Efficient Elliptic Curve Cryptosystems from a Scalar Multiplication Algorithm with Recovery of the y-Coordinate on a Montgomery-Form Elliptic Curve

Katsuyuki Okeya[1] and Kouichi Sakurai[2]

[1] Hitachi, Ltd., Software Division,
5030, Totsuka-cho, Totsuka-ku, Yokohama, 244-8555, Japan
okeya_k@itg.hitachi.co.jp
[2] Kyushu University,
Department of Computer Science and Communication Engineering
6-10-1, Hakozaki, Higashi-ku, Fukuoka, 812-8581, Japan
sakurai@csce.kyushu-u.ac.jp

Abstract. We present a scalar multiplication algorithm with recovery of the y-coordinate on a Montgomery form elliptic curve over any non-binary field.
The previous algorithms for scalar multiplication on a Montgomery form do not consider how to recover the y-coordinate. So although they can be applicable to certain restricted schemes (e.g. ECDH and ECDSA-S), some schemes (e.g. ECDSA-V and MQV) require scalar multiplication with recovery of the y-coordinate.
We compare our proposed scalar multiplication algorithm with the traditional scalar multiplication algorithms (including Window-methods in Weierstrass form), and discuss the Montgomery form versus the Weierstrass form in the performance of implementations with several techniques of elliptic curve cryptosystems (including ECES, ECDSA, and ECMQV). Our results clarify the advantage of the cryptographic usage of Montgomery-form elliptic curves in constrained environments such as mobile devices and smart cards.

Keywords: *Elliptic Curve Cryptosystem, Montgomery form, Fast Scalar Multiplication, y-coordinate recovery*

1 Introduction

Lim and Hwang give the following problem [LH00, page 409]: "Montgomery's method is not a general algorithm for elliptic scalar multiplication in GF(p^n), since it can't compute the y-coordinate of kP." This paper completely solves this problem, and shows that Montgomery's method is indeed a general algorithm.

Ç.K. Koç, D. Naccache, and C. Paar (Eds.): CHES 2001, LNCS 2162, pp. 126–141, 2001.

1.1 Montgomery-Form Elliptic Curves

Montgomery introduced the non-standard form $E^M : BY^2 = X^3 + AX^2 + X$ for elliptic curves in [Mon87], while the most standard form of elliptic curves is $E : y^2 = x^3 + ax + b$, which is called the (short) Weierstrass form. While investigating efficient scalar multiplication algorithms on elliptic curves, some researchers [LD99,LH00,Kur98,OKS00] have independently observed that Montgomery's method [Mon87] has an advantage in preventing timing attacks [Koc, Koc96].[1]

1.2 Elliptic Curve Cryptosystems without the y-Coordinate

The scalar multiplication algorithm on a Montgomery-form elliptic curve is fast, because it requires information on the x-coordinate only. Recall that previous proposed algorithms on a Montgomery form consider only the x-coordinate of kP, the k-time scalar multiplication of the point P over the elliptic curve. This is enough for application to some elliptic curve cryptosystems including the key-establishment scheme ECDH, and signature generation ECDSA-S [IEEEp1363].

However, we should note that the ECDSA verifying algorithm cannot be executed without referencing the y-coordinate of kP. Also ECSVDP-MQV, ECSVDP-MQVC, ECVP-NR, and ECVP-DSA, which are described in the draft standard IEEE P1363 [IEEEp1363], need a scalar multiplication with recovery of the y-coordinate.

1.3 Montgomery Form with Recovery of the y-Coordinate

Recently, López and Dahab [LD99] extended the idea of Montgomery's method [Mon87] to binary fields (i.e. $\mathbf{F}_{2^m}$), and developed an algorithm for a scalar multiplication with recovery of the y-coordinate.

However, their algorithm is valid only over binary fields, and designing an efficient algorithm for recovering a y-coordinate in the Montgomery form with non-binary fields has remained open.

1.4 Our Contributions

We present a scalar multiplication algorithm with recovery of the y-coordinate on a Montgomery-form elliptic curve over non-binary fields.

We further compare our proposed scalar multiplication algorithm with the traditional scalar multiplication algorithms (including Window-methods on the Weierstrass form), and discuss the cryptographic advantage of using the Montgomery-form elliptic curve.

Our analysis shows that the scalar multiplication on a Montgomery-form elliptic curve, which requires no precomputation, is faster than that of the window-method on a Weierstrass-form elliptic curve, if the size of the definition field is smaller than 391 bits in a reasonable implementation with $(S/M) = 0.8$ and

[1] It was shown that a Montgomery-form elliptic curve is effective for preventing differential power analysis [OS00].

$(I/M) = 30$ [LH00]. Recall that elliptic curve cryptosystems over prime fields whose sizes are larger than 272 bits are believed to be secure until the year 2050 even if some cryptanalytic developments occur [LV00].

We further consider the Montgomery form versus the Weierstrass form in the performance of implementations with several techniques of elliptic curve cryptosystems (including ECES, ECDSA, and ECMQV). We compare the amounts of computation and storage of our Montgomery method with recovery of the y-coordinate to those of the Weierstrass window methods (with simultaneous techniques) for point multiplication kP (resp. for $kP+Q$ and for $kP+lQ$). Our results suggest new advantages of the Montgomery form over the Weierstrass form in the implementation of elliptic curve cryptosystems

2 Elliptic Curve Schemes Using the y-Coordinate

The elliptic curve signature scheme ECDSA cannot be executed without referencing the y-coordinate of the point kP, the k-time scalar multiplication of the point P. This is in contrast to some elliptic curve schemes (e.g. ECDH), which can be executed without the y-coordinate. The verifying algorithm of ECDSA requires the operation $kP + k'Q$. To put it simply, computation requires the addition of a scalar-multiplied point and another point. Addition of points, without using their differences, on a Montgomery-form elliptic curve requires the y-coordinates of the points. That is the reason why ECDSA requires the y-coordinate of kP.

The same applies to ECSVDP-MQV, ECSVDP-MQVC, ECVP-NR, and ECVP-DSA, which are described in the draft standard IEEE P1363 [IEEEp1363]. These schemes also require the operation $kP+Q$, which is an addition of a scalar-multiplied point and another point.

We would like to emphasize that recovering the y-coordinate completely solves these problems. Therefore, Montgomery's method of scalar multiplication becomes a general algorithm.

3 Recovering the y-Coordinate

Research has studied recovery of the y-coordinate on a Montgomery-like scalar multiplication method in the case of an elliptic curve defined over a finite field with characteristic 2 [LD99]. However, no similar algorithm is known in the case of a Montgomery-form elliptic curve [Mon87] defined over a prime field (nor over an OEF [BP98]). First, we will take up the case of characteristic 2, and then we will focus our attention on the case of a Montgomery-form elliptic curve.

3.1 Recovering the y-Coordinate ($p = 2$)[LD99]

Let $\mathbf{F}_{2^m}$ be a finite field with characteristic 2. A non-supersingular elliptic curve over $\mathbf{F}_{2^m}$ is defined as follows.

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

where $a, b \in \mathbf{F}_{2^m}$ and $b \neq 0$.

Theorem 1 ([LD99]). *Let $P = (x, y), P_1 = (x_1, y_1), P_2 = (x_2, y_2)$ be points on the elliptic curve. Assume that $P_2 = P_1 + P$ and $x \neq 0$. Then*

$$y_1 = (x_1 + x)\{(x_1 + x)(x_2 + x) + x^2 + y\}/x + y.$$

3.2 Recovering the y-Coordinate ($p \geq 3$)

Let p be a prime and $\mathbf{F}_{p^m}$ be a finite field with characteristic p. A Montgomery-form elliptic curve over $\mathbf{F}_{p^m}$ is defined as follow.

$$BY^2 = X^3 + AX^2 + X,$$

where $A, B \in \mathbf{F}_{p^m}$ and $B(A^2 - 4) \neq 0$.

First, we construct a method for recovering the y-coordinate on a Montgomery-form elliptic curve in a similar fashion to the method on the elliptic curve over $\mathbf{F}_{2^m}$. The proof of the propositions in this section is in the Appendix.

Theorem 2. *Let $P = (x, y), P_1 = (x_1, y_1), P_2 = (x_2, y_2)$ be points on a Montgomery-form elliptic curve. Assume that $P_2 = P_1 + P$ and $y \neq 0$. Then*

$$y_1 = \frac{(x_1 x + 1)(x_1 + x + 2A) - 2A - (x_1 - x)^2 x_2}{2By}.$$

Corollary 1. *Let P, P_1 and P_2 be as in Theorem 2. We express $P_1 = (\frac{X_1}{Z_1}, \frac{Y_1}{Z_1})$, $P_2 = (\frac{X_2}{Z_2}, \frac{Y_2}{Z_2})$, and define $X_1^{rec}, Y_1^{rec}, Z_1^{rec}$ which are given by the following:*

$$\begin{aligned} X_1^{rec} &= 2ByZ_1Z_2X_1 \\ Y_1^{rec} &= Z_2\left[(X_1 + xZ_1 + 2AZ_1)(X_1x + Z_1) - 2AZ_1^2\right] - (X_1 - xZ_1)^2 X_2 \\ Z_1^{rec} &= 2ByZ_1Z_2Z_1 \end{aligned}$$

Then the relation $(X_1^{rec}, Y_1^{rec}, Z_1^{rec}) = (X_1, Y_1, Z_1)$ holds in projective coordinates.

This method for recovering the Y-coordinate needs to compute twelve multiplications and one squaring. An example of the algorithm using Corollary 1 is the next Algorithm 1.

Algorithm 1 : Algorithm for recovering the Y-coordinate

INPUT x, y, X_1, Z_1, X_2, Z_2

OUTPUT $X_1^{rec}, Y_1^{rec}, Z_1^{rec}$

1. $T_1 \leftarrow x \times Z_1$
2. $T_2 \leftarrow X_1 + T_1$
3. $T_3 \leftarrow X_1 - T_1$
4. $T_3 \leftarrow T_3 \times T_3$
5. $T_3 \leftarrow T_3 \times X_2$
6. $T_1 \leftarrow 2A \times Z_1$
7. $T_2 \leftarrow T_2 + T_1$
8. $T_4 \leftarrow x \times X_1$
9. $T_4 \leftarrow T_4 + Z_1$
10. $T_2 \leftarrow T_2 \times T_4$
11. $T_1 \leftarrow T_1 \times Z_1$
12. $T_2 \leftarrow T_2 - T_1$
13. $T_2 \leftarrow T_2 \times Z_2$
14. $Y_1^{rec} \leftarrow T_2 - T_3$
15. $T_1 \leftarrow 2B \times y$
16. $T_1 \leftarrow T_1 \times Z_1$
17. $T_1 \leftarrow T_1 \times Z_2$
18. $X_1^{rec} \leftarrow T_1 \times X_1$
19. $Z_1^{rec} \leftarrow T_1 \times Z_1$

Next, we propose another method for recovering the y-coordinate on a Montgomery-form elliptic curve.

Theorem 3. *Let $P=(x,y), P_1=(x_1,y_1), P_2=(x_2,y_2), P_3=(x_3,y_3)$ be points on a Montgomery-form elliptic curve. Assume that $P_2=P_1+P, P_3=P_1-P$ and $y\neq 0$. Then*

$$y_1=\frac{(x_3-x_2)(x_1-x)^2}{4By}.$$

Corollary 2. *Let P, P_1, P_2 and P_3 be as in Theorem 3. We express $P_1=(\frac{X_1}{Z_1},\frac{Y_1}{Z_1})$, $P_2=(\frac{X_2}{Z_2},\frac{Y_2}{Z_2}), P_3=(\frac{X_3}{Z_3},\frac{Y_3}{Z_3})$, and define $X_1^{rec}, Y_1^{rec}, Z_1^{rec}$ which are given by the following:*

$$\begin{aligned} X_1^{rec} &= 4ByZ_1Z_2Z_3X_1 \\ Y_1^{rec} &= (X_3Z_2-Z_3X_2)\,(X_1-Z_1x)^2 \\ Z_1^{rec} &= 4ByZ_1Z_2Z_3Z_1 \end{aligned}$$

Then $(X_1^{rec}, Y_1^{rec}, Z_1^{rec})=(X_1,Y_1,Z_1)$ in projective coordinates.

This method for recovering the Y-coordinate in projective coordinates needs to compute ten multiplications and one squaring. Thus, this method is faster than that using Corollary 1. However, we have to compute the point P_3 before we use this method if P_3 is not given.

Proposition 1. *Let $P=(x,y)$ be a point on a Montgomery-form elliptic curve, and $kP=(x_k,y_k)$ for $k=1,2,\cdots$ be the scalar-multiplied point of the point P. For any $\frac{X_m}{Z_m}=x_m, \frac{X_n}{Z_n}=x_n$ and $\frac{X_{m+n}}{Z_{m+n}}=x_{m+n}$ $(m>n)$, we define X' and Z' which are given by the following:*

$$X'=Z_{m+n}[(X_m-Z_m)(X_n+Z_n)+(X_m+Z_m)(X_n-Z_n)]^2$$

$$Z'=X_{m+n}[(X_m-Z_m)(X_n+Z_n)-(X_m+Z_m)(X_n-Z_n)]^2$$

Then $\frac{X'}{Z'}=x_{m-n}$ is satisfied.

The calculation of X' and Z' needs to compute four multiplications and two squarings. Thus, recovering the Y-coordinate in projective coordinates without P_3 requires fourteen multiplications and three squarings.

We should notice that a combination of these formulae reduces the computation amount. Let us examine the computation amount of the later method in more detail. We have $x_d=\frac{X_d}{Z_d}$, $x_{d+1}=\frac{X_{d+1}}{Z_{d+1}}$,$x_{d-1}=\frac{X_{d-1}}{Z_{d-1}}$, and we transform the formula of y_1 on Theorem 3 to the formula in projective coordinates with the substitution of these values. Then we obtain the following equation:

$$y_d=\frac{(X_{d-1}Z_{d+1}-Z_{d-1}X_{d+1})(X_d-Z_dx)}{4ByZ_{d-1}Z_{d+1}Z_d^2}$$

Proposition 1 enables us to eliminate X_{d-1} and Z_{d-1} from the equation above. We obtain the following equation with the settings $X_1 = x$ and $Z_1 = 1$:

$$y_d = \frac{\{Z_{d+1}U + X_{d+1}V\}\{Z_{d+1}U - X_{d+1}V\}U^2}{ByZ_{d+1}X_{d+1}V^2Z_d^2},$$

where $U = X_d x - Z_d$,$V = X_d - xZ_d$. On the other hand, we have $x_d = \frac{X_d}{Z_d}$. We obtain

$$x_d = \frac{ByZ_{d+1}X_{d+1}Z_dV^2X_d}{ByZ_{d+1}X_{d+1}Z_dV^2Z_d}$$

with the reduction with the denominator of y_d for reducing the number of inversion.

M, S and I respectively denote $\mathbf{F}_{p^m}$-operations of multiplication, squaring, and inversion. The method for recovering the y-coordinate in affine coordinates needs the computation amount $15M + 2S + I$.

On the other hand, we set

$$\begin{aligned} X_d^{rec} &= ByZ_{d+1}X_{d+1}Z_dV^2X_d \\ Y_d^{rec} &= \{Z_{d+1}U + X_{d+1}V\}\{Z_{d+1}U - X_{d+1}V\}U^2 \\ Z_d^{rec} &= ByZ_{d+1}X_{d+1}Z_dV^2Z_d \end{aligned}$$

Then the relation $(X_d^{rec}, Y_d^{rec}, Z_d^{rec}) = (X_d, Y_d, Z_d)$ holds in projective coordinates. The method for recovering the y-coordinate needs the computation amount $13M + 2S$. Therefore, the computation amount shrinks by one multiplication and one squaring. An example of the algorithm using this method is the following:

Algorithm 2 : Algorithm for recovering the Y-coordinate

INPUT $x, y, X_d, Z_d, X_{d+1}, Z_{d+1}$

OUTPUT $X_d^{rec}, Y_d^{rec}, Z_d^{rec}$

1. $T_1 \leftarrow X_d \times x$
2. $T_1 \leftarrow T_1 - Z_d$
3. $T_2 \leftarrow Z_d \times x$
4. $T_2 \leftarrow X_d - T_2$
5. $T_3 \leftarrow Z_{d+1} \times T_1$
6. $T_4 \leftarrow X_{d+1} \times T_2$
7. $T_1 \leftarrow T_1 \times T_1$
8. $T_2 \leftarrow T_2 \times T_2$
9. $T_2 \leftarrow T_2 \times Z_d$
10. $T_2 \leftarrow T_2 \times X_{d+1}$
11. $T_2 \leftarrow T_2 \times Z_{d+1}$
12. $T_2 \leftarrow T_2 \times y$
13. $T_2 \leftarrow T_2 \times B$
14. $X_d^{rec} \leftarrow T_2 \times X_d$
15. $Z_d^{rec} \leftarrow T_2 \times Z_d$
16. $T_2 \leftarrow T_3 + T_4$
17. $T_3 \leftarrow T_3 - T_4$
18. $T_1 \leftarrow T_1 \times T_2$
19. $Y_d^{rec} \leftarrow T_1 \times T_3$

In summary, Algorithm 1 needs the computation amount $12M + S$, and Algorithm 2 needs the computation amount $13M + 2S$. Therefore, Algorithm 1 is faster than Algorithm 2 by one multiplication and one squaring.

4 Montgomery Form versus Weierstrass Form

It was pointed out in the previous section that we are able to efficiently recover the y-coordinate of a scalar-multiplied point on a Montgomery-form elliptic curve. If we connect the traditional scalar multiplication algorithm with the algorithm for recovering the y-coordinate, we construct a fast scalar multiplication algorithm on a Montgomery-form elliptic curve which gives the whole coordinates of a scalar-multiplied point.

$T^{Mon}(l), T_x^{Mon}(l)$ and $T_y^{Mon}(l)$ denote the computation amount for the proposed scalar multiplication algorithm with l, the size of a scalar value,[2] that for the traditional scalar multiplication on a Montgomery-form elliptic curve, and that for recovery of the y-coordinate, respectively. Then we have the following:

$$\begin{aligned} T_x^{Mon}(l) &= (6l-3)M + (4l-2)S \\ T_y^{Mon}(l) &= 12M + S \\ T^{Mon}(l) &= T_x^{Mon}(l) + T_y^{Mon}(l) \\ &= (6l+9)M + (4l-1)S \end{aligned}$$

We should recall that, on a scalar multiplication algorithm over a Weierstrass-form elliptic curve, the algorithm using the window method in the mixed modified Jacobian coordinates is one of the fastest [CMO98]. $T^{Wei}(w,l)$ denotes the computation amount with window width w and size l. According to [LH00] (and also see [CMO98]), $T^{Wei}(w,l)$ is estimated as follows.

$$\begin{aligned} T^{Wei}(w,l) = wI &+ \left(\frac{8l}{w+2} + 4l + 5\cdot 2^{w-1} - 2w - 14\right) M \\ &+ \left(\frac{3l}{w+2} + 4l + 2^{w-1} - 2w - 2\right) S \end{aligned}$$

First, we compare the computation amount for the Montgomery-form elliptic curve to that for the Weierstrass-form elliptic curve with a window width of 4 in terms of size l. Assume that $T^{Mon}(l) > T^{Wei}(4,l)$, then we obtain

$$(I/M) < \left(\frac{1}{6}l - \frac{5}{2}\right) + \left(-\frac{1}{8}l + \frac{1}{4}\right)(S/M).$$

For simplicity, we denote s and t for (S/M) and (I/M), respectively. Then the inequality above is expressed by

$$l > \frac{24t - 6s + 54}{4 - 3s}.$$

For example, we set $s = 0.8$, and compute l for $t = 10, 20, 30, 40$ and 50. We obtain $l > 181, 331, 481, 631$ and 781, respectively. Thus, in the case that the size is smaller than the bits above, the scalar multiplication on a Montgomery-form elliptic curve is faster than that on a Weierstrass-form elliptic curve.

[2] Actually, the size of a scalar value is approximately the size of the definition field.

Second, we compare the computation amount of a Montgomery-form elliptic curve to that of a Weierstrass-form elliptic curve with a window width of 5 in terms of size l. Assume that $T^{Mon}(l) > T^{Wei}(5, l)$, we obtain

$$l > \frac{35t + 35s + 329}{6 - 3s}.$$

Finally, we compare the computation amount for the Weierstrass-form elliptic curve with window width 4 to that with window width 5 in terms of size l. Assume that $T^{Wei}(4, l) > T^{Wei}(5, l)$, we obtain

$$l > \frac{42t + 252s + 1596}{8 + 3s}.$$

To sum up the comparison between the computation amount in terms of size l we have seen thus far, we obtain Table 1.

According to [LV00], elliptic curve cryptosystems over *prime* fields whose sizes are larger than 272 bits are believed to be secure until the year 2050, even if some cryptanalytic developments occur. Under a reasonable assumption that $(S/M) = 0.8$ and $(I/M) = 30$ for *prime* fields [LH00], we see from Table 1 that the scalar multiplication with $w = 5$ is faster than that with $w = 4$ if the size is larger than 295 bits. Thus, the scalar multiplication on a Montgomery-form elliptic curve is faster than that on a Weierstrass-form elliptic curve if the size is smaller than 391 bits. From this viewpoint, one may say that the scalar multiplication algorithm on a Montgomery-form elliptic curve is faster than that on a Weierstrass-form elliptic curve for cryptographic use.

5 Comparison of Our Proposed Method to Other Methods

In this section, we use computation amount to compare our proposed method, that is the scalar multiplication algorithm with recovery of the y-coordinate on a Montgomery-form elliptic curve, with other methods for several schemes (ECES, ECElGamal, ECDSA-V and ECSVDP-MQV) [ANSI,IEEEp1363,SEC-1]. On the comparison with methods to compute a scalar multiplication and ECDSA scheme for elliptic curves over $\mathbf{F}_{2^m}$, see [HHM00].

5.1 Elliptic Curve Encryption Scheme

An elliptic curve encryption scheme (ECES)[3] needs to compute the operation kP. We select Montgomery's method with/without recovering the y-coordinate and the window method with the width $w = 4, 5$ from the methods to compute kP, and compare the computation amounts for the size 160 bits.

To compute kP by Montgomery's method with (resp. without) recovery of the y-coordinate, the computation needs the computation amount $1480.2M$ (resp. $1467.4M$) with assuming $(S/M) = 0.8$.

[3] For further details about ECES, see [IEEEp1363] for example.

Table 1. Border key sizes between scalar multiplication methods

(S/M)	(I/M)	$w = 5/w = 4$	$w = 4/Mon$	$w = 5/Mon$
1.0	50	359	1249	705
	40	321	1009	589
	30	283	769	472
	20	245	529	355
	10	207	289	239
0.9	50	367	961	640
	40	328	776	534
	30	289	592	428
	20	249	407	322
	10	210	223	216
0.8	50	375	781	586
	40	335	631	489
	30	295	481	391
	20	254	331	294
	10	214	181	197
0.7	50	384	658	540
	40	342	532	450
	30	301	406	360
	20	259	279	271
	10	218	153	181
0.6	50	393	569	501
	40	350	460	417
	30	307	351	334
	20	265	242	251
	10	222	133	167
0.5	50	403	501	466
	40	359	405	389
	30	314	309	311
	20	270	213	233
	10	226	117	155

"$w = 4/Mon$" indicates the border key sizes between the scalar multiplication on a Weierstrass-form elliptic curve using the window method with a window width of 4 and that on a Montgomery-form elliptic curve. That is, the former is faster than the latter if the size is larger than the border size. "$w = 5/w = 4$" and "$w = 5/Mon$" are similar in meaning to "$w = 4/Mon$".

To compute kP by the window method with the width $w = 4$ (resp. $w = 5$), the computation amount $1446.4M + 4I$ (resp. $1449.4M + 5I$) is needed.

5.2 Elliptic Curve ElGamal Encryption Scheme

An elliptic curve ElGamal encryption scheme[4] needs to compute the operation $kP + Q$. We select Montgomery's method with recovery of the y-coordinate and

[4] For further details about the elliptic curve ElGamal encryption scheme, see [IEEEp1363] for example.

Table 2. Computation amounts for point multiplication kP

Method	Points stored	# of M	# of I
Montgomery (traditional)	0	1467.4	0
Montgomery (with recovery of y)	0	1480.2	0
Window Method ($w = 4$)	7	1446.4	4
Window Method ($w = 5$)	15	1449.4	5

the window method with the width $w = 4, 5$ from the methods to compute $kP + Q$, and compare the computation amounts for the size 160 bits.

To compute $kP+Q$ by Montgomery's method with recovery of the y-coordinate, the computation needs to compute the following operations:

(a) The scalar multiplication kP by Montgomery's method.
(b) The addition $kP + Q$.

(a) Since the size of k is 160 bits, the computation needs the computation amount $969M + 639S$ which includes the computation amount for recovering the y-coordinate.
(b) The computation needs the computation amount $11M + 2S$ for the addition in the projective coordinates on a Montgomery-form elliptic curve.

Thus, to compute $kP + Q$ by Montgomery's method with recovery of the y-coordinate needs the computation amount $1492.8M$, assuming $(S/M) = 0.8$.

To compute $kP + Q$ by the window method with the width $w = 4, 5$, the computation needs to compute the following operations:

(c) The scalar multiplication kP by window method.
(d) The addition $kP + Q$.

(c) The computation amount is $872M + 718S + 4I$ if $w = 4$, and it is $879M + 713S + 5I$ if $w = 5$.
(d) The computation amount is $8M + 3S$ by using the addition formulae of $J + A \to J$, where J is the Jacobian coordinates and A is the affine coordinates.

Thus, to compute $kP + Q$ by the window method, the computation needs the computation amount $1456.8M + 4I$ if $w = 4$, or $1459.8M + 5I$ if $w = 5$.

Table 3. Computation amount for point multiplication $kP + Q$

Method	Points stored	# of M	# of I	Remark
Montgomery (traditional)	-	-	-	impossible
Montgomery (with recovery of y)	0	1492.8	0	
Window Method ($w = 4$)	7	1456.8	4	
Window Method ($w = 5$)	15	1459.8	5	

5.3 ECDSA Scheme (Verification)

The signature verification of an elliptic curve digital signature algorithm (ECDSA)[5] needs to compute the operation $kP + lQ$ where P is a fixed point and Q is not known a priori. We select the fixed-base comb method [LL94] with the width $w = 4, 5$ + Montgomery's method, and a simultaneous method [ElG85,HHM00] with the width $w = 2$, and compare the computation amount for the size 160 bits.

To compute the fixed-base comb ($w = 4, 5$) + Montgomery method, the computation needs to compute the following operations:

(e) The scalar multiplication kP by the fixed-base comb method.
(f) The scalar multiplication lQ by Montgomery's method.
(g) The addition $kP + lQ$.

(e) In the case of the width $w = 4$, the computation needs the computation amount $445M + 338S$ by using the addition formulae of $J + A \to J^m$ for the additions, the doubling formulae of $J^m \to J$ for the doublings ahead of addition, and the doubling formulae of $J^m \to J^m$ for the doublings ahead of doubling, where J^m is the modified Jacobian coordinates. In the case of the width $w = 5$, the computation amount is $362M + 273S$.
(f) Since the point Q is on a Weierstrass-form elliptic curve, we transform this into a point on a Montgomery-form elliptic curve. The computation needs the computation amount $2M$. Then, we compute the scalar-multiplied point lQ by Montgomery's method with recovery of the y-coordinate. The computation amount is $969M + 639S$. Then, we transform the point into a point on the Weierstrass-form elliptic curve. The computation amount is $2M$.
(g) For the sake of fast computation of $kP + lQ$, we transform the point lQ in projective coordinates into a point in Chudnovsky Jacobian coordinates. The computation needs the computation amount $3M + S$. The computation amount $11M + 3S$ is needed for the addition $kP + lQ$.

Thus, to compute $kP + lQ$ by fixed-base comb ($w = 4$) + Montgomery method, the computation needs the computation amount $2216.8M$, assuming $(S/M) = 0.8$. The number of points stored is 14. In the case of the fixed-base comb ($w = 5$) + Montgomery method, the computation needs the computation amount $2081.8M$. The number of points stored is 30.

In the case of the simultaneous method with the width $w = 2$, the computation of points stored needs the computation amount $49M + 12S + 2I$ by using the Montgomery trick [Coh93]. The computation of multi scalar multiplication needs the computation amount $1223M + 1001S$ by using the addition formulae of $J + A \to J^m$ for the additions, the doubling formulae of $J^m \to J$ for the doublings ahead of addition and the doubling formulae of $J^m \to J^m$ for the doublings ahead of doubling. Thus, the computation amount $2084.4M + 3I$ is needed. The number of points stored is 13.

[5] For further details about ECDSA, see [ANSI] for example.

Memory Constrained As regards the situation in which memory is constrained, we examine methods in which the number of points stored is within 10.

In the case of fixed-base comb ($w = 3$) + Montgomery, the computation amount is $2420.0M$, and the number of points stored is 6. In the case of the width $w = 2$, the computation amount is $2762.0M$, and the number of points stored is 2. In the case of Montgomery + Montgomery, the computation amount is $2980.0M$, and no extra points are needed. In the case of simultaneous ($w = 1$), the computation amount is $2585.6M + I$, and the number of points stored is 1.

Table 4. Computation amount for point multiplication $kP + lQ$, P fixed

Method	Points stored	# of M	# of I
Fixed-base comb($w = 4$) + Montgomery	14	2216.8	0
Fixed-base comb($w = 5$) + Montgomery	30	2081.8	0
Simultaneous($w = 2$)	13	2082.4	2

Table 5. Computation amount for point multiplication $kP+lQ$, P fixed, when memory is constrained

Method	Points stored	# of M	# of I
Fixed-base comb($w = 2$) + Montgomery	2	2762.0	0
Fixed-base comb($w = 3$) + Montgomery	6	2420.0	0
Montgomery + Montgomery	0	2980.0	0
Simultaneous($w = 1$)	1	2585.6	1

5.4 ECSVDP-MQV

The elliptic curve secret value derivation primitive, Menezes-Qu-Vanstone version (ECSVDP-MQV)[6] needs to compute the operation $k(P + lQ)$, where l is half the size of k, and points P, Q are not known a priori.

To compute $k(P + lQ)$ by Montgomery's method (Montgomery $*$ Montgomery), the computation needs to compute the following operations:

(h) The scalar multiplication lQ by Montgomery's method.
(i) The addition $P + lQ$.
(j) The scalar multiplication $k(P + lQ)$ by Montgomery's method.

(h) Since l is half the size of k, the computation needs the computation amount $489M + 319S$, which includes the computation amount for recovering the y-coordinate.
(i) The computation needs the computation amount $14M + 2S$ for the addition on the Montgomery-form elliptic curve.

[6] For further details about ECSVDP-MQV, see [IEEEp1363] for example.

(j) The computation needs the computation amount $957M + 638S$.

Thus, to compute $k(P + lQ)$ by Montgomery's method, the computation needs the computation amount $2227.2M$.

In the case of the simultaneous method with the width $w = 2$, to compute the points stored needs $63M + 15S + 3I$, and to compute $kP + klQ$ by simultaneous method needs $1223M + 1001S$. Thus, the computation amount $2098.8 + 3I$ is needed.

Table 6. Computation amount for point multiplication $k(P + lQ)$, where l is half the size of k

Method	Points stored	# of M	# of I
Montgomery * Montgomery	0	2227.2	0
Simultaneous($w = 2$)	13	2098.8	3

References

[AMV93] Agnew,G.B., Mullin,R.C., Vanstone,S.A., *An Implementation of Elliptic Curve Cryptosystems Over $F_{2^{155}}$*, IEEE Journal on Selected Areas in Communications, vol.11,No.5,(1993),804-813.

[ANSI] ANSI X9.62, Public Key Cryptography for the Financial Services Industry, *The Elliptic Curve Digital Signature Algorithm(ECDSA)*,(1999).

[BP98] Bailey,D.V., Paar,C.,*Optimal Extension Fields for Fast Arithmetic in Public-Key Algorithms*, Advances in Cryptology - CRYPTO'98, LNCS1462, (1998), 472-485.

[BSS99] Blake,I.F.,Seroussi,G.,Smart,N.P., *Elliptic Curves in Cryptography*, Cambridge University Press,(1999).

[CMO98] Cohen,H., Miyaji,A., Ono,T., *Efficient Elliptic Curve Exponentiation Using Mixed Coordinates*, Advances in Cryptology - ASIACRYPT '98, LNCS1514, (1998), 51-65.

[Coh93] Cohen, H., *A course in computational algebraic number theory*, GTM138, Springer-Verlag, New York, (1993).

[ElG85] ElGamal, T., *A public key cryptosystem and a signature scheme based on discrete logarithms*, IEEE Transactions on Information Theory, 31 (1985), 469-472.

[Enge99] Enge, A., *Elliptic Curves and their applications to Cryptography*, Kluwer Academic publishers,(1999).

[HHM00] Hankerson, D., Hernandez, J.L., Menezes, A., *Software Implementation of Elliptic Curve Cryptography Over Binary Fields*, Pre-Proceedings of Workshop on Cryptographic Hardware and Embedded Systems (CHES2000), (2000), 1-24.

[IEEEp1363] IEEE P1363 Standard Specifications for Public-Key Cryptography (1999), Available at http://grouper.ieee.org/groups/1363/

[Kur98] Kurumatani, H. *A Japanese patent announcement P2000-187438A* (In Japanese) Submitted in 22nd of Dec. (1998), available from http://www.jpomiti.go.jp/home.htm

[Kob87] Koblitz,N., *Elliptic curve cryptosystems*, Math. Comp.48, (1987),203-209.

[Koc] Kocher,C., *Cryptanalysis of Diffie-Hellman,RSA,DSS, and Other Systems Using Timing Attacks*, Available at http://www.cryptography.com/

[Koc96] Kocher,C., *Timing Attacks on Implementations of Diffie-Hellman, RSA,DSS, and Other Systems*, Advances in Cryptology - CRYPTO '96, LNCS1109, (1996), 104-113.

[LD99] López,J., Dahab,R., *Fast Multiplication on Elliptic Curves over $GF(2^m)$ without Precomputation*, Cryptographic Hardware and Embedded Systems (CHES'99), LNCS1717, (1999), 316-327.

[LH00] Lim, C.H. and Hwang,H.S., *Fast implementation of Elliptic Curve Arithmetic in $GF(p^m)$*, Proc. PKC'00 LNCS1751, (2000), 405-421.

[LL94] Lim, C. and Lee, P., *More flexible exponentiation with precomputation*, Advances in Cryptology - CRYPTO '94, LNCS839, (1994), 95-107.

[LV00] Lenstra, A.K. and Verheul, E.R, *Selecting Cryptographic Key Sizes*, Proc. PKC'00 LNCS1751, (2000), 446-465.

[Mil86] Miller,V.S., *Use of elliptic curves in cryptography*, Advances in Cryptology - CRYPTO '85, LNCS218, (1986), 417-426.

[Mon87] Montgomery,P.L., *Speeding the Pollard and Elliptic Curve Methods of Factorizations*, Math. Comp. 48, (1987), 243-264

[OKS00] Okeya,K., Kurumatani,H., Sakurai,K., *Elliptic Curves with the Montgomery - Form and Their Cryptographic Applications*, Public Key Cryptography (PKC2000), LNCS1751, (2000), 238-257.

[OS00] Okeya,K., Sakurai,K., *Power Analysis Breaks Elliptic Curve Cryptosystems even Secure against the Timing Attack*, Progress in Cryptology - INDOCRYPT 2000, LNCS1977, (2000), 178-190.

[OSK99] Ohgishi,K., Sakai,R., Kasahara,M., *Elliptic Curve Signature Scheme with No y Coordinate*, Proc. SCIS'99,W4-1.3 (1999), 285-287.

[SEC-1] Standards for Efficient Cryptography, *Elliptic Curve Cryptography Ver.1.0*, (2000), Available at http://www.secg.org/secg_docs.htm

[Van97] Vanstone,S.A., *Accelerated finite field operations on an elliptic curve*, GB patent, Application number GB9713138.7 (Date Lodged, 20.06.1997).

A Algorithms for Recovering y-Coordinate (Affine Version)

Algorithm 3 : algorithm for recovering y-coordinate

INPUT x, y, X_1, Z_1, X_2, Z_2

OUTPUT x_1, y_1

1. $T_1 \leftarrow x \times Z_1$
2. $T_2 \leftarrow X_1 + T_1$
3. $T_3 \leftarrow X_1 - T_1$
4. $T_3 \leftarrow X_3 \times T_3$
5. $T_3 \leftarrow T_3 \times X_2$
6. $T_1 \leftarrow 2A \times Z_1$
7. $T_2 \leftarrow T_2 + T_1$
8. $T_4 \leftarrow x \times X_1$
9. $T_4 \leftarrow T_4 + Z_1$
10. $T_2 \leftarrow T_2 \times T_4$
11. $T_1 \leftarrow T_1 \times Z_1$
12. $T_2 \leftarrow T_2 - T_1$
13. $T_2 \leftarrow T_2 \times Z_2$
14. $T_2 \leftarrow T_2 - T_3$
15. $T_1 \leftarrow 2B \times y$
16. $T_1 \leftarrow T_1 \times Z_1$
17. $T_1 \leftarrow T_1 \times Z_2$
18. $T_3 \leftarrow T_1 \times Z_1$
19. $T_3 \leftarrow 1/T_3$
20. $y_1 \leftarrow T_2 \times T_3$
21. $T_1 \leftarrow T_1 \times X_1$
22. $x_1 \leftarrow T_1 \times T_3$

The computation amount of Algorithm 3 is $14M + S + I$.

Algorithm 4 : algorithm for recovering y-coordinate

INPUT $x, y, X_d, Z_d, X_{d+1}, Z_{d+1}$

OUTPUT x_d, y_d

1. $T_1 \leftarrow X_d \times x$
2. $T_1 \leftarrow T_1 - Z_d$
3. $T_2 \leftarrow Z_d \times x$
4. $T_2 \leftarrow X_d - T_2$
5. $T_3 \leftarrow X_{d+1} \times T_2$
6. $T_2 \leftarrow T_2 \times T_2$
7. $T_2 \leftarrow T_2 \times X_{d+1}$
8. $T_2 \leftarrow T_2 \times Z_{d+1}$
9. $T_2 \leftarrow T_2 \times y$
10. $T_2 \leftarrow T_2 \times B$
11. $T_2 \leftarrow T_2 \times Z_d$
12. $T_4 \leftarrow T_2 \times X_d$
13. $T_2 \leftarrow T_2 \times Z_d$
14. $T_2 \leftarrow 1/T_2$
15. $x_d \leftarrow T_2 \times T_4$
16. $T_4 \leftarrow T_1 \times Z_{d+1}$
17. $T_1 \leftarrow T_1 \times T_1$
18. $T_2 \leftarrow T_1 \times T_2$
19. $T_1 \leftarrow T_3 + T_4$
20. $T_3 \leftarrow T_3 - T_4$
21. $T_1 \leftarrow T_1 \times T_3$
22. $y_d \leftarrow T_1 \times T_2$

The computation amount of Algorithm 4 is $15M + 2S + I$.

B Proof of the Propositions

Proof (of Theorem 2). Since $P_2 = P_1 + P$, the x-coordinate x_2 is computed as follows.

$$x_2 = B\left(\frac{y_1 - y}{x_1 - x}\right)^2 - A - x_1 - x.$$

Since P_1 and P are on the Montgomery-form elliptic curve, it follows that $By_1^2 = x_1^3 + Ax_1^2 + x_1$ and $By^2 = x^3 + Ax^2 + x$. We find the following equation with an

easy calculation from the equation above.

$$x_2 = \frac{x_1 + x - 2By_1y + 2Ax_1x + x_1x^2 + xx_1^2}{(x_1 - x)^2}.$$

The result follows from this equation. □

Proof (of Theorem 3). Since $P_2 = P_1 + P$, $P_3 = P_1 - P$ and $-P = (x, -y)$, the x-coordinates x_2 and x_3 are computed as follows.

$$x_2 = B\left(\frac{y_1 - y}{x_1 - x}\right)^2 - A - x_1 - x \tag{1}$$

$$x_3 = B\left(\frac{y_1 + y}{x_1 - x}\right)^2 - A - x_1 - x \tag{2}$$

We obtain the following equations with multiplication by $(x_1 - x)^2$ at the equations (1) and (2).

$$B(y_1 - y)^2 = (x_2 + x_1 + x + A)(x_1 - x)^2 \tag{3}$$

$$B(y_1 + y)^2 = (x_3 + x_1 + x + A)(x_1 - x)^2 \tag{4}$$

We obtain the following equation with subtraction the equation (3) from the equation (4).

$$4By_1y = (x_3 - x_2)(x_1 - x)^2 \tag{5}$$

The result follows from this equation. □

Proof (of Proposition 1). For simplicity, we set

$$\alpha = [(X_m - Z_m)(X_n + Z_n) + (X_m + Z_m)(X_n - Z_n)],$$

$$\beta = [(X_m - Z_m)(X_n + Z_n) - (X_m + Z_m)(X_n - Z_n)].$$

Then we have

$$\begin{aligned} \frac{X'}{Z'} &= \frac{Z_{m+n}\alpha^2}{X_{m+n}\beta^2} \\ &= \frac{X_{m-n}\beta^2\alpha^2}{Z_{m-n}\alpha^2\beta^2} \qquad \text{(because of addition formulae)} \\ &= x_{m-n}. \end{aligned}$$

□