = 1$
$P(C[2]) + P_{tpl} + P_{add}$
$P(C[2]) + P_{dbl} + P_{add}$
$P(C[3]) + P_{dbl} + P_{add}$
$C[5] = \langle\langle 1, 1\rangle, \langle 0, 0\rangle, \langle 0, 2\rangle\rangle$ or
$C[5] = \langle\langle 1, -1\rangle, \langle 0, 1\rangle, \langle 0, 1\rangle\rangle$
$P(C[5]) = 3$

Figure 3. Given $D_S = \{0, \pm 1\}$, we can compute $C[5]$ by three ways. The first way is to compute $C[2]$, and perform a point double and a point addition. The second is to compute $C[3]$, perform a point double, and a point substitution (addition with $-S$). The third is to compute $C[2]$, perform a point triple, and a point substitution. All methods consume the same cost.

The similar thing also happens when we perform point addition after point triple, and we also define point triple-and-add as another basic operation, with the computation time

$$P_{tpl+add} < P_{tpl} + P_{add}.$$

With some small improvements of Algorithms 3,4, we can also propose the algorithm which output the optimal chains under the existence of $P_{dbl+add}$ and $P_{tpl+add}$.

To perform experiments, we use the same parameters as [7] for P_{dbl}, P_{tpl}, P_{add}, $P_{dbl+add}$, and $P_{tpl+add}$ (these parameters are shown in Appendix B). We set $D_S = \{0, \pm 1\}$, and randomly select $10,000$ positive integers which are less than 2^{163}, and find the average computation cost comparing between the optimal chain proposed in this paper and the greedy algorithm presented in [7]. The results are shown in Table 1. Our result is 4.06% better than [7] when $[i]/[m] = 4$, and 4.77% better than [7] when $[i]/[m] = 8$. We note that the time complexity of Binary and NAF itself is $O(n)$, while the time complexity of Ternary/Binary, DBC(Greedy), and Optimized DBC is $O(n^2)$.

4.2. Scalar Multiplication on Prime Field

When we compute the scalar multiplication on prime field, field inversion is a very expensive task as $[i]/[m]$ is usually more than 30. To cope with that, we compute scalar multiplication in the coordinate in which optimize the number of field inversion we need to perform such as inverted Edward coordinate with a curve in Edwards form [11]. Up to this state, it is the fastest way to implement scalar multiplication.

In our experiment, we use the computation cost $P_{dbl}, P_{tpl}, P_{add}$ as in [6], and set $D_S = 0, \pm 1$. We perform five experiments, for the positive integer less than 2^{192}, 2^{256}, 2^{320}, and 2^{384}. In each experiment, we randomly select $10,000$ integers, and find the average computation cost in terms of $[m]$. We show that results in Table 2. Our results improve the tree-based approach proposed by Doche and Habsieger by $3.95\%, 3.88\%, 3.90\%, 3.90\%, 3.90\%$ when bit numbers are $192, 256, 320, 384$ respectively.

We also evaluate the average running time of our algorithm itself in this experiment. Shown in Table 3, we compare the average computation of our method with the existing greedy-type algorithm using Java in Windows Vista, AMD Athlon(tm) 64X2 Dual Core Processor 4600+ 2.40GHz. The most notable result in the table is the result for 448-bit inputs. In this case, the average running time of our algorithm is 30ms, while the existing algorithm [7] takes 29ms. We note that the difference between two average running time is negligible, as the average computation time of scalar multiplication in Java is shown to be between 400-650ms [12].

We also compare our results with the other digit

Table 1. Comparing the computation cost for scalar point multiplication using double-base chains when the elliptic curve is implemented in the binary field

Method	$[i]/[m] = 4$	$[i]/[m] = 8$
Binary	$1627[m]$	$2441[m]$
NAF [1]	$1465[m]$	$2225[m]$
Ternary/Binary [5]	$1463[m]$	$2168[m]$
DBC (Greedy) [7]	$1427[m]$	$2139[m]$
Optimized DBC (Our Result)	$1369[m]$	$2037[m]$

Table 2. Comparing the computation cost for scalar point multiplication using double-base chains when the elliptic curve is implemented in the prime field

Method	192 bits	256 bits	320 bits	384 bits
NAF [1]	$1817.6[m]$	$2423.5[m]$	$3029.3[m]$	$3635.2[m]$
Ternary/Binary [5]	$1761.2[m]$	$2353.6[m]$	$2944.9[m]$	$3537.2[m]$
DB-Chain (Greedy) [7]	$1725.5[m]$	$2302.0[m]$	$2879.1[m]$	$3455.2[m]$
Tree-Based Approach [9]	$1691.3[m]$	$2255.8[m]$	$2821.0[m]$	$3386.0[m]$
Our Result	$1624.5[m]$	$2168.2[m]$	$2710.9[m]$	$3254.1[m]$

Table 3. The average running time of our algorithm compared with the existing algorithm [7] when $D_S = \{0, \pm 1\}$

Input Size	[7]	Our Results
192 Bits	4ms	7ms
256 Bits	6ms	13ms
320 Bits	20ms	21ms
384 Bits	29ms	30ms

sets. In this case, we compare our results with the work by Bernstein et al. [8]. In the paper, they use the different way to measure the computation cost of sclar multiplication. In addition to the cost of computing rS, they also consider the cost for precomputations. For example, the cost to compute $\pm 3S, \pm 5S, \ldots, \pm 17S$ is also included in the computation cost of any rP computed using $D_S = \{0, \pm 1, \pm 3, \ldots, \pm 17\}$. We perform the experiment on eight different curves and coordinates. In each curve, the computation cost for point double, point addition, and point triple are different, and we use the same parameters as defined in [8]. We use

$$D_S = \{0, \pm 1, \pm 3, \ldots, \pm(2h+1)\}$$

when we optimize $0 \leq h \leq 20$ that give us the minimal average computation cost. Although, the computation cost of the scalar multiplication tends to be lower if we use larger digit set, the higher precompuation cost makes optimal h lied between 6 to 8 in most of cases.

Recently, Meloni and Hasan [6] proposed a new paradigm to compute scalar multiplication using double-base number system. Instead of using double-base chain, they cope with the difficulties computing the number system introducing Yao's algorithm. Their results significantly improves the result using the double-base chain using greedy algorithm, especially the curve where point triple is expensive.

In Tables 4-5, we compare the results in [8] and [6] with our algorithm. Again, we randomly choose $10,000$ positive integers less than 2^{160} in Table 4, and less than 2^{256} in Table 5. We significantly improve the results of [8]. On the other hand, our results do not improve the result of [6]

Table 4. Comparing the computation cost for scalar point muliplication using double-base chains in larger digit set when the elliptic curve is implemented in the prime field, and the bit number is 160. The results in this table are different from the others. Each number is the cost for computing a scalar multiplication with the precomputation time. In each case, we find the digit set D_S that makes the number minimal.

Method	3DIK	Edwards	ExtJQuartic	Hessian
DBC + Greedy Alg. [8]	$1502.4[m]$	$1322.9[m]$	$1311.0[m]$	$1565.0[m]$
DBNS + Yao's Alg. [6]	$1477.3[m]$	$1283.3[m]$	$1226.0[m]$	$1501.8[m]$
Our Algorithm	$1438.7[m]$	$1284.3[m]$	$1276.5[m]$	$1514.4[m]$

Method	InvEdwards	JacIntersect	Jacobian	Jacobian-3
DBC + Greedy Alg. [8]	$1290.3[m]$	$1438.8[m]$	$1558.4[m]$	$1504.3[m]$
DBNS + Yao's Alg. [6]	$1258.6[m]$	$1301.2[m]$	$1534.9[m]$	$1475.3[m]$
Our Algorithm	$1257.5[m]$	$1376.0[m]$	$1514.5[m]$	$1458.0[m]$

Table 5. Comparing the computation cost for scalar point muliplication using double-base chains in larger digit set when the elliptic curve is implemented in the prime field, and the bit number is 256. The results in this table are different from the others. Each number is the cost for computing a scalar multiplication with the precomputation time. In each case, we find the digit set D_S that makes the number minimal.

Method	3DIK	Edwards	ExtJQuartic	Hessian
DBC + Greedy Alg. [8]	$2393.2[m]$	$2089.7[m]$	$2071.2[m]$	$2470.6[m]$
DBNS + Yao's Alg. [6]	$2319.2[m]$	$2029.8[m]$	$1991.4[m]$	$2374.0[m]$
Our Algorithm	$2287.4[m]$	$2031.2[m]$	$2019.4[m]$	$2407.4[m]$

Method	InvEdwards	JacIntersect	Jacobian	Jacobian-3
DBC + Greedy Alg. [8]	$2041.2[m]$	$2266.1[m]$	$2466.2[m]$	$2379.0[m]$
DBNS + Yao's Alg. [6]	$1993.3[m]$	$2050.0[m]$	$2416.2[m]$	$2316.2[m]$
Our Algorithm	$1989.9[m]$	$2173.5[m]$	$2413.2[m]$	$2319.9[m]$

in many cases such as Hessian curves. These cases are the case when point triple is a costly operation, and we need only few point triples in the optimal chain. In this case, Yao's algorithm works effciently. However, our algorithm works better in the case where point triple is fast compared to point addition such as 3DIK and Jacobian-3. Our algorithm works better in the inverted Edward coordinate, which is commonly used as a benchmark to compare scalar multiplication algorithms.

5. CONCLUSION

In this work, we use the dynamic programming algorithm to present the optimal double-base chain. The chain guarantees the optimal computation cost on the scalar multiplication. The time complexity of the algorithm is $O(\lg^2 r)$ similar to the greedy algorithm. The experimental results show that the optimal chains significatly improve the efficiency of scalar multiplication from the greedy algorithm.

As future works, we want to analyze the minimal average number of terms required for each integer in double-base chain. In double-base number system, it is proved that the average number of terms required to define integer r, when $0 \leq r < 2^q$ is $o(q)$ [5]. However, it is proved that the average number of terms in the double-base chains provided by greedy algorithm is in $\Theta(q)$. Then, it is interesting to prove if the minimal average number of terms in the chain is $o(q)$. The result might introduce us to a sublinear time algorithm for scalar multiplication.

Another future work is to apply the dynamic programming algorithm to DBNS. As the introduction of Yao's algorithm with a greedy algorithm makes scalar multiplication significantly faster, we expect futhre improvement using the algorithm which outputs the optimal double-base number system. However, we recently found many clues suggesting that the problem might be NP-hard.

References

[1] O. Egecioglu and C. K. Koc, "Exponentiation using canonical recoding," *Theoretical Computer Science*, vol. 8, no. 1, pp. 19–38, 1994.

[2] J. A. Muir and D. R. Stinson, "New minimal weight representation for left-to-right window methods," *Department of Combinatorics and Optimization, School of Computer Science, University of Waterloo*, 2004.

[3] T. Takagi, D. Reis, S. M. Yen, and B. C. Wu, "Radix-r non-adjacent form and its application to pairing-based cryptosystem," *IEICE Trans. Fundamentals*, vol. E89-A, pp. 115–123, January 2006.

[4] V. Dimitrov and T. V. Cooklev, "Two algorithms for modular exponentiation based on nonstandard arithmetics," *IEICE Trans. Fundamentals*, vol. E78-A, pp. 82–87, January 1995. special issue on cryptography and information security.

[5] V. S. Dimitrov, G. A. Jullien, and W. C. Miller, "An algorithm for modular exponentiations," *Information Processing Letters*, vol. 66, pp. 155–159, 1998.

[6] N. Meloni and M. A. Hasan, "Elliptic curve scalar multiplication combining yao's algorithm and double bases," in *CHES 2009*, pp. 304–316, 2009.

[7] V. Dimitrov, L. Imbert, and P. K. Mishra, "The double-base number system and its application to elliptic curve cryptography," *Mathematics of Computation*, vol. 77, pp. 1075–1104, 2008.

[8] D. J. Bernstein, P. Birkner, T. Lange, and C. Peters, "Optimizing double-base elliptic-curve single-scalar multiplication," in *In Progress in Cryptology - INDOCRYPT 2007*, vol. 4859 of *Lecture Notes in Computer Science*, pp. 167–182, Springer, 2007.

[9] C. Doche and L. Habsieger, "A tree-based approach for computing double-base chains," in *ACISP 2008*, pp. 433–446, 2008.

[10] Same Authors as This Paper, "Fast elliptic curve cryptography using minimal weight conversion of d integers," in *Proceedings of AISC 2012* (J. Pieprzyk and C. Thomborson, eds.), vol. 125 of *CRPIT*, pp. 15–26, ACS, January 2012.

[11] D. Bernstein and T. Lange, "Explicit-formulas database (http://www.hyperelliptic.org/efd/)," 2008.

[12] J. Grobschadl and D. Page, "E?cient java implementation of elliptic curve cryptography for j2me-enabled mobile devices," *cryptology ePrint Archive*, vol. 712, 2011.

[13] L. Imbert and F. Philippe, "How to compute shortest double-base chains?," in *ANTS IX*, July 2010.

[14] M. Ciet, M. Joye, K. Lauter, and P. L. Montgomery, "Trading inversions for multiplications in elliptic curve cryptography," *Designs, Codes and Cryptography*, vol. 39, no. 6, pp. 189–206, 2006.

[15] K. Eisentrager, K. Lauter, and P. L. Montgomery, "Fast elliptic curve arithmetic and improved Weil pairing evaluation," in *Topics in Cryptology - CT-RSA 2003*, vol. 2612 of *Lecture Notes in Computer Science*, pp. 343–354, Springer, 2003.

[16] P. Longa and C. Gebotys, "Fast multibase methods and other several optimizations for elliptic curve scalar multiplication," *Proc. of PKC 2009*, pp. 443–462, 2009.

A. THE CARRY SET

As discussed in Section 3, C_S depends on the digit set D_S. If $D_S = \{0, 1\}$, we need only the solution of

$$\left\lfloor \frac{r_1}{2} \right\rfloor S_1 + \cdots + \left\lfloor \frac{r_d}{2} \right\rfloor S_d,$$

$$\left\lfloor \frac{r_1}{3} \right\rfloor S_1 + \cdots + \left\lfloor \frac{r_d}{3} \right\rfloor S_d.$$

However, if the digit set is not $\{0, 1\}$, we will also need other sub-solutions. Shown in Example 1, we need

$$(\left\lfloor \frac{r_1}{2} \right\rfloor + c_1)S_1 + \cdots + (\left\lfloor \frac{r_d}{2} \right\rfloor + c_2)S_d,$$

Algorithm 5: Find the carry set of the given digit set

```
input : the digit set $D_S$
output: the carry set $C_S$
1  $Ct \leftarrow \{0\}, C_S \leftarrow \oslash$
2  while $Ct \neq \oslash$ do
3      Pick $x \in Ct$
4      $Ct \leftarrow Ct \cup (\{\frac{x+d}{2} \in \mathbb{Z} | d \in D_S\} - C_S - \{x\})$
5      $Ct \leftarrow Ct \cup (\{\frac{x+d+1}{2} \in \mathbb{Z} | d \in D_S\} - C_S - \{x\})$
6      $Ct \leftarrow Ct \cup (\{\frac{x+d}{3} \in \mathbb{Z} | d \in D_S\} - C_S - \{x\})$
7      $Ct \leftarrow Ct \cup (\{\frac{x+d+1}{3} \in \mathbb{Z} | d \in D_S\} - C_S - \{x\})$
8      $C_S \leftarrow C_S \cup \{x\}$
9      $Ct \leftarrow Ct - \{x\}$
10 end
```

$$(\left\lfloor\frac{r_1}{3}\right\rfloor + c_3)S_1 + \cdots + (\left\lfloor\frac{r_d}{3}\right\rfloor + c_4)S_d,$$

when $c_1, c_2, c_3, c_4 \in \{0, 1\} = C_{S,1}$. Actually, the set $C_{S,1} = C_{BS,1} \cup C_{TS,1}$ when

$$C_{BS,1} = \bigcup_{l\in\{0,1\}} \{\frac{l-d}{2} | d \in D_S \wedge d \equiv l \bmod 2\},$$

$$C_{TS,1} = \bigcup_{l\in\{0,1,2\}} \{\frac{l-d}{3} | d \in D_S \wedge d \equiv l \bmod 3\}$$

However, the carry set $C_{S,1}$ defined above is not enough. When, we find the solutions for each $(\lfloor\frac{r_1}{2}\rfloor + c_1)S_1 + \cdots + (\lfloor\frac{r_d}{2}\rfloor + c_2)S_d$ and $(\lfloor\frac{r_1}{3}\rfloor + c_3)S_1 + \cdots + (\lfloor\frac{r_d}{3}\rfloor + c_4)S_d$, we will need

$$(\left\lfloor\frac{r_1}{4}\right\rfloor + c_5)S_1 + \cdots + (\left\lfloor\frac{r_d}{4}\right\rfloor + c_6)S_d,$$

$$(\left\lfloor\frac{r_1}{6}\right\rfloor + c_7)S_1 + \cdots + (\left\lfloor\frac{r_d}{6}\right\rfloor + c_8)S_d,$$

$$(\left\lfloor\frac{r_1}{9}\right\rfloor + c_9)S_1 + \cdots + (\left\lfloor\frac{r_d}{9}\right\rfloor + c_{10})S_d,$$

when $c_5, c_6, c_7, c_8, c_9, c_{10} \in C_{S,2} = C_{BS,2} \cup C_{TS,2}$ if

$$C_{BS,2} = \bigcup_{l\in\{0,1\}} \{\frac{l+c-d}{2} | d \equiv l \bmod 2\},$$

$$C_{TS,2} = \bigcup_{l\in\{0,1,2\}} \{\frac{l+c-d}{3} | d \equiv l \bmod 3\},$$

when $c \in C_{S,1} \wedge d \in D_S$.

Then, we get $C_{S,n+1} = C_{BS,n+1} \cup C_{TS,n+1}$ if

$$C_{BS,n+1} = \bigcup_{l\in\{0,1\}} \{\frac{l+c-d}{2} | d \equiv l \bmod 2\},$$

$$C_{TS,n+1} = \bigcup_{l\in\{0,1,2\}} \{\frac{l+c-d}{3} | d \equiv l \bmod 3\},$$

when $c \in C_{S,n} \wedge d \in D_S$.

We define C_S as

$$C_S = \bigcup_{t=1}^{\infty} C_{S,\infty}.$$

We propose an algorithm to find C_S in Algorithm 5 based on breadth-first search scheme. Also, we prove that C_S is finite set for all finite digit set D_S in Lemma A.1.

Lemma A.1 *Given the finite digit set D_S, Algorithm 5 always terminates. And,*

$$||C_S|| \leq \max D_S - \min D_S + 2,$$

when C_S is the output carry set.

Proof Since

$$C_S = \bigcup_{l\in\{0,1\}} \{\frac{l+c-d}{2} | c+d \equiv l \bmod 2\} \cup \bigcup_{l\in\{0,1,2\}} \{\frac{l+c-d}{3} | c+d \equiv l \bmod 3\},$$

Table 6. $P_{dbl}, P_{tpl}, P_{add}, P_{dbl+add}, P_{tpl+add}$ used in the experiment on Subsection 4.1

Operation	$[i]/[m] = 4$	$[i]/[m] = 8$
P_{dbl}	$[i] + [s] + 2[m]$	$[i] + [s] + 2[m]$
P_{add}	$[i] + [s] + 2[m]$	$[i] + [s] + 2[m]$
P_{tpl}	$2[i] + 2[s] + 3[m]$	$[i] + 4[s] + 7[m]$
$P_{dbl+add}$	$2[i] + 2[s] + 3[m]$	$[i] + 2[s] + 9[m]$
$P_{tpl+add}$	$3[i] + 3[s] + 4[m]$	$2[i] + 3[s] + 9[m]$

Table 7. $P_{dbl}, P_{tpl}, P_{add}$ used in the experiment on Subsection 4.2

Curve Shape	P_{dbl}	P_{tpl}	P_{add}
3DIK	$2[m] + 7[s]$	$6[m] + 6[s]$	$11[m] + 6[s]$
Edwards	$3[m] + 4[s]$	$9[m] + 4[s]$	$10[m] + 1[s]$
ExtJQuartic	$2[m] + 5[s]$	$8[m] + 4[s]$	$7[m] + 4[s]$
Hessian	$3[m] + 6[s]$	$8[m] + 6[s]$	$6[m] + 6[s]$
InvEdwards	$3[m] + 4[s]$	$9[m] + 4[s]$	$9[m] + 1[s]$
JacIntersect	$2[m] + 5[s]$	$6[m] + 10[s]$	$11[m] + 1[s]$
Jacobian	$1[m] + 8[s]$	$5[m] + 10[s]$	$10[m] + 4[s]$
Jacobian-3	$3[m] + 5[s]$	$7[m] + 7[s]$	$10[m] + 4[s]$

where $d \in D_S$, $c \in C_S$.

$$\min C_S \geq \frac{\min C_S - \max D_S}{2}.$$

Then,

$$\min C_S \geq -\max D_S.$$

Also,

$$\max C_S \leq -\min D_S + 1.$$

We conclude that if D_S is finite, C_S is also finite. And, Algorithm 5 always terminates.

$$||C_S|| \leq \max D_S - \min D_S + 2. \quad \blacksquare$$

B. $P_{dbl}, P_{tpl}, P_{add}$ USED IN OUR EXPERIMENTS

For scalar multiplication on binary field (Subsection 4.1), we use the same parameter as [7] shown in Table 6.

Shown in Table 7, the results in Subsection 4.2 are based on $P_{dbl}, P_{tpl}, P_{add}$ in [6]. In this state, many works have further improved the cost, but we use these cost to make the comparisons between our results and existing works can be done easily and precisely.

Recently, there is a work by Longa and Gebotys [16] on several improvements for double-base chains. The value $P_{dbl}, P_{tpl}, P_{add}$ they used are smaller than those given in Tables 6 and 7. After implementing their parameters on the number with 160 bits, we show the results in Table 8. We can reduce their computation times by 1.03%, 0.40%, and 0.71% in inverted Edwards coordinates, Jacobian-3, and ExtJQuartic respectively. In this state, we are finding the experimental result of other greedy algorithms under these $P_{dbl}, P_{tpl}, P_{add}$ value.

Table 8. Comparing our results with [16] when the bit number is 160.

Curve Shape	[16]	Our Results
InvEdwards	$1351[m]$	$1337[m]$
Jacobian-3	$1460[m]$	$1454[m]$
ExtJQuartic	$1268[m]$	$1259[m]$