Published in J. von zur Gathen, J.L. Imaña, and Ç.K. Koç, Eds, *Arithmetic of Finite Fields (WAIFI 2008)*, vol. 5130 of *Lecture Notes in Computer Science*, pp. 36–46, Springer, 2008.

Fast Point Multiplication on Elliptic Curves Without Precomputation

Marc Joye

Thomson R&D France
Technology Group, Corporate Research, Security Laboratory
1 avenue de Belle Fontaine, 35576 Cesson-Sévigné Cedex, France
marc.joye@thomson.net

Abstract. Elliptic curves find numerous applications. This paper describes a simple strategy to speed up their arithmetic in right-to-left methods. In certain settings, this leads to a non-negligible performance increase compared to the left-to-right counterparts.

Keywords. Elliptic curve arithmetic, binary right-to-left exponentiation, mixed coordinate systems.

1 Introduction

Elliptic curve point multiplication — namely, the computation of $\boldsymbol{Q} = [k]\boldsymbol{P}$ given a point $\boldsymbol{P}$ on an elliptic curve and a scalar k — is central in almost every non-trivial application of elliptic curves (cryptography, coding theory, computational number theory, ...). Its efficiency depends on different factors: the field definition, the elliptic curve model, the internal point representation and, of course, the scalar multiplication method itself.

The choice of the field definition impacts the performance of the underlying field arithmetic: addition, multiplication and inversion. There are two types of fields: fields where inversion is relatively fast and fields where it is not. In the latter case, projective coordinates are preferred over affine coordinates to represent points on an elliptic curve. Points can also be represented with their x-coordinate only. Point multiplication is then evaluated via Lucas chains [13]. This avoids the evaluation of the y-coordinate, which may result in improved overall performance.

Yet another technique to speed up the computation is to use additional (dummy) coordinates to represent points [4]. This technique was later refined by considering mixed coordinate systems [6]. The strategy is to add two points where the first point is given in some coordinate system and the second point is given in some other coordinate system, to get the result point in some (possibly different) coordinate system.

Basically, there exist two main families of scalar multiplication methods, depending on the direction scalar k is scanned: left-to-right methods and right-to-left methods [10, 5]. Left-to-right methods are often used as they lead to many different generalizations, including windowing methods [8]. In this paper, we are

interested in implementations on constrained devices like smart cards. Hence, we restrict our attention to binary methods so as to avoid precomputing and storing (small) multiples of input point $\boldsymbol{P}$. We evaluate the performance of the classical binary algorithms (left-to-right and right-to-left) in different coordinate systems. Moreover, as the inverse of a point on an elliptic curve can in most cases be obtained for free, we mainly analyze their signed variants [15, 14]. Quite surprisingly, we find a number of settings where the right-to-left methods outperform the left-to-right methods. Our strategy is to make use of mixed coordinate systems but, unlike [6], we do this on binary methods for scalar multiplication. Such a strategy only reveals useful for the right-to-left methods because, as will become apparent later, the point addition routine and the point doubling routine may use different input/output coordinate systems. This gives rise to further gains not available for left-to-right methods.

The rest of this paper is organized as follows. In the next section, we introduce some background on elliptic curves and review their arithmetic. We also review the classical binary scalar multiplication methods. In Section 3, we present several known techniques to speed up the point multiplication. In Section 4, we describe fast implementations of right-to-left point multiplication. We analyze and compare their performance with prior methods. Finally, we conclude in Section 5.

2 Elliptic Curve Arithmetic

An *elliptic curve over a field* $\mathbb{K}$ is a plane non-singular cubic curve with a $\mathbb{K}$-rational point [16]. If $\mathbb{K}$ is a field of characteristic $\neq 2, 3$,[1] an elliptic curve over $\mathbb{K}$ can be expressed, up to birational equivalence, by the (affine) Weierstraß equation

$$E_{/\mathbb{K}} : y^2 = x^3 + a_4\, x + a_6 \quad \text{with } \Delta := -(4{a_4}^3 + 27{a_6}^2) \neq 0\,,$$

the rational point being the (unique) point at infinity $\boldsymbol{O}$. The condition $\Delta \neq 0$ implies that the curve is non-singular.

The set of $\mathbb{K}$-rational points on E is denoted by $E(\mathbb{K})$. It forms a commutative group where $\boldsymbol{O}$ is the neutral element, under the *'chord-and-tangent'* law. The inverse of $\boldsymbol{P} = (x_1, y_1)$ is $-\boldsymbol{P} = (x_1, -y_1)$. The addition of $\boldsymbol{P} = (x_1, y_1)$ and $\boldsymbol{Q} = (x_2, y_2)$ on E with $\boldsymbol{Q} \neq -\boldsymbol{P}$ is given by $\boldsymbol{R} = (x_3, y_3)$ where

$$x_3 = \lambda^2 - x_1 - x_2 \quad \text{and} \quad y_3 = \lambda(x_1 - x_3) - y_1 \tag{1}$$

with

$$\lambda = \begin{cases} \dfrac{y_1 - y_2}{x_1 - x_2} & \text{if } \boldsymbol{P} \neq \boldsymbol{Q} \quad \text{[chord]} \\ \dfrac{3{x_1}^2 + a_4}{2y_1} & \text{if } \boldsymbol{P} = \boldsymbol{Q} \quad \text{[tangent]} \end{cases}.$$

[1] We focus on these fields because inversion can be expensive compared to a multiplication. For elliptic curves over binary fields, a fast point multiplication method without precomputation is available [12].

2.1 Coordinate systems

To avoid (multiplicative) inversions in the addition law, points on elliptic curves are usually represented with projective coordinate systems.

In *homogeneous coordinates*, a point $\boldsymbol{P} = (x_1, y_1)$ is represented by the triplet $(X_1 : Y_1 : Z_1) = (\theta x_1 : \theta y_1 : \theta)$ for some non-zero $\theta \in \mathbb{K}$, on the elliptic curve $Y^2Z = X^3 + a_4\, XZ^2 + a_6\, Z^3$. The neutral element is given by the point at infinity $(0 : \theta : 0)$ with $\theta \neq 0$. Conversely, a projective homogeneous point $(X_1 : Y_1 : Z_1)$ with $Z_1 \neq 0$ corresponds to the affine point $(X_1/Z_1, Y_1/Z_1)$.

In *Jacobian coordinates*, a point $\boldsymbol{P} = (x_1, y_1)$ is represented by the triplet $(X_1 : Y_1 : Z_1) = (\lambda^2\, x_1 : \lambda^3\, y_1 : \lambda)$ for some non-zero $\lambda \in \mathbb{K}$. The elliptic curve equation becomes

$$Y^2 = X^3 + a_4\, XZ^4 + a_6\, Z^6 \ .$$

Putting $Z = 0$, we see that the neutral element is given by $\boldsymbol{O} = (\lambda^2 : \lambda^3 : 0)$. Given the projective Jacobian representation of a point $(X_1 : Y_1 : Z_1)$ with $Z_1 \neq 0$, its affine representation can be recovered as $(x_1, y_1) = (X_1/{Z_1}^2, Y_1/{Z_1}^3)$.

2.2 Point addition

We detail the arithmetic with Jacobian coordinates as they give rise to faster formulæ [9].

Replacing (x_i, y_i) with $(X_i/{Z_i}^2, Y_i/{Z_i}^3)$ in Eq. (1) we find after a little algebra that the addition of $\boldsymbol{P} = (X_1 : Y_1 : Z_1)$ and $\boldsymbol{Q} = (X_2 : Y_2 : Z_2)$ with $\boldsymbol{Q} \neq \pm\boldsymbol{P}$ (and $\boldsymbol{P}, \boldsymbol{Q} \neq \boldsymbol{O}$) is given by $\boldsymbol{R} = (X_3 : Y_3 : Z_3)$ where

$$X_3 = R^2 + G - 2V\,, \quad Y_3 = R(V - X_3) - S_1 G\,, \quad Z_3 = Z_1 Z_2 H \tag{2}$$

with $R = S_1 - S_2$, $G = H^3$, $V = U_1 H^2$, $S_1 = Y_1 {Z_2}^3$, $S_2 = Y_2 {Z_1}^3$, $H = U_1 - U_2$, $U_1 = X_1 {Z_2}^2$, and $U_2 = X_2 {Z_1}^2$ [6]. Let M and S respectively denote the cost of a (field) multiplication and of a (field) squaring. We see that the addition of two (different) points requires $12\mathsf{M} + 4\mathsf{S}$. When a fast squaring is available, this can also be evaluated with $11\mathsf{M} + 5\mathsf{S}$ by computing $2Z_1Z_2 = (Z_1 + Z_2)^2 - {Z_1}^2 - {Z_2}^2$ and "rescaling" X_3 and Y_3 accordingly [1].

The doubling of $\boldsymbol{P} = (X_1 : Y_1 : Z_1)$ (i.e., when $\boldsymbol{Q} = \boldsymbol{P}$) is given by $\boldsymbol{R} = (X_3 : Y_3 : Z_3)$ where

$$X_3 = M^2 - 2S\,, \quad Y_3 = M(S - X_3) - 8T\,, \quad Z_3 = 2Y_1 Z_1 \tag{3}$$

with $M = 3{X_1}^2 + a_4\, {Z_1}^4$, $T = {Y_1}^4$, and $S = 4X_1{Y_1}^2$. Letting c denote the cost of a multiplication by constant a_4, the doubling of a point costs $3\mathsf{M} + 6\mathsf{S} + 1\mathsf{c}$ or $1\mathsf{M} + 8\mathsf{S} + 1\mathsf{c}$ by evaluating $S = 2[(X_1 + {Y_1}^2)^2 - {X_1}^2 - T]$ and $Z_3 = (Y_1 + Z_1)^2 - {Y_1}^2 - {Z_1}^2$ [1].

Remark that Eq. (3) remains valid for doubling $\boldsymbol{O}$. We get $[2](\lambda^2 : \lambda^3 : 0) = (\lambda^8 : \lambda^{12} : 0) = \boldsymbol{O}$.

2.3 Point multiplication

Let $k = \sum_{i=0}^{\ell-1} k_i\, 2^i$ with $k_i \in \{0,1\}$ denote the binary expansion of k. The evaluation of $[k]\boldsymbol{P}$, that is, $\boldsymbol{P} + \boldsymbol{P} + \cdots + \boldsymbol{P}$ (k times) can be carried out as

$$[k]\boldsymbol{P} = \sum_{0\le i\le \ell-1} [k_i]\left([2^i]\boldsymbol{P}\right) = \sum_{\substack{0\le i\le \ell-1\\ k_i=1}} [2^i]\boldsymbol{P} = \sum_{\substack{0\le i\le \ell-1\\ k_i=1}} \boldsymbol{P_i} \quad \text{with } \begin{cases} \boldsymbol{P_0} = \boldsymbol{P} \\ \boldsymbol{P_i} = [2]\boldsymbol{P_{i-1}} \end{cases} .$$

By keeping track of the successive values of $\boldsymbol{P_i}$ in a variable $\boldsymbol{R_1}$ and by using a variable $\boldsymbol{R_0}$ to store the accumulated value, $\sum \boldsymbol{P_i}$, we so obtain the following right-to-left algorithm:

Algorithm 1 Right-to-left binary method

Input: $\boldsymbol{P}$, $k \ge 1$
Output: $[k]\boldsymbol{P}$
1: $\boldsymbol{R_0} \leftarrow \boldsymbol{O}$; $\boldsymbol{R_1} \leftarrow \boldsymbol{P}$
2: **while** ($k > 1$) **do**
3: **if** (k is odd) **then** $\boldsymbol{R_0} \leftarrow \boldsymbol{R_0} + \boldsymbol{R_1}$
4: $k \leftarrow \lfloor k/2 \rfloor$
5: $\boldsymbol{R_1} \leftarrow [2]\boldsymbol{R_1}$
6: **end while**
7: $\boldsymbol{R_0} \leftarrow \boldsymbol{R_0} + \boldsymbol{R_1}$
8: **return** $\boldsymbol{R_0}$

There is a similar left-to-right variant. It relies on the obvious observation that $[k]\boldsymbol{P} = [2]\big([k/2]\boldsymbol{P}\big)$ when k is even. Furthermore, since when k is odd, we can write $[k]\boldsymbol{P} = [k']\boldsymbol{P} + \boldsymbol{P}$ with $k' = k - 1$ even, we get:[2]

Algorithm 2 Left-to-right binary method

Input: $\boldsymbol{P}$, $k \ge 1$, ℓ the binary length k (i.e., $2^{\ell-1} \le k \le 2^{\ell} - 1$)
Output: $[k]\boldsymbol{P}$
1: $\boldsymbol{R_0} \leftarrow \boldsymbol{P}$; $\boldsymbol{R_1} \leftarrow \boldsymbol{P}$; $\ell \leftarrow \ell - 1$
2: **while** ($\ell \ne 0$) **do**
3: $\boldsymbol{R_0} \leftarrow [2]\boldsymbol{R_0}$
4: $\ell \leftarrow \ell - 1$
5: **if** ($\mathrm{bit}(k, \ell) \ne 0$) **then** $\boldsymbol{R_0} \leftarrow \boldsymbol{R_0} + \boldsymbol{R_1}$
6: **end while**
7: **return** $\boldsymbol{R_0}$

[2] We denote by $\mathrm{bit}(k, i)$ bit number i of k; bit number 0 being by definition the least significant bit.

3 Boosting the Performance

3.1 Precomputation

The observation the left-to-right binary method relies on readily extends to higher bases. We have:

$$[k]\boldsymbol{P} = \begin{cases} [2^b]\big([k/2^b]\boldsymbol{P}\big) & \text{if } 2^b \mid k \\ [2^b]\big([(k-r)/2^b]\boldsymbol{P}\big) + [r]\boldsymbol{P} \text{ with } r = k \bmod 2^b & \text{otherwise} \end{cases}.$$

The resulting method is called the 2^b-ary method and requires the prior precomputation of $[r]\boldsymbol{P}$ for $2 \leq r \leq 2^b - 1$. Observe that when r is divisible by a power of two, say $2^s \mid r$, we obviously have $[k]\boldsymbol{P} = [2^s]\Big([2^b]\big([(k-r)/2^{b+s}]\boldsymbol{P}\big) + [r/2^s]\boldsymbol{P}\Big)$. Consequently, only odd multiples of $\boldsymbol{P}$ need to be precomputed.

Other choices and optimal strategies for the points to be precomputed are discussed in [6, 2]. Further generalizations of the left-to-right binary method to higher bases, including sliding-window methods, are comprehensively surveyed in [8].

3.2 Special cases

As shown in § 2.2, a (general) point addition in Jacobian coordinates costs $11\mathsf{M}+5\mathsf{S}$. In the case $Z_2 = 1$, the addition of $(X_1 : Y_1 : Z_1)$ and $(X_2 : Y_2 : 1) = (X_2, Y_2)$ only requires $7\mathsf{M}+4\mathsf{S}$ by noting that ${Z_2}^2$, U_1 and S_1 do not need to be evaluated and that $Z_3 = Z_1 H$. The case $Z_2 = 1$ is the case of interest for the *left-to-right* binary method because the same (input) point $\boldsymbol{P}$ is added when $k_i = 1$ (cf. Line 5 in Algorithm 2).

An interesting case for point doubling is when $a_4 = -3$. Intermediate value M (cf. Eq.(3)) can then be computed as $M = 3(X_1 + {Z_1}^2)(X_1 - {Z_1}^2)$. Therefore, using the square-multiply trade-off for computing Z_3, $Z_3 = (Y_1+Z_1)^2-{Y_1}^2-{Z_1}^2$, we see that the cost of point doubling drops to $3\mathsf{M}+5\mathsf{S}$. Another (less) interesting case is when a_4 is a small constant (e.g., $a_4 = \pm 1$ or ± 2) in which case $\mathsf{c} \approx 0$ and so the point doubling only requires $1\mathsf{M} + 8\mathsf{S}$.

3.3 Signed-digit representation

A well-known strategy to speed up the evaluation of $\boldsymbol{Q} = [k]\boldsymbol{P}$ on an elliptic curves is to consider the *non-adjacent form* (NAF) of scalar k [14]. The NAF is a canonical representation using the set of digits $\{-1, 0, 1\}$ to uniquely represent an integer. It has the property that the product of any two adjacent digits is zero. Among the signed-digit representations with $\{-1, 0, 1\}$, the NAF has the smallest Hamming weight; on average, only one third of its digits are non-zero [15].

When the cost of point inversion is negligible, it is advantageous to input the NAF representation of k, $k = \sum_{i=0}^{\ell} k'_i 2^i$ with $k'_i \in \{-1, 0, 1\}$ and $k'_i \cdot k'_{i+1} = 0$, and to adapt the scalar multiplication method accordingly. For example, in Algorithm 2, Line 5, $\boldsymbol{R_1}$ is added when $k'_i = 1$ and $\boldsymbol{R_1}$ is subtracted when $k'_i = -1$. This strategy reduces the average number of point additions in the left-to-right binary method from $(\ell - 1)/2$ to $\ell/3$.

4 Fast Right-to-Left Point Multiplication

In this section, we optimize as much as possible the binary right-to-left method for point multiplication on elliptic curves over fields $\mathbb{K}$ of characteristic $\neq 2, 3$. We assume that inversion in $\mathbb{K}$ is relatively expensive compared to a multiplication in $\mathbb{K}$ and so restrict our attention to inversion-free formulæ.

We do not consider windowing techniques, which require precomputing and storing points. The targets we have in mind are constrained devices. We also wish a general method that works for *all* inputs and elliptic curves. We assume that the input elliptic curve is given by curve parameters a_4 and a_6. We have seen earlier (cf. § 3.2) that the case $a_4 = -3$ is particularly interesting because it yields a faster point doubling. We do not focus on this case because not all elliptic curves over $\mathbb{K}$ can be rescaled to $a_4 = -3$. Likewise, as we consider inversion-free formulæ, we require that the input and output points are given in projective coordinates. This allows the efficient computation of successive point multiplications. In other words, we do not assume *a priori* conditions on the Z-coordinate of input point $\boldsymbol{P}$.

In summary, we are interested in developing of a *fast, compact and general-purpose point multiplication algorithm.*

4.1 Coordinate systems

In Jacobian coordinates, a (general) point addition requires $11\mathsf{M} + 5\mathsf{S}$. In [4], Chudnovsky and Chudnovsky suggested to add two more coordinates to the Jacobian representation of points. A point $\boldsymbol{P}$ is given by five coordinates, $(X_1 : Y_1 : Z_1 : E_1 : F_1)$ with $E_1 = {Z_1}^2$ and $F_1 = {Z_1}^3$. This extended representation is referred to as the *Chudnovsky coordinates* and is abbreviated as $\mathcal{J}^c$. The advantage is that the two last coordinates (i.e., E_i and F_i) only need to be computed for the result point, saving $2(\mathsf{S} + \mathsf{M}) - 1(\mathsf{S} + \mathsf{M}) = 1\mathsf{M} + 1\mathsf{S}$ over the classical Jacobian coordinates. In more detail, from Eq. (2), including the square-multiply trade-off and "rescaling", we see that the sum $(X_3 : Y_3 : Z_3 : E_3 : F_3)$ of two (different) points $(X_1 : Y_1 : Z_1 : E_1 : F_1)$ and $(X_2 : Y_2 : Z_2 : E_2 : F_2)$ can now be evaluated as

$$\begin{aligned} X_3 = R^2 + G - 2V\,, \quad Y_3 = R(V - X_3) - S_1 G\,, \\ Z_3 = \big((Z_1 + Z_2)^2 - E_1 - E_2\big)H\,, \quad E_3 = {Z_3}^2\,, \quad F_3 = E_3 Z_3 \end{aligned} \tag{4}$$

with $R = S_1 - S_2$, $G = 4H^3$, $V = 4U_1H^2$, $S_1 = 2Y_1F_2$, $S_2 = 2Y_2F_1$, $H = U_1 - U_2$, $U_1 = X_1E_2$, and $U_2 = X_2E_1$, that is, with $10\mathsf{M} + 4\mathsf{S}$. The drawback of Chudnovsky coordinates is that doubling is slower. It is easy to see from Eq. (3) that point doubling in Chudnovsky coordinates costs one more multiplication, that is, $2\mathsf{M} + 8\mathsf{S} + 1\mathsf{c}$.

A similar approach was taken by Cohen, Miyaji and Ono [6] but to reduce the cost of point doubling (at the expense of a slower point addition). Their idea is to add a fourth coordinate, $W_1 = a_4\,{Z_1}^4$, to the Jacobian point representation $(X_1 : Y_1 : Z_1)$. This representation, called *modified Jacobian representation*, is

denoted by $\mathcal{J}^m$. With this representation, on input point $(X_1 : Y_1 : Z_1 : W_1)$, its double, $[2](X_1 : Y_1 : Z_1 : W_1)$, is given by $(X_3 : Y_3 : Z_3 : W_3)$ where the expression of X_3, Y_3 and Z_3 is given by Eq. (3) but where M and W_3 are evaluated using W_1. In more detail, we write

$$\begin{array}{c} X_3 = M^2 - 2S\,, \quad Y_3 = M(S - X_3) - 8T\,, \\ Z_3 = 2Y_1Z_1\,, \quad W_3 = 16TW_1 \end{array} \tag{5}$$

with $M = 3{X_1}^2 + W_1$, $T = {Y_1}^4$, and $S = 2[(X_1 + {Y_1}^2)^2 - {X_1}^2 - T]$. The main observation is that $W_3 := a_4\,{Z_3}^4 = 16{Y_1}^4(a_4\,{Z_1}^4) = 16TW_1$. This saves $(2\mathsf{S} + 1\mathsf{c}) - 1\mathsf{M}$. Notice that the square-multiply trade-off cannot be used for evaluating Z_3 since the value of ${Z_1}^2$ is not available. The cost of point doubling is thus $3\mathsf{M} + 5\mathsf{S}$ whatever the value of parameter a_4. The drawback is that point addition is more costly as the additional coordinate, $W_3 = a_4\,{Z_3}^4$, needs to be evaluated. This requires $2\mathsf{S} + 1\mathsf{c}$ and so the cost of point addition becomes $11\mathsf{M} + 7\mathsf{S} + 1\mathsf{c}$.

The different costs are summarized in Table 1. For completeness, we also include the cost when using affine and projective homogeneous coordinates. For affine coordinates, I stands for the cost of a field inversion.

Table 1. Cost of point addition and doubling for various coordinate systems

System	**Point addition**	**Point doubling**	
			$(a_4 = -3)$
Affine ($\mathcal{A}$)	$2\mathsf{M} + \mathsf{S} + \mathsf{I}$	$2\mathsf{M} + 2\mathsf{S} + \mathsf{I}$	—
Homogeneous ($\mathcal{H}$)	$12\mathsf{M} + 2\mathsf{S}$	$5\mathsf{M} + 6\mathsf{S} + 1\mathsf{c}$	$7\mathsf{M} + 3\mathsf{S}$
Jacobian ($\mathcal{J}$)	$11\mathsf{M} + 5\mathsf{S}$	$1\mathsf{M} + 8\mathsf{S} + 1\mathsf{c}$	$\mathbf{3M + 5S}$
Chudnovsky ($\mathcal{J}^c$)	$\mathbf{10M + 4S}$	$2\mathsf{M} + 8\mathsf{S} + 1\mathsf{c}$	$4\mathsf{M} + 5\mathsf{S}$
Modified Jacobian ($\mathcal{J}^m$)	$11\mathsf{M} + 7\mathsf{S} + 1\mathsf{c}$	$\mathbf{3M + 5S}$	—

When using projective coordinates, we see that Chudnovsky coordinates yield the faster point addition and that modified Jacobian coordinates yield the faster point doubling on any elliptic curve. We also see that point doubling in modified Jacobian coordinates is as fast as the fastest $a_4 = -3$ case with (regular) Jacobian coordinates.

4.2 Mixed representations

Rather than performing the computation in a single coordinate system, it would be interesting to consider mixed representations in the hope to get further gains. This approach was suggested in [6]. For left-to-right windowing methods with windows of width $w \geq 2$, the authors of [6] distinguish three type of operations and consider three coordinate systems $\mathcal{C}^i$, $1 \leq i \leq 3$:

1. intermediate point doubling: $\mathcal{C}^1 \to \mathcal{C}^1, \boldsymbol{R_0} \mapsto [2]\boldsymbol{R_0}$;
2. final point doubling: $\mathcal{C}^1 \to \mathcal{C}^2, \boldsymbol{R_0} \mapsto [2]\boldsymbol{R_0}$;
3. point addition: $\mathcal{C}^2 \times \mathcal{C}^3 \to \mathcal{C}^1, (\boldsymbol{R_0}, \boldsymbol{R_1}) \mapsto \boldsymbol{R_0} + \boldsymbol{R_1}$.

For inversion-free routines (or when the relative speed of I to M is slow), they conclude that the optimal strategy is to choose $\mathcal{C}^1 = \mathcal{J}^m$, $\mathcal{C}^2 = \mathcal{J}$ and $\mathcal{C}^3 = \mathcal{J}^c$.

It is worth remarking that the left-to-right binary method (Algorithm 2) and its different generalizations have in common the use of an accumulator (i.e., $\boldsymbol{R_0}$) that is repeatedly doubled and to which the input point or a multiple thereof is repeatedly added. This explains the choices made in [6]:

- the input representation of the point doubling (i.e., $\mathcal{C}^1$) is the same as the output representation of the point addition routine;
- the output representation of the (final) point doubling routine (i.e., $\mathcal{C}^2$) is the same as the input representation of [the first point of] the point addition routine;
- the input representation of [the second point of] the point addition routine (i.e., $\mathcal{C}^3$) should allow the calculation of output point in representation $\mathcal{C}^1$.

4.3 Right-to-left methods

Interestingly, the classical right-to-left method (Algorithm 1) is not subject to the same conditions: a same register (i.e., $\boldsymbol{R_1}$) is repeatedly doubled but its value is not affected by the point additions (cf. Line 3). As a result, the doubling routine can use any coordinate system as long as its output gives enough information to enable the subsequent point addition.[3] Formally, letting the three coordinate systems $\mathcal{D}^i$, $1 \leq i \leq 3$, we require the following conditions on the point addition and the point doubling routines:

1. point addition: $\mathcal{D}^1 \times \mathcal{D}^2 \to \mathcal{D}^1, (\boldsymbol{R_0}, \boldsymbol{R_1}) \mapsto \boldsymbol{R_0} + \boldsymbol{R_1}$;
2. point doubling: $\mathcal{D}^3 \to \mathcal{D}^3, \boldsymbol{R_1} \mapsto [2]\boldsymbol{R_1}$ with $\mathcal{D}^3 \supseteq \mathcal{D}^2$.

The NAF-based approach is usually presented together with the left-to-right binary method. It however similarly applies when scalar k is right-to-left scanned. Indeed, if $k = \sum_{i=0}^{\ell} k'_i 2^i$ denotes the NAF expansion of k, we can write

$$[k]\boldsymbol{P} = \sum_{0 \leq i \leq \ell} [k'_i]\big([2^i]\boldsymbol{P}\big) = \sum_{\substack{0 \leq i \leq \ell \\ k'_i \neq 0}} \mathrm{sgn}(k'_i)\boldsymbol{P_i} \quad \text{with} \begin{cases} \boldsymbol{P_0} = \boldsymbol{P} \\ \boldsymbol{P_i} = [2]\boldsymbol{P_{i-1}} \end{cases} \tag{6}$$

and where $\mathrm{sgn}(k'_i)$ denotes the sign of k'_i (i.e., $\mathrm{sgn}(k'_i) = 1$ if $k'_i > 0$ and $\mathrm{sgn}(k'_i) = -1$ if $k'_i < 0$). Note that our previous analysis on the choice of coordinate systems

[3] More generally, we require an efficient conversion from the output representation of the point doubling (say, $\mathcal{D}_3$) and the input representation of [the second point of] the point addition (say, $\mathcal{D}_2$). With the aforementioned (projective) point representations, $\{\mathcal{H}, \mathcal{J}, \mathcal{J}^c, \mathcal{J}^m\}$, for the sake of efficiency, this translates into $\mathcal{D}_3 \supseteq \mathcal{D}_2$, that is, that the coordinate system $\mathcal{D}_2$ is a subset of coordinate system $\mathcal{D}_3$.

on the (regular) right-to-left binary method remains valid for the NAF-based variant.

We are now ready to present our algorithm. The fastest doubling is given by the modified Jacobian coordinates. Hence, we take $\mathcal{D}^3 = \mathcal{J}^m$. It then follows that we can choose $\mathcal{D}^2 = \mathcal{J}^m$ or $\mathcal{J}$. As the latter leads to a faster point addition, we take $\mathcal{D}^2 = \mathcal{J}$. For the same reason, we take $\mathcal{D}^1 = \mathcal{J}$. The inputs of the algorithm are point $\boldsymbol{P} = (X_1 : Y_1 : Z_1)_{\mathcal{J}}$ given in Jacobian coordinates and scalar $k \geq 1$. The output is $[k]\boldsymbol{P} = (X_k : Y_k : Z_k)_{\mathcal{J}}$ also given in Jacobian coordinates. For further efficiency, we use a NAF representation for k and compute it on-the-fly. JacAdd$[(X^*, Y^*, Z^*), (T_1, T_2, T_3)]$ returns the sum of $(X^* : Y^* : Z^*)$ and $(T_1 : T_2 : T_3)$ as per Eq. (2), provided that $(X^* : Y^* : Z^*) \neq \pm(T_1 : T_2 : T_3)$ and $(X^* : Y^* : Z^*), (T_1 : T_2 : T_3) \neq \boldsymbol{O}$. The JacAdd routine should be adapted to address these special cases as is done e.g. in [9, § A.10.5]. ModJacDouble$[(T_1, T_2, T_3, T_4)]$ returns the double of point $(T_1 : T_2 : T_3 : T_4)$ in modified Jacobian coordinates as per Eq. (3).

Algorithm 3 Fast right-to-left binary method

Input: $\boldsymbol{P} = (X_1 : Y_1 : Z_1)_{\mathcal{J}}$, $k \geq 1$
Output: $[k]\boldsymbol{P} = (X_k : Y_k : Z_k)_{\mathcal{J}}$
1: $(X^*, Y^*, Z^*) \leftarrow (1, 1, 0)$; $(T_1, T_2, T_3, T_4) \leftarrow (X_1, Y_1, Z_1, a_4\, {Z_1}^4)$
2: **while** $(k > 1)$ **do**
3: **if** (k is odd) **then**
4: $u \leftarrow 2 - (k \bmod 4)$; $k \leftarrow k - u$
5: **if** $(u = 1)$ **then**
6: $(X^*, Y^*, Z^*) \leftarrow$ JacAdd$[(X^*, Y^*, Z^*), (T_1, T_2, T_3)]$
7: **else**
8: $(X^*, Y^*, Z^*) \leftarrow$ JacAdd$[(X^*, Y^*, Z^*), (T_1, -T_2, T_3)]$
9: **end if**
10: **end if**
11: $k \leftarrow k/2$
12: $(T_1, T_2, T_3, T_4) \leftarrow$ ModJacDouble$[(T_1, T_2, T_3, T_4)]$
13: **end while**
14: $(X^*, Y^*, Z^*) \leftarrow$ JacAdd$[(X^*, Y^*, Z^*), (T_1, T_2, T_3)]$
15: **return** (X^*, Y^*, Z^*)

Remember that we are targeting constrained devices (e.g., smart cards). In our analysis, we assume that there is no optimized squaring: $\mathsf{S}/\mathsf{M} = 1$. Also as we suppose general inputs, we also assume $\mathsf{c}/\mathsf{M} = 1$. However, to ease the comparison under other assumptions, we present the cost formulæ in their generality. We neglect field additions, subtractions, tests, etc. as is customary.

As a NAF has on average one third of digits non-zero, the expected cost for evaluating $[k]\boldsymbol{P}$ using Algorithm 3 for an ℓ-bit scalar k is

$$\frac{\ell}{3} \cdot (11\mathsf{M} + 5\mathsf{S}) + \ell \cdot (3\mathsf{M} + 5\mathsf{S}) \approx 13.33\ell\,\mathsf{M} \ . \tag{7}$$

This has to be compared with the $\frac{\ell}{3}\cdot(11\mathsf{M}+5\mathsf{S})+\ell\cdot(1\mathsf{M}+8\mathsf{S}+1\mathsf{c}) \approx 15.33\ell\,\mathsf{M}$ of the (left-to-right or right-to-left) inversion-free NAF-based binary methods using Jacobian coordinates. We gain *2 field multiplications per bit of scalar k*.

One may argue that Algorithm 3 requires one more temporary (field) variable, T_4. If *two* more temporary (field) variables are available, the classical methods can be sped up by using modified Jacobian representation; in this case, the cost becomes $\frac{\ell}{3}\cdot(11\mathsf{M}+7\mathsf{S}+1\mathsf{c})+\ell\cdot(3\mathsf{M}+5\mathsf{S}) \approx 14.33\ell\,\mathsf{M}$, which is still larger than $13.33\ell\,\mathsf{M}$. If *three* more temporary (field) variables are available, the performance of the *left-to-right* method can be best enhanced by adapting the optimal strategy of [6] as described earlier to the case $w = 1$: Input point $\boldsymbol{P}$ is then represented in Chudnovsky coordinates. This saves $1\mathsf{M}+1\mathsf{S}$ in the point addition. As a result, the cost for evaluating $[k]\boldsymbol{P}$ becomes $\frac{\ell}{3}\cdot(10\mathsf{M}+6\mathsf{S}+1\mathsf{c})+\ell\cdot(3\mathsf{M}+5\mathsf{S}) \approx 13.67\ell\,\mathsf{M} > 13.33\ell\,\mathsf{M}$.

Consequently, we see that even when further temporary variables are available, Algorithm 3 outperforms *all* NAF-based inversion-free methods without precomputation. The same conclusion holds true when considering unsigned representations for k. Replacing $\ell/3$ with $(\ell-1)/2$, we obtain $\approx 16\ell\,\mathsf{M}$ with the proposed strategy, and respectively $18\ell\,\mathsf{M}$, $17.5\ell\,\mathsf{M}$ and $16.5\ell\,\mathsf{M}$ for the other left-to-right binary methods.

In addition to efficiency, Algorithm 3 presents a couple of further advantages. Like the usual right-to-left algorithm, it is compatible with the NAF computation and does not require the knowledge of the binary length of scalar k ahead of time. Moreover, as doubling is performed using modified Jacobian coordinates, the doubling formula is independent of curve parameter a_4.

For sensitive applications, Algorithm 3 can be protected against SPA-type attacks with almost no penalty using the table-based atomicity technique of [3], as well as against DPA-type attacks using classical countermeasures.[4] Furthermore, because scalar k is right-to-left scanned, Algorithm 3 thwarts the doubling attack described in [7]. Note that, if not properly protected against, all left-to-right point multiplication methods (including the Montgomery ladder) are subject to the doubling attack.

5 Conclusion

This paper presented an optimized implementation for inversion-free point multiplication on elliptic curves. In certain settings, the proposed implementation outperforms all such previously known methods without precomputation. Further, it scans the scalar from the right to left, which offers a couple of additional advantages.

Acknowledgments I am grateful to the reviewers for useful comments.

[4] SPA and DPA respectively stand for "simple power analysis" and "differential power analysis"; see [11].

References

1. Daniel J. Bernstein and Tanja Lange. Explicit-formulas database. `http://www.hyperelliptic.org/EFD/jacobian.html`.
2. Daniel J. Bernstein and Tanja Lange. Fast scalar multiplication on elliptic curves. In Gary Mullen, Daniel Panario, and Igor Shparlinski, editors, *8th International Conference on Finite Fields and Applications*, Contemporary Mathematics. American Mathematical Society, to appear.
3. Benoît Chevallier-Mames, Mathieu Ciet, and Marc Joye. Low-cost solutions for preventing simple side-channel analysis: Side-channel atomicity. *IEEE Transactions on Computers*, 53(6):760–768, 2004.
4. David V. Chudnovsky and Gregory V. Chudnovsky. Sequences of numbers generated by addition in formal groups and new primality and factorization tests. *Advances in Applied Mathematics*, 7(4):385–434, 1986.
5. Henri Cohen. *A Course in Computational Algebraic Number Theory*, volume 138 of *Graduate Texts in Mathematics*. Springer-Verlag, 1993.
6. Henri Cohen, Atsuko Miyaji, and Takatoshi Ono. Efficient elliptic curve exponentiation using mixed coordinates. In Kazuo Ohta and Dingyi Pei, editors, *Advances in Cryptology − ASIACRYPT '98*, volume 1514 of *Lecture Notes in Computer Science*, pages 51–65. Springer, 1998.
7. Pierre-Alain Fouque and Frédéric Valette. The doubling attack - Why upwards is better than downwards. In Colin D. Walter, Çetin K. Koç, and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems − CHES 2003*, volume 2779 of *Lecture Notes in Computer Science*, pages 269–280. Springer, 2003.
8. Daniel M. Gordon. A survey of fast exponentiation methods. *Journal of Algorithms*, 27(1):129–146, 1998.
9. IEEE 1363-2000. Standard specifications for public key cryptography. IEEE Standards, August 2000.
10. Donald E. Knuth. *The Art of Computer Programming*, volume 2. Addison-Welsey, 2nd edition, 1981.
11. Paul Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In M. Wiener, editor, *Advances in Cryptology − CRYPTO '99*, volume 1666 of *Lecture Notes in Computer Science*, pages 388–397. Springer-Verlag, 1999.
12. Julio López and Ricardo Dahab. Fast multiplication on elliptic curves over GF(2^m) without precomputation. In Çetin K. Koç and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems (CHES '99)*, volume 1717 of *Lecture Notes in Computer Science*, pages 316–327. Springer, 1999.
13. Peter L. Montgomery. Speeding the Pollard and elliptic curve methods of factorization. *Mathematics of Computation*, 48(177):243–264, 1987.
14. François Morain and Jorge Olivos. Speeding up the computations on an elliptic curve using addition-subtraction chains. *RAIRO Theoretical Informatics and Applications*, 24(6):531–543, 1990.
15. George W. Reitwiesner. Binary arithmetic. *Advances in Computers*, 1:231–308, 1960.
16. Joseph H. Silverman. *The Arithmetic of Elliptic Curves*, volume 106 of *Graduate Texts in Mathematics*. Springer-Verlag, 1986.