# Fast Simultaneous Scalar Multiplication on Elliptic Curve with Montgomery Form

Toru Akishita

Sony Corporation, 6-7-35 Kitashinagawa Shinagawa-ku, Tokyo, 141-0001, Japan
akishita@pal.arch.sony.co.jp

**Abstract.** We propose a new method to compute $x$-coordinate of $kP + lQ$ simultaneously on the elliptic curve with Montgomery form over $\mathbb{F}_p$ without precomputed points. To compute $x$-coordinate of $kP + lQ$ is required in ECDSA signature verification. The proposed method is about 25% faster than the method using scalar multiplication and the recovery of $Y$-coordinate of $kP$ and $lQ$ on the elliptic curve with Montgomery form over $\mathbb{F}_p$, and also slightly faster than the simultaneous scalar multiplication on the elliptic curve with Weierstrass form over $\mathbb{F}_p$ using NAF and mixed coordinates. Furthermore, our method is applicable to Montgomery method on elliptic curves over $\mathbb{F}_{2^n}$.

## 1 Introduction

Elliptic curve cryptography was first proposed by Koblitz [10] and Miller [15]. In recent years, efficient algorithms and implementation techniques of elliptic curves over $\mathbb{F}_p$ [3,4], $\mathbb{F}_{2^n}$ [7,13] and $\mathbb{F}_{p^n}$ [2,9,12] has been investigated. In particular, the scalar multiplication on the elliptic curve with Montgomery form over $\mathbb{F}_p$ can be computed efficiently without precomputed points [16], and is immune to timing attacks [11,17]. This method is extended to elliptic curves over $\mathbb{F}_{2^n}$ [14].

We need to compute $kP + lQ$, where $P$ and $Q$ are points on the elliptic curve and $k, l$ are integers less than the order of the base point, in Elliptic Curve Digital Signature Algorithm (ECDSA) signature verification [1]. On the elliptic curve with Weierstrass form, $kP + lQ$ can be efficiently computed by a simultaneous multiple point multiplication [3,7,19], which we call a *simultaneous scalar multiplication*. On the other hand, the simultaneous scalar multiplication on the elliptic curve with Montgomery form has not been proposed yet. Then we propose it and call it *Montgomery simultaneous scalar multiplication*. This method is about 25% faster than the method using Montgomery scalar multiplication and the recovery of $Y$-coordinate of $kP$ and $lQ$, and about 1% faster than Weierstrass simultaneous scalar multiplication over $\mathbb{F}_p$ using NAF [8] and mixed coordinates [4]. Moreover, our method is applicable to elliptic curves over $\mathbb{F}_{2^n}$.

This paper is described as follows. Section 2 presents preliminaries including arithmetic over the elliptic curve with Montgomery form and Weierstrass form. In Section 3, we describe the new method, Montgomery simultaneous scalar

S. Vaudenay and A. Youssef (Eds.): SAC 2001, LNCS 2259, pp. 255–267, 2001.

multiplication. Section 4 presents comparison of our method with others and Section 5 presents implementation results. We then apply our method to elliptic curves over $\mathbb{F}_{2^n}$ in Section 6 and conclude in Section 7.

## 2 Preliminaries

### 2.1 Elliptic Curve with Montgomery Form

Montgomery introduced the new form of elliptic curve over $\mathbb{F}_p$ [16]. For $A, B \in \mathbb{F}_p$, the elliptic curve with Montgomery form $E_M$ is represented by

$$E_M : By^2 = x^3 + Ax^2 + x \quad ((A^2 - 4)B \neq 0). \tag{1}$$

We remark that the order of any elliptic curve with Montgomery form is always divisible by 4.

In affine coordinates $(x, y)$, the $x$-coordinate of the sum of the two points on $E_M$ can be computed without the $y$-coordinates of these points if the difference between these points is known. Affine coordinates $(x, y)$ can be transformed into projective coordinates $(X, Y, Z)$ by $x = X/Z, y = Y/Z$. Equation (1) can also be transformed as

$$E_M : BY^2Z = X^3 + AX^2Z + XZ^2.$$

Let $P_0 = (X_0, Y_0, Z_0)$ and $P_1 = (X_1, Y_1, Z_1)$ be points on $E_M$ and $P_2 = (X_2, Y_2, Z_2) = P_1 + P_0$, $P_3 = (X_3, Y_3, Z_3) = P_1 - P_0$. Addition formulas and doubling formulas are described as follows.

**Addition formulas** $P_2 = P_1 + P_0$ $(P_1 \neq P_0)$

$$\begin{aligned} X_2 &= Z_3((X_0 - Z_0)(X_1 + Z_1) + (X_0 + Z_0)(X_1 - Z_1))^2 \\ Z_2 &= X_3((X_0 - Z_0)(X_1 + Z_1) - (X_0 + Z_0)(X_1 - Z_1))^2 \end{aligned} \tag{2}$$

**Doubling formulas** $P_2 = 2P_0$

$$\begin{aligned} 4X_0Z_0 &= (X_0 + Z_0)^2 - (X_0 - Z_0)^2 \\ X_2 &= (X_0 + Z_0)^2(X_0 - Z_0)^2 \\ Z_2 &= (4X_0Z_0)((X_0 - Z_0)^2 + ((A + 2)/4)(4X_0Z_0)) \end{aligned}$$

$P_2 = (X_2, Z_2)$ can be computed without $Y$-coordinate. Since the computational cost of a field addition and subtraction is much lower than that of a field multiplication and squaring, we can ignore it. The computational cost of the addition formulas is $4M + 2S$, where $M$ and $S$ respectively denote that of a field multiplication and squaring. If $Z_3 = 1$, the computational cost of the addition formulas is $3M + 2S$. If $(A+2)/4$ is precomputed, the computational cost of the doubling formulas is also $3M + 2S$.

Let $(k_t \cdots k_1 k_0)_2$ be the binary representation of $k$ with $k_t = 1$. To compute the scalar multiplication $kP$ from $P = (x, y)$, we hold $\{m_iP, (m_i+1)P\}$ for $m_i = (k_t \cdots k_i)_2$. If $k_i = 0$, $m_iP = 2m_{i+1}P$ and $(m_i + 1)P = (m_{i+1} + 1)P + m_{i+1}P$.

Otherwise, $m_iP = (m_{i+1}+1)P + m_{i+1}P$ and $(m_i+1)P = 2(m_{i+1}+1)P$. We can compute $\{kP, (k+1)P\}$ from $\{P, 2P\}$. Montgomery scalar multiplication requires the addition formulas $t-1$ times and the doubling formulas $t$ times. Since the difference between $(m_{i+1}+1)P$ and $m_{i+1}P$ is $P$, we can assume $(X_3, Z_3) = (x, 1)$ in addition formulas (2). The computational cost of Montgomery scalar multiplication $kP$ is $(6|k|-3)M + (4|k|-2)S$, where $|k|$ is the bit length of $k$.

In ECDSA signature verification, we need to compute $x$-coordinate of $kP + lQ$, where $P, Q$ are points on the elliptic curve and $k, l$ are integers less than the order of the base point. $kP$ and $lQ$ can be computed using Montgomery scalar multiplication, but $kP + lQ$ cannot be computed from $kP$ and $lQ$ using formulas (2) because the difference between $kP$ and $lQ$ is unknown. Therefore, the recovery of $Y$-coordinate of $kP$ and $lQ$ is required to compute $kP + lQ$ from $kP$ and $lQ$ using other addition formulas. The method of recovering $Y$-coordinate is described in [18]. If $kP = (X_0, Z_0)$, $(k+1)P = (X_1, Z_1)$ and $P = (x, y)$, we can recover $Y$-coordinate of $kP = (X, Y, Z)$ as:

$$
\begin{aligned}
X &= 2ByZ_0Z_1X_0 \\
Y &= Z_1((X_0 + xZ_0 + 2AZ_0)(X_0x + Z_0) - 2A{Z_0}^2) - (X_0 - xZ_0)^2X_1 \\
Z &= 2ByZ_0Z_1Z_0
\end{aligned}
$$

The computational cost of recovering $Y$-coordinate is $12M + S$.

To compute $x$-coordinate of $kP + lQ$, we require these 6 steps.

**Step1** Compute $kP$ using Montgomery scalar multiplication
**Step2** Recover $Y$-coordinate of $kP$
**Step3** Compute $lQ$ using Montgomery scalar multiplication
**Step4** Recover $Y$-coordinate of $lQ$
**Step5** Compute $kP + lQ$ from $kP$ and $lQ$ in projective coordinates
**Step6** Compute $x$-coordinate of $kP + lQ$ using $x = X/Z$

The computational cost of Step5 is $10M + 2S$ and that of Step6 is $M + I$, where $I$ denotes that of a field inversion. We can assume $|k| = |l|$ without loss of generality. The computational cost of $x$-coordinate of $kP + lQ$ is $(12|k|+29)M + 8|k|S + I$.

### 2.2 Simultaneous Scalar Multiplication on Elliptic Curve with Weierstrass Form

For $a, b \in \mathbb{F}_p$, the elliptic curve with Weierstrass form $E_W$ is represented by

$$E_W : y^2 = x^3 + ax + b \ (4a^2 + 27b^3 \neq 0).$$

We remark that all elliptic curves with Montgomery form can be transformed into Weierstrass form, but not all elliptic curves with Weierstrass form can be transformed into Montgomery form.

$kP + lQ$ can be computed simultaneously on the elliptic curve with Weierstrass form without precomputed points [19]. This method is known as Shamir's

trick [5]. On the elliptic curve with Weierstrass form over $\mathbb{F}_p$, the most effective method computing $kP + lQ$ without precomputed points is the simultaneous scalar multiplication using non-adjacent form (NAF) $(k'_{t'} \cdots k'_1 k'_0)$, where $k'_i \in \{0, \pm 1\}$ $(0 \leq i \leq t')$, and mixed coordinates [4]. In [3], Weierstrass simultaneous scalar multiplication using window method and mixed coordinates is described. This method is faster than Weierstrass simultaneous scalar multiplication using NAF, but requires much more memories where the points are stored. That is why we pick Weierstrass simultaneous scalar multiplication using NAF and mixed coordinates in this section.

NAF has the property that no two consecutive coefficients $k'_i$ are non-zero and the average density of non-zero coefficients is approximately 1/3. In mixed coordinates, we use the addition formulas of $J^m \leftarrow J + A$ for the additions, the doubling formulas of $J \leftarrow 2J^m$ for the doublings ahead of addition, and the doubling formulas of $J^m \leftarrow 2J^m$ for the doublings ahead of doubling, where $J$, $A$, and $J^m$ respectively denote Jacobian coordinate, affine coordinate, and modified Jacobian coordinate. This method is described as follows.

---

Algorithm 1: Weierstrass Simultaneous Scalar Multiplication using NAF and mixed coordinates

---

Input: $k = (k_t \cdots k_1 k_0)_2,\ l = (l_t \cdots l_1 l_0)_2,\ P, Q \in E_W\ (k_t$ **or** $l_t = 1)$.
Output: $x$-coordinate of $W = kP + lQ$.

1. Compute $P + Q,\ P - Q$.
2. Let $(k'_{t'} \cdots k'_1 k'_0)$ and $(l'_{t'} \cdots l'_1 l'_0)$ be NAF of $k$ and $l$ $\ (k'_{t'}$ or $l'_{t'} = 1)$.
3. $W \leftarrow k'_{t'} P + l'_{t'} Q$.
4. For $i$ from $t' - 1$ downto 0 do
   - 4.1 if $(k'_i, l'_i) = (0, 0)$ then
     $W \leftarrow 2W \quad (J^m \leftarrow 2J^m)$;
   - 4.2 else then
     $W \leftarrow 2W \quad (J \leftarrow 2J^m)$,
     $W \leftarrow W + (k'_i P + l'_i Q) \quad (J^m \leftarrow J + A)$.
5. Compute $x$-coordinate of $W$.

---

At step 1, $P + Q,\ P - Q$ are computed in affine coordinates and their computational cost is $4M + 2S + I$. In mixed coordinates, the computational cost of the addition formulas of $J^m \leftarrow J + A$, the doubling formulas of $J \leftarrow 2J^m$, and the doubling formulas of $J^m \leftarrow 2J^m$ are respectively $9M + 5S$, $3M + 4S$, and $4M + 4S$. Since the probability that $(k'_i, l'_i) = (0, 0)$ is $(1 - 1/3)^2 = 4/9$, we repeat step 4.1 $4|k|/9$ times and step 4.2 $5|k|/9$ times if $t' = t + 1$. The computational cost of step 4.1 is $(4|k|/9) \cdot (4M + 4S)$ and that of step 4.2 is $(5|k|/9) \cdot (12M + 9S)$. This shows that the computational cost of step 4 is $76|k|/9 \cdot M + 61|k|/9 \cdot S$. The computational cost of step 5 is $M + S + I$ by $x = X/Z^2$. Therefore, the computational cost of $x$-coordinate of $kP + lQ$ is $(76|k| + 45)/9 \cdot M + (61|k| + 27)/9 \cdot S + 2I$.

## 3 Proposed Method — Montgomery Simultaneous Scalar Multiplication

Now we propose the new method to compute $kP + lQ$ simultaneously on the elliptic curve with Montgomery form over $\mathbb{F}_p$.

At first, we define a set of four points $G_i$,

$$G_i = \left\{ \begin{array}{c} m_iP + n_iQ, \\ m_iP + (n_i+1)Q, \\ (m_i+1)P + n_iQ, \\ (m_i+1)P + (n_i+1)Q \end{array} \right\}, \tag{3}$$

for $m_i = (k_t \cdots k_i)_2,\ n_i = (l_t \cdots l_i)_2$. Now, we present how to compute $G_i$ from $G_{i+1}$ in every case of $(k_i, l_i)$.

1. $(k_i, l_i) = (0, 0)$
   Since $m_i = 2m_{i+1}$ and $n_i = 2n_{i+1}$, we can compute all elements of $G_i$ from $G_{i+1}$ as:

$$\begin{aligned} m_iP + n_iQ &= 2(m_{i+1}P + n_{i+1}Q) \\ m_iP + (n_i+1)Q &= (m_{i+1}P + (n_{i+1}+1)Q) + (m_{i+1}P + n_{i+1}Q) \\ (m_i+1)P + n_iQ &= ((m_{i+1}+1)P + n_{i+1}Q) + (m_{i+1}P + n_{i+1}Q) \\ (m_i+1)P + (n_i+1)Q &= ((m_{i+1}+1)P + n_{i+1}Q) + (m_{i+1}P + (n_{i+1}+1)Q) \end{aligned}$$

   All elements of $G_i$ can be computed without $(m_{i+1}+1)P + (n_{i+1}+1)Q \in G_{i+1}$.

2. $(k_i, l_i) = (0, 1)$
   Since $m_i = 2m_{i+1}$ and $n_i = 2n_{i+1} + 1$, we can compute all elements of $G_i$ from $G_{i+1}$ as:

$$\begin{aligned} m_iP + n_iQ &= (m_{i+1}P + (n_{i+1}+1)Q)) + (m_{i+1}P + n_{i+1}Q) \\ m_iP + (n_i+1)Q &= 2(m_{i+1}P + (n_{i+1}+1)Q) \\ (m_i+1)P + n_iQ &= ((m_{i+1}+1)P + (n_{i+1}+1)Q) + (m_{i+1}P + n_{i+1}Q) \\ (m_i+1)P + (n_i+1)Q &= ((m_{i+1}+1)P + (n_{i+1}+1)Q) \\ &\quad + (m_{i+1}P + (n_{i+1}+1)Q) \end{aligned}$$

   All elements of $G_i$ can be computed without $(m_{i+1}+1)P + n_{i+1}Q \in G_{i+1}$.

3. $(k_i, l_i) = (1, 0)$
   Since $m_i = 2m_{i+1} + 1$ and $n_i = 2n_{i+1}$, we can compute all elements of $G_i$ from $G_{i+1}$ as:

$$\begin{aligned} m_iP + n_iQ &= ((m_{i+1}+1)P + n_{i+1}Q) + (m_{i+1}P + n_{i+1}Q) \\ m_iP + (n_i+1)Q &= ((m_{i+1}+1)P + (n_{i+1}+1)Q) + (m_{i+1}P + n_{i+1}Q) \\ (m_i+1)P + n_iQ &= 2((m_{i+1}+1)P + n_{i+1}Q) \\ (m_i+1)P + (n_i+1)Q &= ((m_{i+1}+1)P + (n_{i+1}+1)Q) \\ &\quad + ((m_{i+1}+1)P + n_{i+1}Q) \end{aligned}$$

All elements of $G_i$ can be computed without $m_{i+1}P + (n_{i+1}+1)Q \in G_{i+1}$.

4. $(k_i, l_i) = (1, 1)$
Since $m_i = 2m_{i+1} + 1$ and $n_i = 2n_{i+1} + 1$, we can compute all elements of $G_i$ from $G_{i+1}$ as:

$$\begin{aligned} m_iP + n_iQ &= ((m_{i+1}+1)P + n_{i+1}Q)) \\ &\quad + (m_{i+1}P + (n_{i+1}+1)Q) \\ m_iP + (n_i+1)Q &= ((m_{i+1}+1)P + (n_{i+1}+1)Q) \\ &\quad + (m_{i+1}P + (n_{i+1}+1)Q) \\ (m_i+1)P + n_iQ &= ((m_{i+1}+1)P + (n_{i+1}+1)Q) \\ &\quad + ((m_{i+1}+1)P + n_{i+1}Q) \\ (m_i+1)P + (n_i+1)Q &= 2((m_{i+1}+1)P + (n_{i+1}+1)Q) \end{aligned}$$

All elements of $G_i$ can be computed without $m_{i+1}P + n_{i+1}Q \in G_{i+1}$.

In every case, all elements of $G_i$ can be computed from $G_{i+1}$ without $(m_{i+1}+1-k_i)P + (n_{i+1}+1-l_i)Q \in G_{i+1}$. When we define a set of three points $G'_i$,

$$G'_i = G_i - \{(m_i + 1 - k_{i-1})P + (n_i + 1 - l_{i-1})Q\}, \tag{4}$$

all elements of $G_i$ can be computed from $G'_{i+1}$. Therefore, we can compute $G'_i$ from $G'_{i+1}$. The way to compute $G'_i$ fro m $G'_{i+1}$ depends on $(k_i, l_i, k_{i-1}, l_{i-1})$, since computing $G_i$ from $G'_{i+1}$ depends on $(k_i, l_i)$ while extracting $G'_i$ from $G_i$ depends on $(k_{i-1}, l_{i-1})$.

**Example 1** $(k_i, l_i, k_{i-1}, l_{i-1}) = (0, 0, 0, 0)$

$m_i$, $n_i$, $G'_{i+1}$ and $G'_i$ would be described as: $m_i = 2m_{i+1},\ n_i = 2n_{i+1}$,

$$G'_{i+1} = \left\{ \begin{array}{c} m_{i+1}P + n_{i+1}Q, \\ m_{i+1}P + (n_{i+1}+1)Q, \\ (m_{i+1}+1)P + n_{i+1}Q \end{array} \right\}, G'_i = \left\{ \begin{array}{c} m_iP + n_iQ, \\ m_iP + (n_i+1)Q, \\ (m_i+1)P + n_iQ \end{array} \right\}$$

Therefore, we can compute $G'_i$ from $G'_{i+1}$ as:

$$\begin{aligned} m_iP + n_iQ &= 2(m_{i+1}P + n_{i+1}Q) \\ m_iP + (n_i+1)Q &= (m_{i+1}P + (n_{i+1}+1)Q) + (m_{i+1}P + n_{i+1}Q) \\ (m_i+1)P + n_iQ &= ((m_{i+1}+1)P + n_{i+1}Q) + (m_{i+1}P + n_{i+1}Q) \end{aligned} \tag{5}$$

If we define $G'_i = \{T_0[i], T_1[i], T_2[i]\}$, equations (5) can be described as:

$$\begin{aligned} T_0[i] &= 2T_0[i+1] \\ T_1[i] &= T_1[i+1] + T_0[i+1]\ (T_1[i+1] - T_0[i+1] = Q) \\ T_2[i] &= T_2[i+1] + T_0[i+1]\ (T_2[i+1] - T_0[i+1] = P). \end{aligned}$$

**Example 2** $(k_i, l_i, k_{i-1}, l_{i-1}) = (0, 1, 1, 0)$

$m_i$, $n_i$, $G'_{i+1}$ and $G'_i$ would be described as: $m_i = 2m_{i+1},\ n_i = 2n_{i+1} + 1$,

$$G'_{i+1} = \left\{ \begin{array}{c} m_{i+1}P + n_{i+1}Q, \\ m_{i+1}P + (n_{i+1}+1)Q, \\ (m_{i+1}+1)P + (n_{i+1}+1)Q \end{array} \right\}, G'_i = \left\{ \begin{array}{c} m_iP + n_iQ, \\ (m_i+1)P + n_iQ, \\ (m_i+1)P + (n_i+1)Q \end{array} \right\}$$

Therefore, we can compute $G'_i$ from $G'_{i+1}$ as:

$$\begin{aligned} m_iP + n_iQ &= (m_{i+1}P + (n_{i+1}+1)Q) + (m_{i+1}P + n_{i+1}Q) \\ (m_i+1)P + n_iQ &= ((m_{i+1}+1)P + (n_{i+1}+1)Q) \\ &\quad + (m_{i+1}P + n_{i+1}Q) \qquad (6) \\ (m_i+1)P + (n_i+1)Q &= ((m_{i+1}+1)P + (n_{i+1}+1)Q) \\ &\quad + (m_{i+1}P + (n_{i+1}+1)Q) \end{aligned}$$

Equations (6) can be described as:

$$\begin{aligned} T_0[i] &= T_1[i+1] + T_0[i+1]\ (T_1[i+1] - T_0[i+1] = Q) \\ T_1[i] &= T_2[i+1] + T_0[i+1]\ (T_2[i+1] - T_0[i+1] = P + Q) \\ T_2[i] &= T_2[i+1] + T_1[i+1]\ (T_2[i+1] - T_1[i+1] = P) \end{aligned}$$

From equations (3) and (4), we can define $G'_{t+1}$ as an initial set of $G'_i$ as:

$$\begin{aligned} G_{t+1} &= \{\mathcal{O}, Q, P, P+Q\} \\ G'_{t+1} &= G_{t+1} - \{(1-k_t)P + (1-l_t)Q\}, \end{aligned}$$

where $\mathcal{O}$ is the point at infinity. By calculating $G'_i$ from $G'_{i+1}$ repeatedly, we can compute $G'_1$ from $G'_{t+1}$ whereas $kP + lQ$ will be computed from $G'_1$. Our method to compute $x$-coordinate of $kP + lQ$ can be described as next page. $T_i + T_j\ (P)$ means that the difference between $T_i$ and $T_j$ is $P$.

At step1, $P - Q$ must be computed because the difference between $(m_i + 1)P + n_iQ$ and $m_iP + (n_i + 1)Q$ is $P - Q$.

We consider about the computational cost of the proposed method. At step1, we compute $P + Q, P - Q$ in affine coordinates and their computation cost is $4M + 2S + I$. At step2, 3 and 4, we use projective coordinates in the same way as Section 2.1. We assume $|k| = |l|$ as referred in the previous section. At step3, we require the addition formulas twice and the doubling formulas once, or the addition formulas three times per bit of $k$. In either case, since the computational cost per bit of $k$ is $9M + 6S$, the computational cost of step3 is $9(|k|-1)M + 6(|k|-1)S$. The computational cost of step4 is $3M + 2S$ and that of step5 is $M + I$. Therefore, the computational cost of the proposed method is $(9|k|-1)M + (6|k|-2)S + 2I$.

Algorithm 2: Montgomery Simultaneous Scalar Multiplication

Input: $k = (k_t \cdots k_1 k_0)_2,\ l = (l_t \cdots l_1 l_0)_2,\ P, Q \in E_M$ ($k_t$ **or** $l_t = 1$).
Output: $x$-coordinate of $W = kP + lQ$.

1. Compute $P + Q,\ P - Q$.
2. If $(k_t, l_t) = (0, 1)$ then: $T_0 \leftarrow \mathcal{O},\ T_1 \leftarrow Q,\ T_2 \leftarrow P + Q$;
   else if $(k_t, l_t) = (1, 0)$ then: $T_0 \leftarrow \mathcal{O},\ T_1 \leftarrow P,\ T_2 \leftarrow P + Q$;
   else then: $T_0 \leftarrow Q,\ T_1 \leftarrow P,\ T_2 \leftarrow P + Q$.
3. For $i$ from $t$ downto 1 do
   3.1. If $(k_i, l_i, k_{i-1}, l_{i-1}) = (0, 0, 0, 0)$ then:
   $T_2 \leftarrow T_2 + T_0\ (P),\ T_1 \leftarrow T_1 + T_0\ (Q),\ T_0 \leftarrow 2T_0$;
   3.2. else if $(k_i, l_i, k_{i-1}, l_{i-1}) = (0, 0, 0, 1)$ then:
   $T_2 \leftarrow T_2 + T_1\ (P - Q),\ T_1 \leftarrow T_1 + T_0\ (Q),\ T_0 \leftarrow 2T_0$;
   3.3. else if $(k_i, l_i, k_{i-1}, l_{i-1}) = (0, 0, 1, 0)$ then:
   $T \leftarrow T_1,\ T_1 \leftarrow T_2 + T_0\ (P),\ T_0 \leftarrow 2T_0,\ T_2 \leftarrow T_2 + T\ (P - Q)$;
   3.4. else if $(k_i, l_i, k_{i-1}, l_{i-1}) = (0, 0, 1, 1)$ then:
   $T \leftarrow T_1,\ T_1 \leftarrow T_2 + T_0\ (P),\ T_0 \leftarrow T + T_0\ (Q),\ T_2 \leftarrow T_2 + T\ (P - Q)$;
   3.5. else if $(k_i, l_i, k_{i-1}, l_{i-1}) = (0, 1, 0, 0)$ then:
   $T_2 \leftarrow T_2 + T_0\ (P + Q),\ T_0 \leftarrow T_1 + T_0\ (Q),\ T_1 \leftarrow 2T_1$;
   3.6. else if $(k_i, l_i, k_{i-1}, l_{i-1}) = (0, 1, 0, 1)$ then:
   $T_2 \leftarrow T_2 + T_1\ (P),\ T_0 \leftarrow T_1 + T_0\ (Q),\ T_1 \leftarrow 2T_1$;
   3.7. else if $(k_i, l_i, k_{i-1}, l_{i-1}) = (0, 1, 1, 0)$ then:
   $T \leftarrow T_1,\ T_1 \leftarrow T_2 + T_0\ (P + Q),\ T_0 \leftarrow T + T_0\ (Q),\ T_2 \leftarrow T_2 + T\ (P)$;
   3.8. else if $(k_i, l_i, k_{i-1}, l_{i-1}) = (0, 1, 1, 1)$ then:
   $T \leftarrow T_1,\ T_1 \leftarrow T_2 + T_0\ (P + Q),\ T_0 \leftarrow 2T,\ T_2 \leftarrow T_2 + T\ (P)$;
   3.9. else if $(k_i, l_i, k_{i-1}, l_{i-1}) = (1, 0, 0, 0)$ then:
   $T \leftarrow T_1,\ T_1 \leftarrow T_2 + T_0\ (P + Q),\ T_0 \leftarrow T + T_0\ (P),\ T_2 \leftarrow 2T$;
   3.10. else if $(k_i, l_i, k_{i-1}, l_{i-1}) = (1, 0, 0, 1)$ then:
   $T \leftarrow T_1,\ T_1 \leftarrow T_2 + T_0\ (P + Q),\ T_0 \leftarrow T + T_0\ (P),\ T_2 \leftarrow T_2 + T\ (Q)$;
   3.11. else if $(k_i, l_i, k_{i-1}, l_{i-1}) = (1, 0, 1, 0)$ then:
   $T_0 \leftarrow T_1 + T_0\ (P),\ T_2 \leftarrow T_2 + T_1\ (Q),\ T_1 \leftarrow 2T_1$;
   3.12. else if $(k_i, l_i, k_{i-1}, l_{i-1}) = (1, 0, 1, 1)$ then:
   $T_0 \leftarrow T_2 + T_0\ (P + Q),\ T_2 \leftarrow T_2 + T_1\ (Q),\ T_1 \leftarrow 2T_1$;
   3.13. else if $(k_i, l_i, k_{i-1}, l_{i-1}) = (1, 1, 0, 0)$ then:
   $T \leftarrow T_1,\ T_1 \leftarrow T_2 + T_0\ (P),\ T_0 \leftarrow T + T_0\ (P - Q),\ T_2 \leftarrow T_2 + T\ (Q)$;
   3.14. else if $(k_i, l_i, k_{i-1}, l_{i-1}) = (1, 1, 0, 1)$ then:
   $T \leftarrow T_1,\ T_1 \leftarrow T_2 + T_0\ (P),\ T_0 \leftarrow T + T_0\ (P - Q),\ T_2 \leftarrow 2T_2$;
   3.15. else if $(k_i, l_i, k_{i-1}, l_{i-1}) = (1, 1, 1, 0)$ then:
   $T_0 \leftarrow T_1 + T_0\ (P - Q),\ T_1 \leftarrow T_2 + T_1\ (Q),\ T_2 \leftarrow 2T_2$;
   3.16. else then:
   $T_0 \leftarrow T_2 + T_0\ (P),\ T_1 \leftarrow T_2 + T_1\ (Q),\ T_2 \leftarrow 2T_2$.
4. If $(k_0, l_0) = (0, 0)$ then $W \leftarrow 2T_0$;
   else if $(k_0, l_0) = (0, 1)$ then $W \leftarrow T_1 + T_0\ (Q)$;
   else if $(k_0, l_0) = (1, 0)$ then $W \leftarrow T_1 + T_0\ (P)$;
   else then $W \leftarrow T_1 + T_0\ (P - Q)$.
5. Compute $x$-coordinate of $W$ by $x = X/Z$.

## 4 Comparison

Now we compare the computational cost of the proposed method to that of both methods in Section 2. In addition, we compare this to the computational cost of the method described in IEEE P1363 Draft [8], which is based on the scalar multiplication using NAF on the elliptic curve with Weierstrass form. This is a fair comparison because all four methods require no precomputed point. Table 1 shows the computational cost of each method to compute $x$-coordinate of $kP + lQ$. $M$, $S$ and $I$ respectively denote the computational costs of a field multiplication, squaring and inversion.

**Table 1.** The computational costs of every method

| Method | $M$ | $S$ | $I$ |
|---|---|---|---|
| Weierstrass NAF [8] | $(40\|k\| - 1)/3$ | $14\|k\| - 9$ | 1 |
| Montgomery | $12\|k\| + 29$ | $8\|k\|$ | 1 |
| Weierstrass Simultaneous +NAF (Algorithm 1) | $(76\|k\| + 45)/9$ | $(61\|k\| + 27)/9$ | 2 |
| Montgomery Simultaneous (Algorithm 2) | $9\|k\| - 1$ | $6\|k\| - 2$ | 2 |

Table 2 shows the computational cost of each method for $|k| = 160$ and the total cost when we assume $S/M = 0.8$ and $I/M = 30$ [12]. We also compare the computational cost of our method to that of Weierstrass simultaneous scalar multiplication using window method and mixed coordinates, which requires memories for 13 points [3]. The proposed method, Montgomery simultaneous scalar multiplication, is about 45% faster than the method described in IEEE P1363 Draft, and about 25% faster than the method using Montgomery scalar multiplication and the recovery of $Y$-coordinate. Moreover, the proposed method is about 1% faster than Weierstrass simultaneous scalar multiplication using NAF. Our method is about 2% slower than Weierstrass simultaneous scalar multiplication using window method, but requires much less memories.

## 5 Running Times

Here we present the running times of each method described in Section 4. To calculate arbitrary precision arithmetic over $\mathbb{F}_p$, we used the GNU MP library GMP [6]. The running times were obtained on a Pentium II 300 MHz machine. We used the following elliptic curve over $\mathbb{F}_p$, where $|p| = 162$ and the order of the base point $r$ was 160 bits. $\#E$ means the order of this elliptic curve.

**Table 2.** The computational cost of each method for $|k| = 160$ and $S/M = 0.8, I/M = 30$.

| Method | $M$ | $S$ | $I$ | $M$ $(S/M = 0.8, I/M = 30)$ |
|---|---|---|---|---|
| Weierstrass NAF | 2133 | 2231 | 1 | 3938 |
| Montgomery | 1949 | 1280 | 1 | 3003 |
| Weierstrass Simultaneous + NAF | 1356 | 1087 | 2 | 2286 |
| Montgomery Simultaneous | 1439 | 958 | 2 | 2265 |
| Weierstrass Simultaneous + Window | 1281 | 1018 | 4 | 2215 |

$$
\begin{aligned}
p &= \texttt{2 0aa6fc4d 8396f3ac 06200db7 3e819694 067a0e7b} \\
a &= \texttt{1 5fed3282 429907d6 03b41b7a 309abf87 bed9bd83} \\
b &= \texttt{1 74019686 9a423134 f3cdf013 b13564d0 ba3999e8} \\
\#E &= 4 * \texttt{82a9bf13 60e5bceb 01878167 1d478cea 881e1d1d} \\
A &= \texttt{1 8be6a098 c28d6bc0 3286dc51 e7e3f705 8a5b9d98} \\
B &= \texttt{0 120c2550 f6ff7a01 440d78d1 122fa3ac aa70fd53}
\end{aligned}
$$

We obtained average running times to compute $x$-coordinate of $kP + lQ$ by randomly choosing 100 points $P, Q$ over this elliptic curve and 100 integers $k, l < r$. Table 3 shows the average time of each method to compute $x$-coordinate of $kP + lQ$. From Table 3, we notice the proposed method, Montgomery simultaneous scalar multiplication, is about 44% faster than the method described in IEEE P1363 Draft, and about 25% faster than the method using Montgomery scalar multiplication and the recovery of $Y$-coordinate. Moreover, the proposed method is about 3% faster than Weierstrass simultaneous scalar multiplication using NAF. This shows that the theoretical advantage of our method is actually observed.

**Table 3.** The average time of each method

| Method | Average time (ms) |
|---|---|
| Weierstrass NAF | 36.2 |
| Montgomery | 26.9 |
| Weierstrass Simultaneous + NAF | 20.8 |
| Montgomery Simultaneous | 20.1 |

## 6 Elliptic Curve over $\mathbb{F}_{2^n}$

A non-supersingular elliptic curve $E$ over $\mathbb{F}_{2^n}$ is represented by $E : x^2 + xy = x^3 + ax^2 + b$, where $a, b \in \mathbb{F}_{2^n}, b \neq 0$. We can apply the proposed method to Montgomery method on elliptic curves over $\mathbb{F}_{2^n}$ [14]. The advantage of Montgomery method on elliptic curves over $\mathbb{F}_{2^n}$ is that we need not consider transformability from Weierstrass form to Montgomery form. Since the computational cost of a field squaring over $\mathbb{F}_{2^n}$ is much lower than that of a field multiplication over $\mathbb{F}_{2^n}$, we can ignore it.

In Montgomery method, the computational cost of addition formulas and doubling formulas are respectively $4M$ and $2M$ in projective coordinates. If we compute $kP + lQ$ using Montgomery scalar multiplication, we requires addition formulas twice and doubling formulas twice per bit of $k$. Therefore, the computational cost per bit of $k$ is estimated to be about $12M$.

On the other hand, if we compute $kP + lQ$ using Montgomery simultaneous scalar multiplication, we require addition formulas twice and doubling formulas once at probability of $3/4$, and addition formulas three times at probability of $1/4$ per bit of $k$, as described in Algorithm 2. Since we require addition formulas $9/4$ times and doubling formulas $3/4$ times per bit of $k$, the computational cost per bit of $k$ is estimated to be about $21/2 \cdot M$.

In Weierstrass simultaneous scalar multiplication over $\mathbb{F}_{2^n}$ using NAF [7,13], the computational cost per bit of $k$ is estimated to be about $9M$ if $a = 0, 1$.

Therefore, the proposed method is only 13% faster than the method using Montgomery scalar multiplication and 17% slower than Weierstrass simultaneous scalar multiplication using NAF. This shows that the proposed method on elliptic curves over $\mathbb{F}_{2^n}$ is not so efficient as that on elliptic curves with Montgomery form over $\mathbb{F}_p$.

## 7 Conclusion

We proposed the new method to compute $x$-coordinate of $kP + lQ$ simultaneously on the elliptic curve with Montgomery form over $\mathbb{F}_p$ without precomputed points. To compute $x$-coordinate of $kP + lQ$ is required in ECDSA signature verification. Our method is about 25% faster than the method using Montgomery scalar multiplication and the recovery of $Y$-coordinate over $\mathbb{F}_p$, and slightly faster than Weierstrass simultaneous scalar multiplication over $\mathbb{F}_p$ using NAF and mixed coordinates. Our method is considered to be particularly useful in case that ECDSA signature generation is performed using Montgomery scalar multiplication on the elliptic curve over $\mathbb{F}_p$ because of its efficiency of computation and its immunity to timing attacks, since all arithmetic on the elliptic curve can be computed with Montgomery form and we don't require transformation to the elliptic curve with Weierstrass form. Furthermore, we showed that our method was applicable to Montgomery method on elliptic curves over $\mathbb{F}_{2^n}$.

**Acknowledgements**

We would like to thank Taizo Shirai for his careful reading and valuable suggestions. We also thank the referees for their numerous helpful comments.

## References

1. ANSI X9.62-1998, *Public Key Cryptography for the Financial Services Industry: The Elliptic Curve Digital Signature Algorithm (ECDSA)*, Working Draft, September 1998.
2. D. V. Bailey and C. Paar, "Optimal Extension Field for Fast Arithmetic in Public Key Algorithms", *Advances in Cryptography - CRYPTO '98*, LNCS 1462, pp. 472-485, 1998.
3. M. Brown, D. Hankerson, J. Lopez, and A. Menezes, "Software Implementation of the NIST Elliptic Curves Over Prime Fields", *Topics in Cryptology – CT-RSA 2001*, LNCS 2020, pp. 250-265, 2001.
4. H. Cohen, A. Miyaji, and T. Ono, "Efficient Elliptic Curve Exponentiation Using Mixed Coordinates", *Advances in Cryptography - ASIACRYPT '98*, LNCS 1514, pp. 51-65, 1998.
5. T. ElGamal, "A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms", *IEEE Transaction on Information Theory*, Vol. 31, pp. 469-472, 1985.
6. GNU MP Library GMP, version 3.1.1, October 2000. http://www.swox.com/gmp/
7. D. Hankerson, J. Lopéz, and A. Menezes, "Software Implementation of Elliptic Curve Cryptography over Binary Fields", *Cryptographic Hardware and Embedded Systems - CHES 2000*, LNCS 1965, pp. 3-24, 2000.
8. IEEE P1363, *Standard Specifications for Public Key Cryptography*, Draft Version 13, November 1999. http://grouper.ieee.org/groups/1363/
9. T. Kobayashi, H. Morita, K. Kobayashi, and F. Hoshino, "Fast Elliptic Curve Algorithm Combining Frobenius Map and Table Reference to Adapt to Higher Characteristic", *Advances in Cryptography - EUROCRYPT '99*, LNCS 1592, pp. 176-189, 1999.
10. N. Koblitz, "Elliptic Curve Cryptosystems", *Mathematics of Computation*, vol. 48, pp. 203-209, 1987.
11. P. C. Kocher, "Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems", *Advances in Cryptology – CRYPTO '96*, LNCS 1109, pp. 104-113, 1996.
12. C. H. Lim and H. S. Hwang, "Fast Implementation of Elliptic Arithmetic in $GF(p^n)$", *Public Key Cryptography - PKC 2000*, LNCS 1751, pp. 405-421, 2000.
13. J. Lopéz and R. Dahab, "Improved Algorithms for Elliptic Curve Arithmetic in $GF(2^n)$", *Selected Areas in Cryptology - SAC '98*, LNCS 1556, pp. 201-212, 1999.
14. J. Lopéz and R. Dahab, "Fast Multiplication on Elliptic Curves over $GF(2^m)$ without Precomputation", *Cryptographic Hardware and Embedded Systems - CHES '99*, LNCS 1717, pp. 316-327, 1999.
15. V. Miller, "Uses of Elliptic Curves in Cryptography", *Advances in Cryptography - CRYPTO '85*, LNCS 218, pp. 417-426, 1986.
16. P. L. Montgomery, "Speeding the Pollard and Elliptic Curve Methods of Factorization", *Mathematics of Computation*, vol. 48, pp. 243-264, 1987.

17. K. Okeya, H. Kurumatani, and K. Sakurai, "Elliptic Curves with the Montgomery-Form and Their Cryptographic Applications", *Public Key Cryptography - PKC 2000*, LNCS 1751, pp. 238-257, 2000.
18. K. Okeya and K. Sakurai, "Efficient Elliptic Curve Cryptosystems from a Scalar Multiplication Algorithm with Recovery of the $y$-coordinate on a Montgomery-Form Elliptic Curve", *Preproceedings of Cryptographic Hardware and Embedded Systems - CHES 2001*, pp. 129-144, 2001.
19. E. D. Win, S. Mister, B. Preneel, and M. Wiener, "On the Performance of Signature Schemes Based on Elliptic Curves", *Algorithm Number Theory - ANTS III*, LNCS 1423, pp. 252-266, 1998.