International Journal of Digital Information and Wireless Communications (IJDIWC) 2(1): 115-134
The Society of Digital Information and Wireless Communications, 2012(ISSN 2225-658X)

OPTIMAL ELLIPTIC CURVE SCALAR MULTIPLICATION USING 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. We also extend our idea, and propose an algorithm to optimize multi-scalar multiplication. By that extension, we can improve a computation time of the operation by 3.2-11.3%.

KEYWORDS

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

1. INTRODUCTION

Elliptic curve cryptography is an alternative building block for cryptograpic scheme similar to the conventional RSA, but it is widely believed to be much more secure when implemented using the same key size. Thus, the cryptosystem is more suitable for the computation environment with limited memory consumption such as wireless sensor node, and possibly becomes the central part of secure wireless communication in the near future. Despite of that advantage, elliptic curve cryptography is considered to be slower compared to the conventional cryptosystem with the same key size. In this work, we want to reduce its computation by focusing on its bottleneck operation, *scalar multiplication*. The operation 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. DBNS 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 DBNS 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 DBNS 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 DBNS is difficult. To cope with the problem, Dimitrov et al. proposed to used double-base chains (DBC), DBNS with more restrictions. DBC $C[r] = \langle \langle r_t \rangle_{t=0}^{m-1}, \langle x_t \rangle_{t=0}^{m-1}, \langle y_t \rangle_{t=0}^{m-1} \rangle$ is similar to DBNS, but DBC 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 DBC, while $C_2[14]$ is.

Like binary expansion and DBNS, some integers have more than one DBC, and the efficiency of elliptic curve cryptography strongly depends on which chain we use. The algorithm to select efficient DBC 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 DBC. By the experiment, we show that the optimal DBC are better than the best greedy algorithm proposed on DBC [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 DBNS 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 DBC.

In this paper, we also extend ideas in our method to propose an algorithm that can output the optimal DBC for multi-scalar multiplication. The operation is one of bottleneck operations in elliptic curve digital signature scheme, and can be defined as

$$Q = r_1 S_1 + r_2 S_2 + \cdots + r_d S_d,$$

where $S_1, \ldots, S_d, Q$ are points on elliptic curve and $r_1, \ldots, r_d$ are positive integers. We show that our optimal DBC are better than the greedy algorithm proposed on double-chain [14] by $3.2 - 4.5\%$ when $d = 2$ and $D_S = \{0, \pm 1\}$, given the cost for elementary operations shown in [6], [7]. When $D_S = \{0, \pm 1 \pm 5\}$ and $D_S = \{0, \pm 1, \pm 2, 3\}$, our chains are $5.3 - 11.3\%$ better than [14], [15]. For multi-scalar multiplication, the computation time of our algorithms itself is only less than a second for 448-bit input.

The paper is organized as follows: we describe the double-and-add scheme, and how we utilize the DBC to elliptic curve cryptography in Section 2. In Section 3, we show our algorithm which outputs the optimal DBC. 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 DBC

2.1. Binary Expansion and Scalar Multiplication

Using the binary expansion $E = \langle e_0, \ldots, e_{n-1} \rangle$, where $r = \sum_{t=0}^{n-1} e_t 2^t$ explained in Section 1, we can compute the scalar multiplication $Q = rS$ by double-and-add scheme. 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, which are point doubles ($S + S$, $2S$) and point additions ($S + S'$ when $S \neq S'$). 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 $n - 1$ point doubles, and n point additions. Note that we need not the point addition on the first iteration. Also, $e_t S = O$ if $e_t = 0$, and we need not the point addition in this case. Hence, the number of the point additions is $W(E) - 1$, where $W(E)$ the Hamming weight of the expansion defined as:

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

where $W(e_t) = 0$ when $e_t = 0$ and $W(e_t) = 1$ otherwise. In our case, $W(E) = 7$.

The Hamming weight tends to be less if the digit set D_S is larger such as $D_S = \{0, \pm 1\}$. However, the cost for the precomputation of $e_t S$ for all $e_t \in D_S$ is higher in a bigger digit set.

2.2. DBC and Scalar Multiplication

In this subsection, we show how to apply the DBC $C[r] = \langle E, X, Y\rangle$, when $E = \langle e_0, e_1, \ldots, e_{m-1}\rangle$, $X = \langle x_0, x_1, \ldots, x_{m-1}\rangle$, $Y = \langle y_0, y_1, \ldots, y_{m-1}\rangle$ to compute scalar multiplication. For example, one of the DBC of $127 = 2^0 3^0 + 2^1 3^2 + 2^2 3^3$ is $C[127] = \langle E, X, Y\rangle$, where $E = \langle 1, 1, 1\rangle$, $X = \langle 0, 1, 2\rangle$, $Y = \langle 0, 2, 3\rangle$. 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 needed in the binary expansion, we also require point triples ($3S$). In this case, we need two point additions, two point doubles, and three point triples.

In the double-and-add method, the number of point doubles required is proved to be constantly equal to $n-1 = \lfloor \lg r \rfloor$. Then, the efficiency of the binary expansion strongly depends on the number of point additions or the Hamming weight. However, the number of point doubles and point triples are not constant in this representation. The number of point additions is $W(C) - 1$ when $W(C) := m$ is the number of terms in the chain C, and the number of point doubles and point triples are x_{m-1} and y_{m-1} respectively. Hence, we have 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}, and P_{add} are the cost for point double, point triple, and point addition respectively. Note that these costs are not considered in the literature [13] where only the Hamming weight is considered.

2.3. Binary Expansion and Multi-Scalar Multiplication

To compute multi-scalar multiplication $Q = r_1 S_1 + \cdots + r_d S_d$, we can utilize Shamir's trick [16]. Using the trick, the operation is claimed to be faster than operation to compute $r_1 S_1, \ldots, r_d S_d$ separately and add them together. We show the Shamir's trick for joint binary expansion in Algorithm 1. For example, we can compute $Q = r_1 S_1 + r_2 S_2 = 127 S_1 + 109 S_2$ given the expansion of r_1, r_2 as $E_1 = \langle 1, 1, 1, 1, 1, 1, 1\rangle$ and $E_2 = \langle 1, 0, 1, 1, 0, 1, 1\rangle$ as follows:

$$Q = 2(2(2(2(2(2D+D)+S_1)+D)+D)+S_1)+D,$$

where $D = S_1 + S_2$. Thus, our computation requires 6 point doubles, 6 point additions in this case, while we require 12 point doubles and 11 point additions for separated computation.

Algorithm 1: Shamir's trick with joint binary expansion

input : A point on elliptic curve $S_1, \ldots, S_d$, the positive integer $r_1, \ldots, r_d$ with the binary expansion $E_i = \langle e_{i,t}\rangle_{t=0}^{n-1}$
output: $Q = \sum_{i=1}^{d} r_i S_i$

```
Q ← O
for t ← n − 1 to 0 do
    Q ← Q + Σ_{i=1}^{d} e_{i,t} S_i
    if t ≠ 0 then Q ← 2Q
end
```

Similar to the Hamming weight for scalar multiplication, we define the joint Hamming weight for multi-scalar multiplication. Let

$$w_t = \begin{cases} 0 & \text{if } \langle e_{1,t}, \ldots, e_{d,t}\rangle = \langle \mathbf{0}\rangle, \\ 1 & \text{otherwise.} \end{cases}$$

We can define the joint Hamming weight $JW(E_1, \ldots, E_d)$ as

$$JW(E_1, \ldots, E_d) = \sum_{t=0}^{n-1} w_t.$$

In our example, the joint Hamming weight $JW(E_1, E_2)$ is 7.

From the definition of joint Hamming weight, we need $\lfloor \lg(\max(r_1, \ldots, r_d)) \rfloor$ point doubles, and $JW(E_1, \ldots, E_d) - 1$ point additions.

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

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

We show in Subsection 3.4 that G is a finite set if D_S is finite. This infers that it is enough to compute $C[\lfloor \frac{r}{2^x 3^y} \rfloor + g]$ for each $g \in G$ when we consider a standard $\lfloor \frac{r}{2^x 3^y} \rfloor$. We illustrate the idea in Example 3 and Figure 3.

Example 3 Compute the optimal DBC 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 Subsection 3.4.

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

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.

Algorithm 5: The algorithm finding the optimal DBC for d integers on any digit set D_S

input : A tuple $R = \langle r_1, \ldots, r_d \rangle$, the finite digit set D_S, and the carry set G
output: An optimal DBC of R, $C[R]$

```
q ← max_i ⌊lg r_i⌋
while q ≥ 0 do
    forall the x, y ∈ Z^+ such that x + y = q do
        v_i ← ⌊r_i / (2^x 3^y)⌋ for 1 ≤ i ≤ d
        foreach ⟨g_i⟩_{i=1}^d ∈ G^d do
            va_i ← v_i + g_i for 1 ≤ i ≤ d
            V ← ⟨va_i⟩_{i=1}^d
            if V = 0 then
            C[V] ← ⟨⟨⟩, ..., ⟨⟩, ⟨⟩, ⟨⟩⟩
            else if V ∈ D_S^d then
                C[V] ←
                ⟨⟨va_1⟩, ..., ⟨va_d⟩, ⟨0⟩, ⟨0⟩⟩
            else
                V/2 ← ⟨r_i / (2^{x+1} 3^y)⟩_{i=1}^d
                V/3 ← ⟨r_i / (2^x 3^{y+1})⟩_{i=1}^d
                C[V] ←
                FO(V, C[V/2 + G^d], C[V/3 + G^d])
            end
        end
    end
    q ← q − 1;
end
```

3.3. Generalized Algorithm for Multi-Scalar Multiplication

In this subsection, we extend our ideas proposed in Algorithms 3, 4 to our algorithm for multi-scalar multiplication ($Q = r_1 S_1 + \cdots + r_d S_d$).

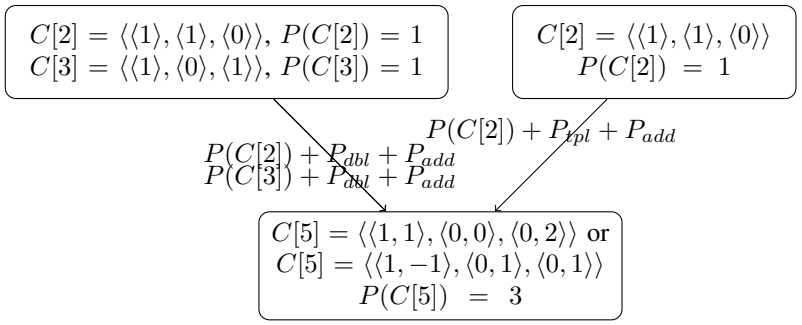

Figure 3. Given $D_S = \{0, \pm 1\}$, we can compute $C[5]$ by three ways. The first way is to compute $C[2]$, and perform a point double and a point addition. The second is to compute $C[3]$, perform a point double, and a point substitution (addition with $-S$). The third is to compute $C[2]$, perform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

The algorithms are shown in Algorithms 5, 6, and the example is shown in Example 4 and Figure 4.

Example 4 Find the optimal chain $C[\langle 7, 9\rangle]$ given $D_S = \{0, 1\}$, $P_{tpl} = P_{dbl} = P_{add} = 1$.

Assume that we are given the optimal chain

$$C[\langle \left\lfloor \frac{7}{2} \right\rfloor, \left\lfloor \frac{9}{2} \right\rfloor \rangle] = C[\langle 3, 4\rangle]$$

and

$$C[\langle \left\lfloor \frac{7}{3} \right\rfloor, \left\lfloor \frac{9}{3} \right\rfloor \rangle] = C[\langle 2, 3\rangle],$$

i.e. we know how to compute $Q_1 = 3S_1 + 4S_2$ and $Q_2 = 2S_1 + 3S_2$ efficiently. It is obvious that there are only two ways to compute the multi-scalar multiplication $Q = 7S_1 + 9S_2$ using DBCs. The first way is to compute Q_1, do point double to $Q_3 = 6S_1 + 8S_2$, and add the point Q_3 with $D = S_1 + S_2$. As we know the optimal chain of Q_1, the cost using this way is

$$P_1 = PJ(C[\langle 7, 9\rangle]) = PJ(C[\langle 3, 4\rangle]) + P_{dbl} + P_{add}.$$

The other way is to compute Q_2, do point triple to $Q_4 = 6S_1 + 9S_2$, and add the point Q_4 with S_1. As we know the optimal chain of Q_2, the cost using this way is

$$P_2 = PJ(C[\langle 7, 9\rangle]) = PJ(C[\langle 2, 3\rangle]) + P_{tpl} + P_{add}.$$

We will show later that

$$PJ(C[\langle 3, 4\rangle]) = PJ(C[\langle 2, 3\rangle]) = 2.$$

Then, $P_1 = P_2 = 2+1+1 = 4$. Hence, both ways are the optimal one, and we can choose any of them. Assume that we select the first way. Given

$$\begin{aligned} C[\langle 3, 4\rangle] &= \langle E_{2,1}, E_{2,2}, X_2, Y_2\rangle \\ &= \langle E_{2,1}, E_{2,2}, \langle x_{2,t}\rangle_{t=0}^{m-1}, \langle y_{2,t}\rangle_{t=0}^{m-1}\rangle \end{aligned}$$

$C[\langle 7, 9\rangle]) = \langle E_1, E_2, X, Y\rangle$ where

$$E_1 = \langle 1, E_{2,1}\rangle,$$

$$E_2 = \langle 1, E_{2,2}\rangle,$$

$$X = \langle 0, x_{2,0} + 1, \ldots, x_{2,m-1} + 1\rangle,$$

$$Y = \langle 0, Y_2\rangle.$$

Next, we find $C[\langle 3, 4\rangle]$. Similar to $C[\langle 7, 9\rangle]$, we can compute $Q_1 = 3S_1 + 4S_2$ in two ways. The first way is to compute $Q_5 = S_1 + 2S_2$, double the point to $2S_1 + 4S_2$, and add the point with S_1 to $3S_1 + 4S_2$. The other way is to compute $Q_6 = S_1 + S_2$, triple the point to $3S_1 + 3S_2$, and add the point with S_2 to $3S_1 + 4S_2$. Assume that we know the optimal way to compute Q_5 and Q_6 with the optimal cost $PJ(C[\langle 1, 2\rangle]) = 2$ and $PJ(C[\langle 1, 1\rangle]) = 0$ (As $Q_6 = D$, which we have already precomputed). The optimal cost to compute Q_1 is

$$\begin{aligned} PJ(C[\langle 3, 4\rangle]) &= PJ(C[\langle 1, 1\rangle]) + P_{tpl} + P_{add} \\ &= 0 + 1 + 1 = 2. \end{aligned}$$

Algorithm 6: Function FO (Used in Algorithm 5)

input : $V = \langle va_i \rangle_{i=1}^{d}$, the optimal double base chain of $\frac{V}{2} + g$ and $\frac{V}{3} + g$ for all $g \in G^d$
output: The optimal double base chain of V, $C[V]$

foreach $U = \langle u_i \rangle_{i=1}^{d} \in D_S^d$ *such that* $va_i - u_i \equiv 0 \bmod 2$ *for all* i **do**
  $VC_{2,U} \leftarrow \langle \frac{va_i - u_i}{2} \rangle_{i=1}^{d}$
  $c_{2,U} \leftarrow PJ(C[VC_{2,U}]) + P_{dbl}$
  if $U \neq \mathbf{0}$ **then** $c_{2,U} \leftarrow c_{2,U} + P_{add}$
end
$c_2 \leftarrow \min_U c_{2,U}$, $U_2 \leftarrow \text{minarg}_U c_{2,U}$
$VC_2 \leftarrow VC_{2,U_2} = \langle E_{2,1}, \ldots, E_{2,d}, X_2, Y_2 \rangle$
foreach $U = \langle u_i \rangle_{i=1}^{d} \in D_S^d$ *such that* $va_i - u_i \equiv 0 \bmod 3$ *for all* i **do**
  $VC_{3,U} \leftarrow \langle \frac{va_i - u_i}{3} \rangle_{i=1}^{d}$
  $c_{3,U} \leftarrow PJ(C[VC_{3,U}]) + P_{tpl}$
  if $U \neq \mathbf{0}$ **then** $c_{3,U} \leftarrow c_{3,U} + P_{add}$
end
$c_3 \leftarrow \min_U c_{3,U}$, $U_3 \leftarrow \text{minarg}_U c_{3,U}$
$VC_3 \leftarrow VC_{3,U_3} = \langle E_{3,1}, \ldots, E_{3,d}, X_3, Y_3 \rangle$
if $U_2 = \mathbf{0}$ *and* $c_2 \leq c_3$ **then**
  $E_i \leftarrow E_{2,i}$ for $1 \leq 1 \leq d$
  $X \leftarrow \langle x_0, \ldots, x_{m-1} \rangle$ where $x_t \leftarrow x_{2,t} + 1$, $Y \leftarrow Y_2$
end
else if $c_2 \leq c_3$ **then**
  $E_i \leftarrow \langle U_{2,i}, E_{2,i} \rangle$ for $1 \leq i \leq d$
  $X \leftarrow \langle 0, x_1, \ldots, x_{m-1} \rangle$ where $x_t \leftarrow x_{2,t-1} + 1$, $Y \leftarrow \langle 0, Y_2 \rangle$
end
else if $U_3 = \mathbf{0}$ **then**
  $E_i \leftarrow E_{3,i}$ for $1 \leq i \leq d$, $X \leftarrow X_3$
  $Y \leftarrow \langle y_0, \ldots, y_{m-1} \rangle$ where $y_t \leftarrow y_{3,t} + 1$
end
else
  $E_i \leftarrow \langle U_{3,i}, E_{3,i} \rangle$ for $1 \leq i \leq d$
  $X \leftarrow \langle 0, X_3 \rangle$
  $Y \leftarrow \langle 0, y_1, \ldots, y_{m-1} \rangle$ where $y_t \leftarrow y_{3,t-1} + 1$
end
$C[V] \leftarrow \langle E_1, \ldots, E_d, X, Y \rangle$

In this example, we use top-down dynamic programming scheme. If we begin with $C[\langle 7, 9 \rangle]$, we need the solutions of

$$C[\langle \left\lfloor \frac{7}{2} \right\rfloor, \left\lfloor \frac{9}{2} \right\rfloor \rangle] = C[\langle 3, 4 \rangle],$$

and

$$C[\langle \left\lfloor \frac{7}{3} \right\rfloor, \left\lfloor \frac{9}{3} \right\rfloor \rangle] = C[\langle 2, 3 \rangle].$$

Then, we need the solution of

$$C[\langle \left\lfloor \frac{3}{2} \right\rfloor, \left\lfloor \frac{4}{2} \right\rfloor \rangle] = C[\langle 1, 2 \rangle]$$

and

$$C[\langle \left\lfloor \frac{3}{3} \right\rfloor, \left\lfloor \frac{4}{3} \right\rfloor \rangle] = C[\langle 1, 1 \rangle]$$

for $C[\langle 3, 4 \rangle]$. However, we use bottom-up dynamic programming algorithm in practice. We begin with the computation of $C[\langle \frac{7}{2^x 3^y}, \frac{9}{2^x 3^y} \rangle]$ for all $x, y \in \mathbb{Z}$ such that $x + y = 3 = \lfloor \lg \max(7, 9) \rfloor$. Then, we proceed to find the solution for $C[\langle \frac{7}{2^x 3^y}, \frac{9}{2^x 3^y} \rangle]$ such that $x+y = 2$ using the solution when $x + y = 3$. After that, we compute the case where $x + y = 1$, i.e. $(x, y) = (1, 0)$ and $(0, 1)$, and we get the optimal DBC when $(x, y) = (0, 0)$. This example is illustrated in Figure 1.

3.4. The Carry Set

As discussed in Section 3, G 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

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

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

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 $(\left\lfloor \frac{r_1}{2} \right\rfloor + c_1)S_1 + \cdots + (\left\lfloor \frac{r_d}{2} \right\rfloor + c_2)S_d$ and $(\left\lfloor \frac{r_1}{3} \right\rfloor + c_3)S_1 + \cdots + (\left\lfloor \frac{r_d}{3} \right\rfloor + c_4)S_d$, 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 G as

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

We propose an algorithm to find G in Algorithm 5 based on breadth-first search scheme. Also, we prove that G is finite set for all finite digit set D_S in Lemma 3.1.

Lemma 3.1 *Given the finite digit set D_S, Algorithm 5 always terminates. And,*

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

when G is the output carry set.

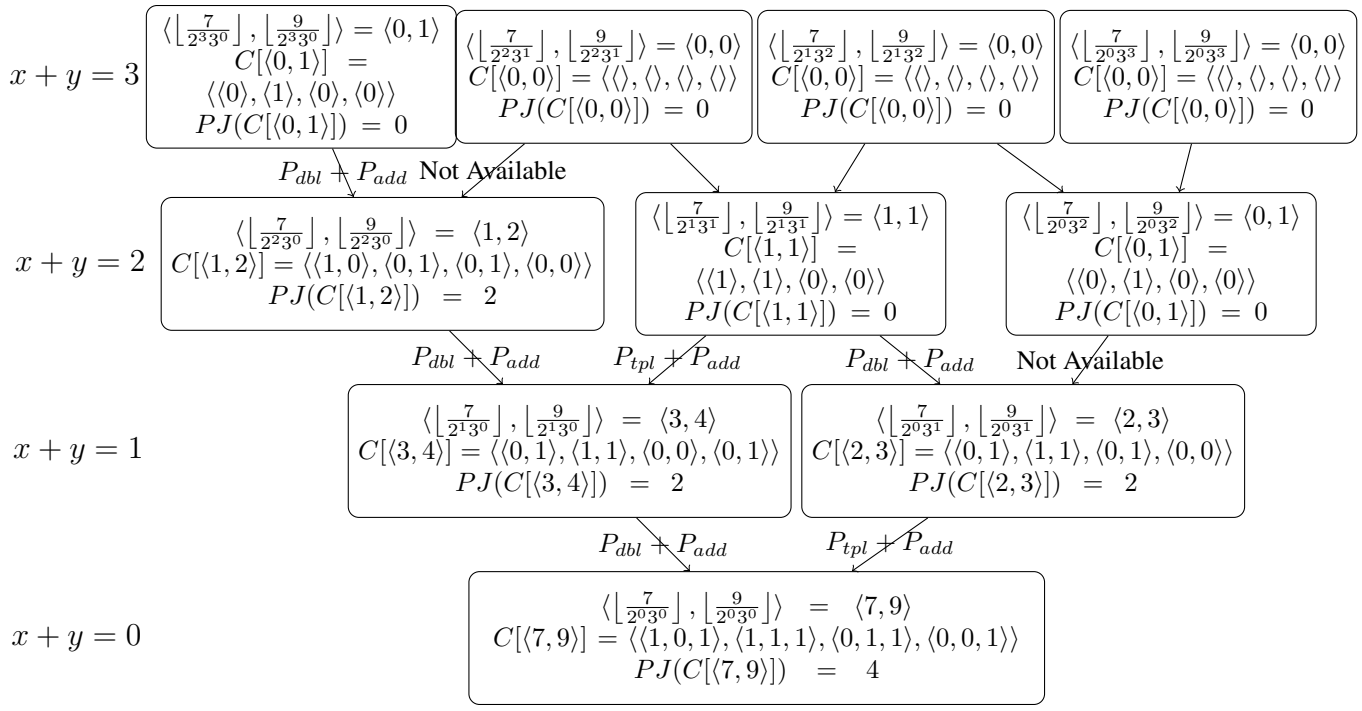

Figure 4. The bottom-up dynamic programming algorithm used for computing the optimal double base chains of $R=\langle7,9\rangle$

Algorithm 7: Find the carry set of the given digit set

input : the digit set D_S
output: the carry set G

$Ct \leftarrow \{0\}, G \leftarrow \oslash$
while $Ct \neq \oslash$ **do**
  Pick $x \in Ct$
  $Ct \leftarrow Ct \cup (\{\frac{x+d}{2} \in \mathbb{Z} | d \in D_S\} - G - \{x\})$
  $Ct \leftarrow Ct \cup (\{\frac{x+d+1}{2} \in \mathbb{Z} | d \in D_S\} - G - \{x\})$
  $Ct \leftarrow Ct \cup (\{\frac{x+d}{3} \in \mathbb{Z} | d \in D_S\} - G - \{x\})$
  $Ct \leftarrow Ct \cup (\{\frac{x+d+1}{3} \in \mathbb{Z} | d \in D_S\} - G - \{x\})$
  $G \leftarrow G \cup \{x\}$
  $Ct \leftarrow Ct - \{x\}$
end

Proof Since

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

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

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

Then,

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

Also,

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

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

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

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. If the computation of field inversion and field squaring are ν and μ times of field multiplication, the computation time in terms of multiplicative factors of $[m]$ is $(\alpha\nu + \mu + \xi)[m]$.

4.1. Results for 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 [17] [18]. 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

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

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 Table 1). 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 2. 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. Results for 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.

Table 1. $P_{dbl}, P_{tpl}, P_{add}, P_{dbl+add}, P_{tpl+add}$ used in the experiment in 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 2. Comparing the computation cost for scalar point multiplication using DBCs 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)	$1369[m]$	$2037[m]$

In our experiment, we use the computation cost $P_{dbl}, P_{tpl}, P_{add}$ as in Table 3 [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 4. 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 5, 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 5. The average running time of Algorithms 3-4 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

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

Table 3. $P_{dbl}, P_{tpl}, P_{add}$ used in the experiment in 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	$3[m]+5[s]$	$7[m]+7[s]$	$10[m]+4[s]$

Table 4. Comparing the computation cost for scalar point multiplication using DBCs 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	$1624.5[m]$	$2168.2[m]$	$2710.9[m]$	$3254.1[m]$

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 DBNS. Instead of using DBC, they cope with the difficulties computing the number system introducing Yao's algorithm. Their results significantly improves the result using the DBC using greedy algorithm, especially the curve where point triple is expensive.

In Tables 6-7, 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 6, and less than 2^{256} in Table 7. We significantly improve the results of [8]. On the other hand, our results do not improve the result of [6] 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.

Recently, there is a work by Longa and Gebotys [19] on several improvements for DBCs. The value $P_{dbl}, P_{tpl}, P_{add}$ they used are smaller than those given in Tables 1 and 3. 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.

4.3. Results for Multi-Scalar Multiplication

In this section, we compare our experimental results with the work by Doche et al. [14]. In the

Table 6. Comparing the computation cost for scalar point muliplication using DBCs 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 7. Comparing the computation cost for scalar point muliplication using DBCs 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	$1989.9[m]$	$2173.5[m]$	$2413.2[m]$	$2319.9[m]$

Table 8. Comparing our results with [19] when the bit number is 160.

Curve Shape	[19]	Our Results
InvEdwards	$1351[m]$	$1337[m]$
Jacobian-3	$1460[m]$	$1454[m]$
ExtJQuartic	$1268[m]$	$1259[m]$

experiment, we are interested on the multi-scalar multiplication when $d = 2$ and $D_S = \{0, \pm 1\}$. There are five experiments shown in Table 9. Again, we randomly select 10000 pairs of positive integers, which are less than 2^{192}, 2^{256}, 2^{384}, 2^{448} in each experiment. Our algorithm improves the tree-based approach by 3.4%, 3.2%, 3.2%, 4.1%, 4.5% when the bit number is 192, 256, 320, 384, 448 respectively.

The computation times of Algorithms 5-6 compared with the existing works is shown in Table 10. Similar to the experiment in the previous subsection, we perform these experiments using Java in Windows Vista, AMD Athlon(tm) 64 X2 Dual Core Processor 4600+ 2.40GHz. As a result, the computation time of our algorithm is only less than a second for 448-bit input. It is shown in [21] that the elliptic curve digital signature algo-

Table 9. Comparing the computation cost for multi scalar multiplication using DBCs when the elliptic curve is implemented in the prime field.

Method	192 bits	256 bits	320 bits	384 bits	448 bits
JSF [20]	$2044[m]$	$2722[m]$	$3401[m]$	$4104[m]$	$4818[m]$
JBT [14]	$2004[m]$	$2668[m]$	$3331[m]$	$4037[m]$	$4724[m]$
Tree-Based [14]	$1953[m]$	$2602[m]$	$3248[m]$	$3938[m]$	$4605[m]$
Our Result	$1886[m]$	$2518[m]$	$3144[m]$	$3777[m]$	$4397[m]$

Table 10. The average running time of Algorithms 5-6 compared with the existing algorithm [14] when $D_S = \{0, \pm 1\}$

Input Size	[14]	Our Result
192 bits	22ms	145ms
256 bits	36ms	275ms
320 bits	45ms	406ms
384 bits	54ms	603ms
448 bits	72ms	817ms

rithm implementing in Java takes much more time than our algorithms, e.g. it takes 2653 ms for 512-bits elliptic curve signature verification. Therefore, the computation is efficient when we use the same scalar for many points on elliptic curve. The results in this paper is also able to use as benchmarks to show how each algorithm is close to the optimality.

We also compare our algorithm in the case that digit set is larger than $\{0, \pm 1\}$. In [14], there is also a result when $Ds = \{0, \pm 1, \pm 5\}$. We compare our result with the result in Table 11. In this case, our algorithm improves the tree-based approach by 5.7%, 5.7%, 5.3%, 6.3%, 6.4% when the bit number is 192, 256, 320, 384, 448 respectively.

Moreover, we observe that the hybrid binary-ternary number system proposed by Adikari et al. [15] is the DBCs for multi-scalar multiplication when $D_S = \{0, \pm 1, \pm 2, 3\}$. Although the conversion algorithm is comparatively fast, there is a large gap between the efficiency of their outputs and the optimal output. We show in Table 12 that the computation cost of the optimal chains are better than the hybrid binary-ternary number system by $10.3 - 11.3\%$.

5. CONCLUSION

In this work, we use the dynamic programming algorithm to present the optimal DBC. 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 DBC. In DBNS, 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 DBCs 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 DBNS. 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.

Table 11. Comparing the computation cost for multi scalar multiplication using DBCs when the elliptic curve is implemented in the prime field, and $D_S = \{0, \pm 1, \pm 5\}$

Method	192 bits	256 bits	320 bits	384 bits	448 bits
Tree-Based [14]	$1795[m]$	$2390[m]$	$2984[m]$	$3624[m]$	$4234[m]$
Our Result	$1692[m]$	$2253[m]$	$2824[m]$	$3395[m]$	$3962[m]$

Table 12. Comparing the computation cost for multi scalar multiplication using DBCs when the elliptic curve is implemented in the prime field, and $D_S = \{0, \pm 1, \pm 2, 3\}$

Method	192 bits	256 bits	320 bits	384 bits	448 bits
Hybrid Binary-Ternary [15]	$1905[m]$	$2537[m]$	$3168[m]$	$3843[m]$	$4492[m]$
Our Result	$1698[m]$	$2266[m]$	$2842[m]$	$3413[m]$	$3986[m]$

[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] C. Doche, D. R. Kohel, and F. Sica, "Double-base number system for multi-scalar multiplications," in *EUROCRYPT 2009*, pp. 502–517, 2009.

[15] J. Adikari, V. Dimitrov, and L. Imbert, "Hybrid binary-ternary number system for elliptic curve cryptosystems," *IEEE Transactions on Computers*, vol. 99, p. to appear, 2010.

[16] T. ElGamal, "A public key cryptosystem and a signature scheme based on discrete logarithms," *IEEE Trans. on Information Theory*, vol. IT-31, pp. 469–472, 1985.

[17] 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.

[18] 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.

[19] 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.

[20] J. A. Solinas, "Low-weight binary representation for pairs of integers," *Centre for Applied Cryptographic Research, University of Waterloo, Combinatorics and Optimization Research Report CORR*, 2001.

[21] R. Stadick, "A Java implementation of the elliptic curve digital signature algorithm using NIST curves over $GF(p)$," Master's thesis, Oregon State University, University Honors College, 2006.