Efficient $\tau$-Adic Sliding Window Method on Elliptic Curve Cryptosystems**

Hiroaki OGURO(++)*, Nonmember and Tetsutaro KOBAYASHI(b), Regular Member

SUMMARY We introduce efficient algorithms for the $\tau$-adic sliding window method, which is a scalar multiplication algorithm on Koblitz curves over $\mathbb{F}_{2^m}$. The $\tau$-adic sliding window method is divided into two parts: the precomputation part and the main computation part. Until now, there has been no efficient way to deal with the precomputation part; the required points of the elliptic curves were calculated one by one. We propose two fast algorithms for the precomputation part. One of the proposed methods decreases the cost of the precomputation part by approximately 30%. Since more points are calculated, the total cost of scalar multiplication is decreased by approximately 7.5%.

**key words: elliptic curve cryptosystem, scalar multiplication, $\tau$-adic sliding window method, precomputation, fast implementation

1. Introduction

Scalar multiplication of a rational point on an elliptic curve accounts for the majority of the cost of elliptic curve cryptosystems. The $\tau$-adic sliding window method [8] is known as an efficient method of employing scalar multiplication algorithms on Koblitz curves [4] over $\mathbb{F}_{2^m}$. The $\tau$-adic sliding window method as well as the conventional window method is divided into two parts. The first part is the precomputation part, which calculates several points in advance before the main computation part is initiated. The second part is the main computation part, which actually performs the scalar multiplication using the precomputed points. If the window size, $w$, is enlarged, the number of required points for the precomputation is increased; consequently, the number of elliptic additions in the main computation part is decreased. However, this causes an exponential increase in the cost of the precomputation itself. Therefore, there is a trade-off in this area. The optimal window size depends on the bit length of scalar $k$ and so on.

Since no efficient method of using the $\tau$-adic sliding window method in the precomputation part has been developed, the required points of elliptic curves must be calculated in a straightforward way, namely one by one. However, when the bit length of keys becomes longer to ensure a higher degree of security, the optimal window size on the scalar multiplication also becomes larger, resulting in an increase in the cost of the precomputation part. The amount increases exponentially with the window size. At this point, a cut in the cost of the precomputation part results in a large contribution toward cutting the total cost of the scalar multiplication.

In this paper, we consider efficient ways to employ the $\tau$-adic sliding window method on Koblitz curves in the precomputation part, and propose two efficient methods.

This paper is organized as follows. We describe the precomputation of the $\tau$-adic sliding window method in Sect. 2 and show the cost of precomputation by conventional methods in Sect. 3. In Sect. 4, we describe the proposed methods and costs. The comparisons of the conventional methods to the proposed methods are evaluated in Sect. 5 and our conclusions are presented in Sect. 6.

2. Precomputation of $\tau$-Adic Sliding Window Method

When calculating $kP$, which is a scalar $k$ multiplication of point $P$, on general elliptic curves using a sliding window method, there are a variety of ways of calculation which differ based on the expanding expression of $k$.

Koblitz curves [4] are the curves on which we can calculate the scalar multiplication on Koblitz curves faster than on general curves by using the window method, which is improved so that it can take advantage of the Frobenius map. As such, there are some methods of applying the $\tau$-adic sliding window method after representing $k$ as the unsigned or signed $\tau$-adic expansion [4] or $\tau$-adic NAF [8]. The number of required points for precomputation depends on the type of the expansion of $k$, and is shown below where $w$ represents the
window size.

(a) Unsigned $\tau$-adic expansion

The required points for precomputation are:

\[
\begin{align*}
\alpha_1 P &= P, \\
\alpha_3 P &= (\tau + 1)P, \\
\alpha_5 P &= (\tau^2 + 1)P, \\
\alpha_7 P &= (\tau^2 + \tau + 1)P, \\
\vdots \\
\alpha_{2^w-1} P &= (\tau^{w-1} + \cdots + \tau + 1)P,
\end{align*}
\]

where $\alpha_i$ represents the polynomial of $\tau$ whose vector representation is the same as the unsigned binary expansion of $i$. The number of the required points for precomputation is $B(w) = 2^{w-1}$.

(b) Signed $\tau$-adic expansion

For integer $l (\geq 2)$, let $S_l$ be the set of vectors such that

\[
S_l = \{1\} \times \{0, 1, -1\}^{l-2} \times \{1, -1\}.
\]

Let $T_l$ be the set of polynomials of $\tau$ whose element’s vector representation is $S_l$. Then, the required points for precomputation are $P$ and points multiplied by all elements of $\bigcup_{l=2}^{w} T_l$, namely,

\[
P, \\
(\tau + 1)P, \\
(\tau - 1)P, \\
(\tau^2 + 1)P, \\
(\tau^2 - 1)P, \\
(\tau^2 + \tau + 1)P, \\
(\tau^2 + \tau - 1)P, \\
(\tau^2 - \tau + 1)P, \\
(\tau^2 - \tau - 1)P, \\
\vdots
\]

The number of required points for precomputation is $B_S(w) = 3^{w-1}$.

(c) $\tau$-adic NAF

The required points for precomputation are:

\[
\begin{align*}
\beta_1 P &= P, \\
\beta_3 P &= (\tau^2 - 1)P, \\
\beta_5 P &= (\tau^2 + 1)P, \\
\beta_7 P &= (\tau^3 - 1)P, \\
\beta_9 P &= (\tau^3 + 1)P, \\
\beta_{11} P &= (\tau^4 - \tau^2 - 1)P, \\
\vdots \\
\beta_{2^w - C(w) - 1} P &= \left\{\begin{array}{ll}
(\tau^{w-1} + \cdots + \tau^3 + 1)P, & (w: \text{even}) \\
(\tau^{w-1} + \cdots + \tau^2 + 1)P, & (w: \text{odd})
\end{array}\right.
\]
\]

where $\beta_i$ represents the polynomial of $\tau$ whose vector representation is the same as the bit string of NAF($i$). NAF (non-adjacent form) [3, 8]–[10] is the signed binary expansion with the property in which no two consecutive coefficients are nonzero. $C(w)$ is the number of required points for precomputation, including $\beta_1 P = P$, and is represented by

\[
C(w) = \frac{1}{3} (2^w - (-1)^w).
\]

Summarizing (a) to (c) above, the number of required points for precomputation is given in Table 1.

<table>
<thead>
<tr>
<th>Expansion for $k$</th>
<th>Number of points</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a) Unsigned $\tau$-adic expansion</td>
<td>$B(w) = 2^{w-1}$</td>
</tr>
<tr>
<td>(b) Signed $\tau$-adic expansion</td>
<td>$B_S(w) = 3^{w-1}$</td>
</tr>
<tr>
<td>(c) $\tau$-adic NAF</td>
<td>$C(w) = (2^w - (-1)^w)/3$</td>
</tr>
</tbody>
</table>

### Table 1: The number of required points for precomputation.

#### 3. Conventional Methods

3.1 Conventional Method Using Affine Coordinates

Usually, in precomputation on the $\tau$-adic sliding window method with $\tau$-adic NAF in Sect. 2, we calculated the required points in a straightforward way, namely one by one. The cost for this was $C(w) - 1$ elliptic additions and $w - 1$ multiplications by $\tau$ [8]. The cost of an elliptic addition using affine coordinates is $I + 2M + S$, where $I$ represents the cost of an inversion on $\mathbb{F}_{2^m}$, $M$ represents the cost of a multiplication on $\mathbb{F}_{2^m}$, $S$ represents the cost of a square on $\mathbb{F}_{2^m}$, and the cost of multiplication by $\tau$ is $2S$.

We can reduce the number of inversions to be calculated in the precomputation to half since both elliptic addition $P + Q$ and elliptic subtraction $P - Q$ using affine coordinates require the same inversion for each calculation. For example, when we calculate both $\beta_3 P = \tau^2 P - P$ and $\beta_5 P = \tau^3 P + P$, only one calculation of the inversion is enough to do.

Therefore, the cost of precomputation using the conventional method with affine coordinates is represented by
4.1.2 NAF Polynomial

3.2 Conventional Method Using Projective Coordinates

Using projective coordinates, the cost of an elliptic addition on Koblitz curves is $13M + 6S$ and the cost of multiplication by $\tau$ is $3S$. Then, the cost of precomputation using the conventional method with projective coordinates is represented by

$$13 \cdot \left\{ \frac{2^w}{3} - \frac{(-1)^w}{3} - 1 \right\} M + \left\{ \frac{2^{w+1}}{3} - 2 \cdot (-1)^w + 3w - 9 \right\} S.$$  (2)

4. Proposed Methods

4.1 Fast Precomputation Using Affine Coordinates

In this subsection, we propose a fast precomputation method using affine coordinates. In Sects. 4.1.1 and 4.1.2, we present two basic tools used in the method. Next, we describe the first proposed method in Sect. 4.1.3.

4.1.1 Montgomery Trick of Simultaneous Inversions

The Montgomery trick of simultaneous inversions [1] is a method that simultaneously calculates inversions of $n$ elements in a field. When $I > 3M$, this method calculates $n$ inversions faster than calculating them one by one. Figure 1 shows the algorithm of the Montgomery trick of simultaneous inversions. The cost is $I + 3(n - 1)M$.

4.1.2 NAF Polynomial

Definition 1 (NAF polynomial): Let $i$ be a positive integer. The polynomial whose vector representation is the same as the bit string of NAF$(i)$ is called a NAF polynomial, and is denoted by $NAFP_i(t) \in \mathbb{Z}[t]$.

Example 1: Since NAF(19) = $(1, 0, 1, 0, -1)$, then $NAFP_{19}(t) = P + t$.  
$\beta_i$ denoted in Sect. 2 is equal to $NAFP_i(\tau)$. The NAF polynomial has the following properties.

Lemma 1 (Property of NAF polynomial): Let $i$ be an integer greater than or equal to 2. For integer $j$ such that $1 \leq j \leq \frac{1}{3} \{2^i - 1 - (i \mod 2)\}$,

$$NAFP_{2^i}(t) \pm NAFP_j(t) = NAFP_{2^i \pm j}(t),$$

where a double sign is the same order.

Intuitively explaining Lemma 1, let $G_i$ be the set of all NAF polynomials whose length is $l$, then $G_{i+2}$ is constructed by addition-subtraction of a vector $(1, 0, 0, \ldots, 0)$ whose length is $l + 2$ and all elements of $\bigcup_{i=1}^{l} G_i$.

4.1.3 Proposed Method 1

Figure 2 shows a fast method for calculating the precomputation of (c) in Sect. 2 using affine coordinates.

We call this proposed method 1. Proposed method 1 utilizes the Montgomery trick of simultaneous inversions (Sect. 4.1.1) and the properties of the NAF polynomial (Sect. 4.1.2). Roughly speaking, proposed method 1 is executed as follows:

[Proposed method 1]

Step 1: Add $\beta_2 P = P$ to the table.
Step 2: Calculate $T_i = \tau^i P$ for all $i$, and store them to memory. (Thus, $T_i = \beta_2 P$.)

[Montgomery trick of simultaneous inversions]

Input: $a_1, a_2, \ldots, a_n \in \mathbb{F}_q$.
Output: $b_1, b_2, \ldots, b_n \in \mathbb{F}_q$, where $a_i b_i = 1$.
1. $c_1 \leftarrow a_1$.
2. For $i$ from 2 to $n$ do: $c_i \leftarrow c_{i-1} \cdot a_i$.
3. $u \leftarrow c_1^{w-1}$.
4. For $i$ from $n$ downto 2 do:
   4.1 $b_i \leftarrow u \cdot c_{i-1}$.
   4.2 $u \leftarrow u \cdot a_i$.
5. $b_1 \leftarrow u$.
6. Return $\{b_i : 1 \leq i \leq n\}$.

Fig. 1 Montgomery trick of simultaneous inversions.

[Proposed method 1]

Input: $P$ (affine point), $w$ (window size, $w \geq 3$).
Output: $\{P_{2i-1} = \beta_{2i-1} P : 1 \leq i \leq C(w)\}$ (affine points).
1. $P_1 \leftarrow P$.
2. $T_1 \leftarrow \tau P$.
3. For $i$ from 2 to $w - 1$ do: $T_i \leftarrow \tau T_{i-1}$.
4. For $i$ from 1 to $\left\lfloor \frac{w - 1}{2} \right\rfloor$ do:
   (Calculate inversions simultaneously)
   4.1 For $j$ from 2 to $w - 1$ do:
      4.1.1 For $k$ from $C(2(i-1)) + 1$ to $C(2i)$ do:
         4.1.1.1 If $-P_{i} \land (P_{2i} \lor P_{P3})$, then do:
         4.1.1.1.1 $P_{2i} + k \leftarrow T_j + P_{2k-1}$.
         4.1.1.1.2 $P_{2i} - k \leftarrow T_j - P_{2k-1}$.
      5. Return $\{P_{2i-1} : 1 \leq i \leq C(w)\}$.

$P_1 : k > C(2(i - 1) + 1)$
$P_2 : 2(i + 1) > w$
$P_3 : (i = 2) \land (j = 4)$

Fig. 2 Fast precomputation using affine coordinates.
Step 3: Calculate

\[ \beta_{2^i \pm 1} P = T_i \pm P \ (i = 2, 3, \ldots, w - 1), \]

where a double sign is the same order, and add them to the table. In this operation, we use the same inversion in calculation of \( T_i + P \) and \( T_i - P \), and apply the Montgomery trick of simultaneous inversions to inversions in all elliptic additions or subtractions.

Step 4: By using \( \beta_3 P, \beta_5 P, \beta_7 P, \) and \( \beta_9 P \) which were calculated in Step 3, calculate

\[ \beta_{2^i \pm 3} P = T_i \pm \beta_3 P \ (i = 4, 5, \ldots, w - 1), \]
\[ \beta_{2^i \pm 5} P = T_i \pm \beta_5 P \ (i = 4, 5, \ldots, w - 1), \]
\[ \beta_{2^i \pm 7} P = T_i \pm \beta_7 P \ (i = 5, 6, \ldots, w - 1), \]
\[ \beta_{2^i \pm 9} P = T_i \pm \beta_9 P \ (i = 5, 6, \ldots, w - 1), \]

where a double sign is the same order, and add them to the table. In this operation, we use the same inversion in calculation of \( T_i + \beta_3 P \) and \( T_i - \beta_3 P \), and apply the Montgomery trick of simultaneous inversions to inversions in all elliptic additions or subtractions.

Step 5: Similarly, apply the Montgomery trick of simultaneous inversions while calculating inversions required for elliptic addition and subtraction of \( T_i \) and points which have already been recorded to the table. Then complete the table.

Proposed method 1, first calculates \( \tau^i P = \beta_{2^i} P \ (i = 2, 3, \ldots, w - 1) \) for all \( i \). These values can be calculated quickly because the cost of multiplication by \( \tau \) is very small. Next, set these points as standard points, and apply the Montgomery trick of simultaneous inversions while calculating inversions required for elliptic addition and subtraction of the standard points and points which have already been recorded to the table.

Figure 3 shows the view of precomputation by proposed method 1 when \( w = 7 \). In Fig. 3, we use the same inversion for calculation of the two elements pointed by double-headed arrows. Italic numbers which bundle the arrows mean the order of applying Montgomery trick of simultaneous inversions. For example, Fig. 3 requires three Montgomery tricks of simultaneous inversions.

Now, we consider the cost of proposed method 1. Let \( D(w) = C(w) - C(w - 1) \) be the number of points \( \beta_i P \) whose multiplier \( \beta_i \) of which degree is \( w - 1 \). \( D(w) \) means the number of elements in the groups divided by solid lines in Fig. 3. Let \( m_k \) be the number of inversions which is applied to the \( k \)-th Montgomery trick of simultaneous inversions. Generally, \( m_k \) is represented by

\[ m_k = D(2k - 1) \cdot (w - 2k) + D(2k) \cdot (w - 2k - 1). \]

For example, in Fig. 3, \( m_1 = 5, m_2 = 10, m_3 = 6. \) Since the cost of an elliptic addition using affine coordinates is \( I + 2M + S \), the cost of multiplication by \( \tau \) is \( 2S \), and \( n \) inversions can be calculated in the cost \( I + 3(n - 1)M \) by Montgomery trick of simultaneous inversions, then we require \( \left\lceil \frac{w-1}{2} \right\rceil \) inversions, \( \sum_{k=1}^{\left\lceil \frac{w-1}{2} \right\rceil} 3(m_k - 1) + 2\{C(w - 1) \}

\[ + \frac{2}{3}(2^w - (-1)^w - 2) \]
4.2 Fast Precomputation Using Projective Coordinates

In this subsection, we propose a fast method of precomputation using projective coordinates. First, we show a basic tool used in the method in Sect. 4.2.1. Next, we describe the second proposed method in Sect. 4.2.2.

4.2.1 Simultaneous Elliptic Addition-Subtraction

Elliptic additions to elliptic curves over \( \mathbb{F}_{2^m} \) using projective coordinates [6], which are represented by \((X,Y,Z) \ (x = X/Z, y = Y/Z^2)\), are,

**Elliptic additions using projective coordinates**

\[
(X_0,Y_0,Z_0) + (X_1,Y_1,Z_1) = (X_2,Y_2,Z_2),
\]

\[
A_0 = Y_1 \cdot Z_0^2, \quad G = D^2 \cdot (F + a \cdot E^2),
\]

\[
A_1 = Y_0 \cdot Z_1^2, \quad H = C^* \cdot F,
\]

\[
B_0 = X_1 \cdot Z_0, \quad X_2 = C^* + H + G,
\]

\[
B_1 = X_0 \cdot Z_1, \quad I = D^2 \cdot B_0 \cdot E + X_2,
\]

\[
C = A_0 + A_1, \quad Z_2 = F^2,
\]

\[
D = B_0 + B_1, \quad J = D^2 \cdot A_0 + X_2,
\]

\[
E = Z_0 \cdot Z_1, \quad Y_2 = H \cdot I + Z_2 \cdot J.
\]

\[
F = D \cdot E,
\]

The cost is \(13M + 6S\) since \(a = 0\) or \(a = 1\) on Koblit curves, although it is \(14M + 6S\) on general curves.

As for elliptic subtraction, if we first generate \(-P(X,Y,Z) = (X,XZ + Y,Z)\) and substitute it for the formula of elliptic addition, a higher cost of generating the \(Y\) coordinate, \(1 \cdot M\), arises. However, suitable reforming of intermediate formulae enables us to calculate it with the same cost as the elliptic addition, namely \(13M + 6S\).

Now, we consider calculating elliptic addition \(P + Q\) and elliptic subtraction \(P - Q\) consecutively using projective coordinates. Since each elliptic addition and subtraction includes some common intermediate formulae, these consecutive calculations can be performed quickly by sharing them.

Figure 4 shows the algorithm for simultaneously calculating the elliptic addition and subtraction, \(P \pm Q\). The conventional method, which calculates elliptic addition and subtraction consecutively, costs \(2(13M + 6S) = 26M + 12S\). On the other hand, our method shown in Fig. 4 costs only \(16M + 7S\). Moreover, if \(Z_0 = 1\) and \(Z_1 \neq 1\), namely we calculate \(A+P = P\), where \(A\) represents the affine coordinates and \(P\) represents the projective coordinates, the cost is decreased to \(13M + 6S\). If \(Z_0 = 1\) and \(Z_1 = 1\), namely we calculate \(A+\mathcal{A} = \mathcal{P}\), the cost is decreased to \(10M + 4S\).

Similarly, we can also apply the idea of the simultaneous elliptic addition-subtraction to elliptic curves over \(\mathbb{F}_{p^m}\), where \(p > 3\) is a prime and \(m\) is a positive integer, as well as \(\mathbb{F}_{2^m}\). As an example, we show the case of applying to elliptic curves using projective coordinates over \(\mathbb{F}_{p^m}\) in Appendix. In this case, it costs only \(15M + 3S\) while the conventional method costs \(2(12M + 2S) = 24M + 4S\).

4.2.2 Proposed Method 2

Figure 5 shows a fast method to calculate the precomputation of (c) in Sect. 2 using projective coordinates. We call this proposed method 2. Proposed

**[Proposed method 2]**

Input: \(P\) (affine point), \(w\) (window size, \(w \geq 3\)).

Output: \(\{P_{2i-1} = \beta_{2i-1}P : 1 \leq i \leq C(w)\}\) (projective points).

1. \(P_1 \leftarrow P\).
2. \(T_1 \leftarrow \tau P\).
3. For \(i\) from 2 to \(w - 1\) do: \(T_i \leftarrow \tau T_{i-1}\).
4. For \(i\) from 1 to \(\left\lfloor \frac{w - 1}{2} \right\rfloor\) do:
   4.1 For \(j\) from 2 to \(w - 1\) do:
      4.1.1 For \(k\) from \(C(2(i - 1)) + 1\) to \(C(2i))\) do:
         4.1.1.1 If \(-P1 \land P2 \lor P3\), then do:
            (Calculate addition-subtraction simultaneously)
            4.1.1.1.1 \(P_{2i+j-2(k-1)} \leftarrow T_j + P_{2k-1}\).
            4.1.1.1.2 \(P_{2i+j-2(k-1)} \leftarrow T_j - P_{2k-1}\).
5. Return \(\{P_{2i-1} : 1 \leq i \leq C(w)\}\).
method 2 utilizes the properties of the NAF polynomial (Sect. 4.1.2) and simultaneous elliptic addition-subtraction (Sect. 4.2.1). Proposed method 2 is similar to proposed method 1. It replaces elliptic addition and subtraction with the Montgomery trick of simultaneous inversions in Step 4 in proposed method 1 by simultaneous elliptic addition-subtraction. Roughly speaking, proposed method 2 is executed as follows:

[Proposed method 2]

Step 1: Add $\beta_1P = P$ represented by affine coordinates to the table.

Step 2: Calculate $T_i = \tau^i P (i = 2, 3, \ldots, w - 1)$ represented by affine coordinates for all $i$, and store them to memory. (Thus, $T_i = \beta_2^i P$)

Step 3: Calculate

$$\beta_{2^i+1} P = T_i \pm P (i = 2, 3, \ldots, w - 1),$$

where a double sign is the same order, and add them to the table. In this operation, apply simultaneous elliptic addition-subtraction with $A + \mathcal{A} = \mathcal{P}$ to each calculation.

Step 4: By using $\beta_3 P, \beta_5 P, \beta_7 P$, and $\beta_9 P$ which were calculated in Step 3 and represented by projective coordinates, calculate

$$\beta_{2^i+3} P = T_i \pm 3 P (i = 4, 5, \ldots, w - 1),$$

$$\beta_{2^i+5} P = T_i \pm 5 P (i = 4, 5, \ldots, w - 1),$$

$$\beta_{2^i+7} P = T_i \pm 7 P (i = 5, 6, \ldots, w - 1),$$

$$\beta_{2^i+9} P = T_i \pm 9 P (i = 5, 6, \ldots, w - 1),$$

where a double sign is the same order, and add them to the table. In this operation, apply simultaneous elliptic addition-subtraction with $A + \mathcal{P} = \mathcal{P}$ to each calculation.

Step 5: Similarly, apply simultaneous elliptic addition-subtraction with $A + \mathcal{P} = \mathcal{P}$ to each calculation while calculating simultaneous elliptic addition-subtraction of $T_i$ and points which have already been recorded to the table. Then complete the table.

The cost of proposed method 2 is represented by

$$\left[\frac{13}{6} \{2^w - (-1)^w\} - 3w - \frac{1}{2}\right] M$$

$$+ \{2^w - (-1)^w - 1\} S.$$  (4)

5. Comparisons and Consideration

5.1 Comparison Based on Affine Coordinates

In this subsection, we compare the conventional and proposed methods based on affine coordinates and evaluate them. From Formulae (1) and (3), we can derive Table 2, which shows each cost of precomputation using the affine coordinates. Besides, by applying the data shown in [3] which are actually measured values of field operations over $\mathbb{F}_{2^m}$ on a Pentium II 400MHz computer, we can derive Fig. 6 when $m = 163$. In Fig. 6, taking into consideration that the average density of $\tau$-adic NAF is $(w + 1)^{-1}$, we set the average cost of the main computation part as

$$\frac{m}{w + 1} ECADD + (m - w) TAU,$$

where $ECADD$ represents the cost of an elliptic addition, and $TAU$ represents the cost of multiplication by $\tau$.

Figure 6 shows that proposed method 1 decreases the cost of the precomputation part by approximately 30%. This effect causes the optimal window size to change from 5 to 6. Consequently, it decreases the total cost of the scalar multiplication by approximately 7.5%.

<table>
<thead>
<tr>
<th>$w$</th>
<th>Conventional</th>
<th>Proposed Method 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>1 + 4M + 6S</td>
<td>1 + 4M + 6S</td>
</tr>
<tr>
<td>4</td>
<td>2I + 8M + 10S</td>
<td>1 + 11M + 10S</td>
</tr>
<tr>
<td>5</td>
<td>5I + 20M + 18S</td>
<td>2I + 29M + 18S</td>
</tr>
<tr>
<td>6</td>
<td>10I + 40M + 30S</td>
<td>2I + 64M + 30S</td>
</tr>
<tr>
<td>7</td>
<td>21I + 84M + 54S</td>
<td>3I + 138M + 54S</td>
</tr>
<tr>
<td>8</td>
<td>42I + 168M + 98S</td>
<td>3I + 285M + 98S</td>
</tr>
</tbody>
</table>

Table 2 Cost comparison of precomputation using affine coordinates.

Fig. 6 Comparison of the total cost using affine coordinates ($m = 163$).
In this subsection, we compare the conventional and proposed methods based on projective coordinates. In the same way when using affine coordinates (Sect. 5.1), we can derive Fig. 7 using projective coordinates. In this subsection, we compare the conventional and proposed methods based on projective coordinates. In the same way when using affine coordinates (Sect. 5.1), we can derive Fig. 7 using projective coordinates. In the same way when using affine coordinates (Sect. 5.1), we can derive Fig. 7 when $m = 163$.

Figure 7 shows that proposed method 2 decreases the cost of the precomputation part by approximately 55%. This effect causes the optimal window size to change from 4 to 5. Consequently, it decreases the total cost of the scalar multiplication by approximately 12%.

5.3 Consideration of Application to Other Methods

In recent years, other kinds of $\tau$-adic sliding window methods have been proposed such as the $\tau$-adic width-$w$ window method [3], [9], [10]. This method expands $k$ by width-$w$ $\tau$-adic NAF [3], [9], [10]. In this method, the required points for precomputation are $\gamma_i P$ ($i = 1, 3, 5, \ldots, 2^{w-1} - 1$), where $\gamma_i = i \mod \tau^w$. The number of these points is $B(w-1) = 2^{w-2}$, which is half of the original $\tau$-adic sliding window method [8].

Although it is remarkable that the number of precomputation points becomes half, the number of inversions does not become exactly half of $B(w-1) - 1$. Because we can not apply the property that both elliptic addition $P + Q$ and subtraction $P - Q$ using affine coordinates require the same inversion to all precomputation points in this case. For example, when $w = 5$, we calculate as follows:

$$\gamma_1 P = P,$$

$$\gamma_3 P = \left(\tau^2 - 1\right) P,$$

$$\gamma_5 P = \left(\tau^2 + 1\right) P,$$

$$\gamma_7 P = \left(-\tau^3 - 1\right) P,$$

$$\gamma_9 P = \left(-\tau^3 - \tau^3 + 1\right) P = \left(-\tau^3 \gamma_5 + 1\right) P,$$

$$\gamma_{11} P = \left(-\tau^4 - \tau^2 - 1\right) P = \left(-\tau^2 \gamma_5 - 1\right) P,$$

$$\gamma_{13} P = \left(-\tau^4 - \tau^2 + 1\right) P = \left(-\tau^2 \gamma_5 + 1\right) P,$$

$$\gamma_{15} P = \left(\tau^4 - 1\right) P.$$

Still, we can partially apply our basic idea mentioned in Sect. 4 to the $\tau$-adic width-$w$ window method. However, it is difficult to describe the algorithm systematically because the optimal order of the precomputation becomes so complicated when the window size $w$ becomes larger. Then, as an example, we show the case when $w = 5$. Using affine coordinates, we first calculate $\tau^i P$ ($i = 1, 2, 3, 4$), and apply the Montgomery trick of simultaneous inversions to $\{\gamma_i P, \gamma_5 P\}, \gamma_7 P$ and $\gamma_{15} P$.

Next, we calculate $\tau^i \gamma_5 P$ ($i = 1, 2, 3$), and apply the Montgomery trick of simultaneous inversions to $\gamma_9 P$ and $\{\gamma_{11} P, \gamma_{13} P\}$. This costs $2\tau + 19M + 19S$ while the way not using our idea costs $5I + 14M + 21S$. When $m = 163$, we can decrease the cost of the precomputation part by approximately 38%, and decrease the total cost by approximately 5.8%. Similarly, using projective coordinates, the way using our idea costs $44M + 37S$ while the way not using it costs $51M + 41S$. When $m = 163$, we can decrease the cost of the precomputation part by approximately 13%, and decrease the total cost by approximately 1.5%.

6. Conclusions

We presented two fast algorithms for the precomputation part of the $\tau$-adic sliding window method, a scalar multiplication algorithm on Koblitz curves over $\mathbb{F}_{2^m}$. The first is proposed method 1 using affine coordinates, and the other is proposed method 2 using projective coordinates. Proposed methods 1 and 2 decrease of the
cost of the precomputation part by approximately 30% and 55%, respectively. When \( m = 163 \), for proposed methods 1 and 2, this effect causes the respective optimal window size to change from 5 to 6 and 4 to 5; consequently, we can decrease the total cost of the scalar multiplication by approximately 7.5% and 12%, respectively.

References


Appendix: Simultaneous Elliptic Addition-Subtraction over \( \mathbb{F}_{p^m} \)

In this appendix, we show the simultaneous elliptic addition-subtraction using projective coordinates over \( \mathbb{F}_{p^m} \), where \( p > 3 \) is a prime and \( m \) is a positive integer. Elliptic curves over \( \mathbb{F}_{p^m} \) using projective coordinates \((X, Y, Z)\) are given by

\[
E_p : Y^2Z = X^3 + aXZ^2 + bZ^3,
\]

where \( a, b \in \mathbb{F}_p, x = X/Z, y = Y/Z \), and elliptic curves over \( \mathbb{F}_{p^m} \) using affine coordinates \((x, y)\) are given by

\[
y^2 = x^3 + ax + b.
\]

The simultaneous elliptic addition-subtraction to elliptic curves using projective coordinates over \( \mathbb{F}_{p^m} \) is calculated as follows:

\[
\begin{align*}
(X_0, Y_0, Z_0) + (X_1, Y_1, Z_1) &= (X_2^+, Y_2^+, Z_2^+) \\
(X_0, Y_0, Z_0) - (X_1, Y_1, Z_1) &= (X_2^-, Y_2^-, Z_2^-)
\end{align*}
\]

Input : \( X_0, Y_0, Z_0, X_1, Y_1, Z_1 \)
Output : \( X_2^+, Y_2^+, Z_2^+ \), \( X_2^−, Y_2^−, Z_2^− \)

\[
\begin{align*}
S_1 &= X_0Z_1, & A &= u^2 \cdot C + B, \\
S_2 &= X_1Z_0, & X_2^+ &= v \cdot A, \\
T_1 &= Y_0Z_1, & Y_2^+ &= u \cdot (E - A) + D, \\
T_2 &= Y_1Z_0, & Z_2^+ &= v^3 \cdot C, \\
u &= T_2 - T_1, & u' &= -T_2 - T_1, \\
v &= S_2 - S_1, & A' &= u^2 \cdot C + B, \\
C &= Z_0Z_1, & X_2^− &= v \cdot A', \\
D &= -v^3 \cdot T_1, & Y_2^− &= u' \cdot (E - A') + D, \\
E &= v^2 \cdot S_1, & Z_2^− &= Z_2^+, \\
B &= -v^3 - 2E
\end{align*}
\]