Trading Inversions for Multiplications in Elliptic Curve Cryptography

Mathieu Ciet * (mathieu.ciet@innova-card.com)
Innova Card, Avenue Coriandre, ZI Athélia II, 13600 La Ciotat, France
http://www.innova-card.com/

Marc Joye (marc.joye@gemplus.com)
Gemplus S.A., Card Security Group, La Vigie, Avenue du Jujubier, ZI Athélia IV, 13705 La Ciotat Cedex, France
http://www.gemplus.com/smart/

Kristin Lauter (klauter@microsoft.com) and Peter L. Montgomery (petmon@microsoft.com)
Microsoft Research, One Microsoft Way, Redmond, WA 98052, USA
http://research.microsoft.com/crypto/

Abstract. Recently, Eisenträger *et al.* proposed a very elegant method for speeding up scalar multiplication on elliptic curves. Their method relies on improved formulæ for evaluating $\boldsymbol{S} = (2\boldsymbol{P} + \boldsymbol{Q})$ from given points $\boldsymbol{P}$ and $\boldsymbol{Q}$ on an elliptic curve. Compared to the naive approach, the improved formulæ save a field multiplication each time the operation is performed.

This paper proposes a variant which is faster whenever a field inversion is more expensive than six field multiplications. We also give an improvement when tripling or quadrupling a point, and present a ternary/binary method to perform efficient scalar multiplication.

Keywords: Elliptic curves, cryptography, fast arithmetic, radix-r decompositions, affine coordinates.

1. Introduction

Elliptic curve cryptography was introduced in the mid-1980s independently by Koblitz [11] and Miller [15] as a promising alternative for cryptographic protocols based on the discrete logarithm problem in the multiplicative group of a finite field (e.g., Diffie-Hellman key exchange or ElGamal encryption/signature).

Efficient elliptic curve arithmetic is crucial for cryptosystems based on elliptic curves. Such cryptosystems often require computing a scalar multiple $n\boldsymbol{P}$ of a point $\boldsymbol{P}$, where n might be 160 bits or more [1]. Various methods have been devised to this end [7]. The integer n

[*] This work was performed while Mathieu Ciet was with the UCL Crypto Group, Belgium (see http://www.dice.ucl.ac.be/crypto/), under Walloon region project Milos.

can be decomposed and written either in an integer base or using an endomorphism. In this paper we deal with the decomposition of n in an integer base.
For general elliptic curves, an improved version of scalar multiplication was proposed by Eisenträger *et al.* in [4] based on a savings obtained when doubling a point and adding it to another point on the elliptic curve. This method finds applications for decompositions signed or not, in integer bases, as well as in simultaneous multiple exponentiation.

The current paper proposes another way to compute $(2\boldsymbol{P}+\boldsymbol{Q})$ from given points $\boldsymbol{P}$ and $\boldsymbol{Q}$. Our variant is faster whenever a field inversion costs more than 6 field multiplications. We also propose a method for computing the triple $3\boldsymbol{P}$ of an elliptic curve point $\boldsymbol{P}$. Computing $3\boldsymbol{P}$ in the new way is less costly than computing $(2\boldsymbol{P}+\boldsymbol{Q})$ for general $\boldsymbol{Q}$, and so we also propose a mixed ternary/binary method for scalar multiplication to take advantage of this savings. Efficient scalar multiplication is usually performed by expressing the exponent n as a sum of (possibly negated) powers of 2 (radix-2) or another base. Here the ternary/binary method we propose refers to expressing n as a sum of products of powers of 2 and 3. We will compare the cost of a scalar multiplication using various exponent representations. Sometimes, while comparing two methods, we will assume that a field squaring costs 80% as much as a field multiplication.

The idea of finding methods for trading field inversions for field multiplications in elliptic curve cryptography has appeared previously in several papers, including [8] and [19]. We will use and in some cases improve upon those authors' results.

The paper is organized as follows. The next section presents the new methods for computing $(2\boldsymbol{P}+\boldsymbol{Q})$, $3\boldsymbol{P}$, and related operations over (large) prime fields and binary fields. Sections 3 and 4 deal respectively with radix-3 and radix-4 computations. Section 5 presents a method for combined ternary/binary scalar multiplication. Finally, Section 6 concludes the paper.

2. Radix-2 Computations

Let $\mathbb{K}$ be a field. An elliptic curve over $\mathbb{K}$ is given by the generalized Weierstraß equation

$$E : y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6 \tag{1}$$

with $a_1, a_2, a_3, a_4, a_6 \in \mathbb{K}$. When the characteristic $\mathrm{Char}\,\mathbb{K} \neq 2, 3$, one can complete the square in y and the cube in x. These transform (1) into the (short) Weierstraß form

$$E : y^2 = x^3 + a_4x + a_6 \tag{2}$$

in which $a_1 = a_2 = a_3 = 0$. Over binary (i.e., characteristic 2) fields, the short (non-supersingular) form is [1]

$$E : y^2 + xy = x^3 + a_2x^2 + a_6 \quad . \tag{3}$$

Computing $2\boldsymbol{P}+\boldsymbol{Q}$*.* Let $\boldsymbol{O}$ denote the identity element on the elliptic curve, which is taken to be the point at infinity.

Consider the short GF(p) form (2). Given two points $\boldsymbol{P} = (x_1, y_1)$ and $\boldsymbol{Q} = (x_2, y_2)$ in $E \setminus \{\boldsymbol{O}\}$ with $x_1 \neq x_2$, their sum is the point $\boldsymbol{R} = \boldsymbol{P} + \boldsymbol{Q} = (x_3, y_3)$ and is obtained by

$$\lambda_1 = \frac{y_2 - y_1}{x_2 - x_1}, \quad x_3 = \lambda_1^2 - x_1 - x_2, \quad y_3 = (x_1 - x_3)\lambda_1 - y_1 \quad .$$

Then $\boldsymbol{P}$ is added to $\boldsymbol{P}+\boldsymbol{Q}$ to form point $\boldsymbol{S} = 2\boldsymbol{P}+\boldsymbol{Q} = (x_4, y_4)$ whose coordinates are given by

$$\lambda_2 = \frac{y_3 - y_1}{x_3 - x_1}, \quad x_4 = \lambda_2^2 - x_1 - x_3, \quad y_4 = (x_1 - x_4)\lambda_2 - y_1 \quad .$$

The authors of [4] observe that the computation of y_3 can be omitted by evaluating λ_2 as

$$\lambda_2 = -\lambda_1 - \frac{2y_1}{x_3 - x_1} = \frac{2y_1}{x_1 - x_3} - \lambda_1 \quad .$$

As a result, the computation of $2\boldsymbol{P} + \boldsymbol{Q}$ only requires 2 divisions, 2 squarings and 1 (field) multiplication.

We first remark that x_4 can be obtained as

$$x_4 = (\lambda_2 - \lambda_1)(\lambda_1 + \lambda_2) + x_2 \quad .$$

Furthermore, letting $d := (x_2 - x_1)^2(2x_1 + x_2) - (y_2 - y_1)^2$, we see that $d = (x_2 - x_1)^2(x_1 - x_3)$. Defining $D := d(x_2 - x_1)$ and $I := D^{-1}$, we have

$$\frac{1}{x_2 - x_1} = dI \quad \text{and} \quad \frac{1}{x_1 - x_3} = (x_2 - x_1)^3 I \quad .$$

Consequently, the value of x_3 is not needed. The computation of d, D, I, λ_1 and λ_2 requires 1 inversion, 2 squarings and 9 (field) multiplications.

Figure 2 adapts this algorithm to the more general (1). Its last two columns count the operations (I = inversion, S = squaring, M = multiplication) needed on each line. One column has the cost for GF(p)

Input: $\boldsymbol{P} = (x_1, y_1) \neq \boldsymbol{O}$ and $\boldsymbol{Q} = (x_2, y_2) \neq \boldsymbol{O}$
Output: $\boldsymbol{S} = 2\boldsymbol{P} + \boldsymbol{Q}$

if $(x_1 = x_2)$ **then**
  if $(y_1 = y_2)$ **then return** $3\boldsymbol{P}$ **else return** $\boldsymbol{P}$
$X \leftarrow (x_2 - x_1)^2$; $Y \leftarrow (y_2 - y_1)^2$
$d \leftarrow X(2x_1 + x_2) - Y$
if $(d = 0)$ **then return** $\boldsymbol{O}$
$D \leftarrow d(x_2 - x_1)$; $I \leftarrow D^{-1}$
$\lambda_1 \leftarrow dI(y_2 - y_1)$
$\lambda_2 \leftarrow 2y_1X(x_2 - x_1)I - \lambda_1$
$x_4 \leftarrow (\lambda_2 - \lambda_1)(\lambda_1 + \lambda_2) + x_2$; $y_4 \leftarrow (x_1 - x_4)\lambda_2 - y_1$
return (x_4, y_4)

Figure 1. $(2\boldsymbol{P} + \boldsymbol{Q})$ algorithm.

fields using the short curve equation (2) and another has the cost for binary fields using (3).

For both GF(p) and binary fields, this shows that the cost of computing $2\boldsymbol{P} + \boldsymbol{Q} = (x_4, y_4)$ is at most 1 inversion, 2 squarings, and 9 (field) multiplications, which we abbreviate as $1\mathsf{I} + 2\mathsf{S} + 9\mathsf{M}$. Using equation (2), only seven registers are needed (including two unchanged registers for $\boldsymbol{P}$ and with the point $\boldsymbol{Q}$ updated in its dedicated register). See the pseudo-code in Appendix A.

Cost of non-adjacent form. The non-adjacent form (NAF) of an exponent n is

$$n = 2^{e_k} \pm 2^{e_{k-1}} \pm ... \pm 2^{e_2} \pm 2^{e_1},$$

in which $0 \leq e_1 < e_2 < \ldots < e_k$, and no two e_i are consecutive. The value of k will be about $\log_2(n)/3$ and e_k will be about $\log_2(n)$.

Point doubling is done with 1 inversion, 2 squarings and 2 (field) multiplications (assuming equation (2)). We will need e_k doublings, of which $k - 1$ are followed immediately by an add (or subtract). The overall cost is

$$\begin{aligned}(k-1)(\mathsf{I} + 2\mathsf{S} + 9\mathsf{M}) &+ (e_k - k + 1)(\mathsf{I} + 2\mathsf{S} + 2\mathsf{M}) \\ &= (k-1)(7\mathsf{M}) + e_k(\mathsf{I} + 2\mathsf{S} + 2\mathsf{M})\end{aligned}$$

which should be about

$$(\log_2(n)/3)(7\mathsf{M}) + \log_2(n)(\mathsf{I} + 2\mathsf{S} + 2\mathsf{M}) = \log_2(n)(\mathsf{I} + 2\mathsf{S} + (13/3)\mathsf{M}) \ .$$

Input: $\boldsymbol{P} = (x_1, y_1) \neq \boldsymbol{O}$ and $\boldsymbol{Q} = (x_2, y_2) \neq \boldsymbol{O}$ Output: $\boldsymbol{S} = 2\boldsymbol{P} + \boldsymbol{Q}$		
	GF(p) $a_1 = 0$	Binary $a_1 = 1$
$N_1 \leftarrow y_2 - y_1; \quad D_1 \leftarrow x_2 - x_1$		
if $(D_1 = 0)$ **then**		
if $(y_1 = y_2)$ **then return** $3\boldsymbol{P}$ **else return** $\boldsymbol{P}$		
$D_2 \leftarrow D_1^2(2x_1 + x_2 + a_2) - N_1(N_1 + a_1 D_1)$	*SMS*	*SMM*
if $(D_2 = 0)$ **then return** $\boldsymbol{O}$		
$I \leftarrow (D_1 D_2)^{-1}$	*MI*	*MI*
$\lambda_1 \leftarrow D_2 I N_1$	*MM*	*MM*
$\lambda_2 \leftarrow D_1^3(2y_1 + a_1 x_1 + a_3)I - \lambda_1 - a_1$	*MMM*	*MMM*
$x_4 \leftarrow (\lambda_2 - \lambda_1)(\lambda_2 + \lambda_1 + a_1) + x_2$	*M*	*S*
$y_4 \leftarrow (x_1 - x_4)\lambda_2 - y_1 - a_1 x_4 - a_3$	*M*	*M*
return (x_4, y_4)		
	I + 2S + 9M	I + 2S + 9M

Figure 2. $(2\boldsymbol{P} + \boldsymbol{Q})$ algorithm for generalized Weierstraß (1).

Divide by $\log_2(n)$ to get the average cost per bit using (2):

$$\boxed{\mathsf{I} + 2\mathsf{S} + (13/3)\mathsf{M}}\ .$$

The comparisons in Table I neglect pre- and post-computations.

Table I. Table of comparison for NAF on (2).

System of coordinates	Cost per bit	$\mathsf{S} = 0.8\mathsf{M}$
Affine	$\frac{4}{3}\mathsf{I} + \frac{7}{3}\mathsf{S} + \frac{8}{3}\mathsf{M}$	$1.33\mathsf{I} + 4.54\mathsf{M}$
ELM method ([4])	$\frac{4}{3}\mathsf{I} + 2\mathsf{S} + \frac{7}{3}\mathsf{M}$	$1.33\mathsf{I} + 3.93\mathsf{M}$
Our formulæ	$1\mathsf{I} + 2\mathsf{S} + \frac{13}{3}\mathsf{M}$	$1.00\mathsf{I} + 5.93\mathsf{M}$

The graph of Figure 9 in Appendix B gives break-even points. Our formulæ allow better performance than those in [4] if one inversion costs more than six (field) multiplications.

Straus-Shamir trick. Another significant and useful application of the '$2\boldsymbol{P}+\boldsymbol{Q}$' algorithm is with the Straus-Shamir trick [22, 6]. This method allows computing $a\boldsymbol{P}+b\boldsymbol{Q}$ with $\ell = \log_2(\max(|a|, |b|, 1))$ doublings and fewer than ℓ point additions if $\boldsymbol{P}\pm\boldsymbol{Q}$ are pre-computed and stored. If we suppose that a and b have the same length and that a and b are in non-adjacent form, then ℓ doublings and $(5/9)\ell$ additions are needed. In the following we refer to this decomposition as joint-NAF. In [21], Solinas introduced the Joint-Sparse-Form (JSF) that reduces the number of additions. Using the JSF, computation of $a\boldsymbol{P}+b\boldsymbol{Q}$ is done with ℓ doublings and $\ell/2$ additions. This is equivalent to $\ell/2$ applications of '$2\boldsymbol{P}+\boldsymbol{Q}$' and $\ell/2$ doublings. These joint decompositions are useful mainly for three applications: for the verification part of ECDSA [1], for the Lim-Lee method [13], and finally for the method using efficient endomorphisms proposed by Gallant, Lambert and Vanstone [5]. Table II gives the cost per bit with the various systems of coordinates and the various joint integer decompositions.

Table II. Table of comparison for joint decompositions (cost per bit) using (2).

System of coordinates	Joint-NAF		JSF	
	Cost per bit	$\mathsf{S} = 0.8\mathsf{M}$	Cost per bit	$\mathsf{S} = 0.8\mathsf{M}$
Affine	$\frac{14}{9}\mathsf{I} + \frac{23}{9}\mathsf{S} + \frac{28}{9}\mathsf{M}$	$1.56\mathsf{I} + 5.16\mathsf{M}$	$\frac{3}{2}\mathsf{I} + \frac{5}{2}\mathsf{S} + 3\mathsf{M}$	$1.50\mathsf{I} + 5.00\mathsf{M}$
ELM ([4])	$\frac{14}{9}\mathsf{I} + \frac{23}{9}\mathsf{S} + 2\mathsf{M}$	$1.56\mathsf{I} + 4.04\mathsf{M}$	$\frac{3}{2}\mathsf{I} + 2\mathsf{S} + \frac{5}{2}\mathsf{M}$	$1.50\mathsf{I} + 4.10\mathsf{M}$
Our formulæ	$1\mathsf{I} + 2\mathsf{S} + \frac{53}{9}\mathsf{M}$	$1.00\mathsf{I} + 7.49\mathsf{M}$	$1\mathsf{I} + 2\mathsf{S} + \frac{11}{2}\mathsf{M}$	$1.00\mathsf{I} + 7.10\mathsf{M}$

The break-even point is still when one inversion is equivalent to six (field) multiplications.

3. Radix-3 Computations

Computing $3\boldsymbol{P}$. When $\boldsymbol{P} = \boldsymbol{Q}$, Figure 2 does not tell us how to form $3\boldsymbol{P}$. The problem is rectified by initializing $N_1 = 3x_1^2+2a_2x_1+a_4-a_1y_1$ and $D_1 = 2y_1 + a_1x_1 + a_3$ (so N_1/D_1 is the tangent slope) rather than $N_1 = y_2 - y_1$ and $D_1 = x_2 - x_1$. If $D_1 = 0$, then $\boldsymbol{P}$ has order 2 and $3\boldsymbol{P} = \boldsymbol{P}$. Otherwise the rest of Figure 2 applies. The computation of N_1 takes one more squaring than when $x_1 \neq x_2$, but the λ_2 computation

$$\lambda_2 = D_1^3(2y_1+a_1x_1+a_3)I-\lambda_1-a_1 = D_1^3D_1I-\lambda_1-a_1 = (D_1^2)^2I-\lambda_1-a_1$$

can substitute one squaring for two multiplies (D_1^2 is known). Overall, the cost of $3\boldsymbol{P}$ is at most $1\mathsf{I} + 4\mathsf{S} + 7\mathsf{M}$, for both GF($p$) and binary fields. This is cheaper than evaluating $2\boldsymbol{P} + \boldsymbol{Q}$ for general $\boldsymbol{Q}$.

Input: $\boldsymbol{P} = (x_1, y_1) \neq \boldsymbol{O}$
Output: $\boldsymbol{T} = 3\boldsymbol{P}$

if $(y_1 = 0)$ **then return** $\boldsymbol{P}$	
$X \leftarrow (2y_1)^2$; $Z = 3x_1^2 + a_4$; $Y \leftarrow Z^2$	*SSS*
$d \leftarrow X(3x_1) - Y$	*M*
if $(d = 0)$ **then return** $\boldsymbol{O}$	
$D \leftarrow d(2y_1)$; $I \leftarrow D^{-1}$	*MI*
$\lambda_1 \leftarrow dIZ$	*MM*
$\lambda_2 \leftarrow X^2 I - \lambda_1$	*SM*
$x_4 \leftarrow (\lambda_2 - \lambda_1)(\lambda_1 + \lambda_2) + x_1$; $y_4 \leftarrow (x_1 - x_4)\lambda_2 - y_1$	*MM*
return (x_4, y_4)	

Figure 3. Tripling algorithm for GF(p) curves (2).

Moreover, only six registers are needed. See Appendix A.

Remark. Note that $d = 3x_1^4 + 6a_4x_1^2 + 12a_6x_1 - a_4^2$ ($= \psi_3(x_1, y_1)$, the 3rd division polynomial).

Computing $3\boldsymbol{P}+\boldsymbol{Q}$ *over* GF(p) *fields.* We can combine the technique to exchange an inversion for 6 (field) multiplications with the technique from [4] to save a multiply in computing $3\boldsymbol{P} + \boldsymbol{Q}$ for curves (2). If (x_4, y_4) are the coordinates of $2\boldsymbol{P} + \boldsymbol{Q}$ and (x_5, y_5) are the coordinates of $3\boldsymbol{P} + \boldsymbol{Q}$, and if $\lambda_3 = (y_4 - y_1)/(x_4 - x_1)$, then the coordinates of $3\boldsymbol{P} + \boldsymbol{Q}$ are given by $x_5 = \lambda_3^2 - x_1 - x_4$ and $y_5 = (x_1 - x_5)\lambda_3 - y_1$. The trick in [4] to save a multiply can be applied at this stage to avoid the computation of y_4 by computing λ_3 via the formula:

$$\lambda_3 = -\lambda_2 - 2y_1/(x_4 - x_1).$$

Now suppose that $2\boldsymbol{P} + \boldsymbol{Q}$ had been computed via the new method using 1 inversion, 2 squarings, and 9 (field) multiplications. Then we can still compute (x_5, y_5) without computing y_4. So one multiply is saved, computing λ_3 costs 1 inversion and 1 (field) multiplication, x_5 costs 1 squaring, and y_5 costs 1 (field) multiplication. So the total cost to compute $3\boldsymbol{P} + \boldsymbol{Q}$ this way is: 2 inversions, 3 squarings, 10 (field) multiplications, and the same trade-off applies: this is better if one inversion costs more than six (field) multiplications.

Alternatively, $3\boldsymbol{P}+\boldsymbol{Q}$ can be computed with 2 inversions, 4 squarings and 9 (field) multiplications by sharing an inversion when computing $2\boldsymbol{P}$ and $\boldsymbol{P}+\boldsymbol{Q}$, and then adding the results. We have: $3\boldsymbol{P}+\boldsymbol{Q} = (2\boldsymbol{P}) + (\boldsymbol{P}+\boldsymbol{Q})$. Let $(x_3, y_3) := 2\boldsymbol{P}$, $(x_4, y_4) := \boldsymbol{P}+\boldsymbol{Q}$ and $(x_5, y_5) := 3\boldsymbol{P}+\boldsymbol{Q}$, then $x_3 = \lambda_1^2 - 2x_1$, $y_3 = (x_1 - x_3)\lambda_1 - y_1$ with $\lambda_1 = \dfrac{3x_1^2 + a}{2y_1}$, and $x_4 = \lambda_2^2 - x_1 - x_2$, $y_4 = (x_1 - x_4)\lambda_2 - y_1$ with $\lambda_2 = \dfrac{y_1 - y_2}{x_1 - x_2}$. Computing $\lambda_c := ((2y_1)(x_1 - x_2))^{-1}$, λ_1 and λ_2 are obtained by saving one inversion and doing some extra multiplies. This approach is better than the one above since in general a squaring is less costly than a multiply.

Input: $\boldsymbol{P} = (x_1, y_1) \neq \boldsymbol{O}$ and $\boldsymbol{Q} = (x_2, y_2) \neq \boldsymbol{O}$
Output: $\boldsymbol{T} = 3\boldsymbol{P} + \boldsymbol{Q}$

---

if $(y_1 = 0)$ **then return** $\boldsymbol{P} + \boldsymbol{Q}$
if $(x_1 = x_2)$ **if** $(y_1 = y_2)$ **then return** $4\boldsymbol{P}$ else **return** $2\boldsymbol{P}$
$\lambda_c \leftarrow ((2y_1)(x_1 - x_2))^{-1}$ *MI*
$\lambda_1 \leftarrow (x_1 - x_2)(3x_1^2 + a_4)\lambda_c$ *MMS*
$\lambda_2 \leftarrow 2y_1(y_1 - y_2)\lambda_c$ *MM*
$x_3 \leftarrow \lambda_1^2 - 2x_1$; $y_3 \leftarrow (x_1 - x_3)\lambda_1 - y_1$ *MS*
$x_4 \leftarrow \lambda_2^2 - x_1 - x_2$; $y_4 \leftarrow (x_1 - x_4)\lambda_2 - y_1$ *MS*
if $(x_3 = x_4)$ **then return** $\boldsymbol{O}$
$\lambda_3 \leftarrow (y_3 - y_4)/(x_3 - x_4)$ *IM*
$x_5 \leftarrow \lambda_3^2 - x_3 - x_4$; $y_5 \leftarrow (x_3 - x_5)\lambda_3 - y_3$ *MS*
return (x_5, y_5)

Figure 4. Triple and add algorithm for GF(p) curves (2).

Computing $3\boldsymbol{P} + \boldsymbol{Q}$ *over binary fields.* The expansion $3\boldsymbol{P}+\boldsymbol{Q} = (2\boldsymbol{P}) + (\boldsymbol{P}+\boldsymbol{Q})$ works well for binary curves (3) too. This is illustrated in Figure 5. Because $2\boldsymbol{P}$ takes one fewer squaring for binary curves than for GF(p) curves, this cost is 2I + 3S + 9M, one fewer squaring than in Figure 4.

Input: $\boldsymbol{P} = (x_1, y_1) \neq \boldsymbol{O}$ and $\boldsymbol{Q} = (x_2, y_2) \neq \boldsymbol{O}$ Output: $\boldsymbol{T} = 3\boldsymbol{P} + \boldsymbol{Q}$	
if $(x_1 = 0)$ **then return** $\boldsymbol{P} + \boldsymbol{Q}$	
if $(x_1 = x_2)$ **if** $(y_1 = y_2)$ **then return** $4\boldsymbol{P}$ else **return** $2\boldsymbol{P}$	
$\lambda_c \leftarrow (x_1(x_1 + x_2))^{-1}$	*MI*
$\lambda_1 \leftarrow x_1 + (x_1 + x_2)y_1\lambda_c$	*MM*
$\lambda_2 \leftarrow x_1(y_1 + y_2)\lambda_c$	*MM*
$x_3 \leftarrow \lambda_1^2 + \lambda_1 + a_2$; $y_3 \leftarrow x_3 + (x_1 + x_3)\lambda_1 + y_1$	*SM*
$x_4 \leftarrow \lambda_2^2 + \lambda_2 + a_2 + x_1 + x_2$; $y_4 \leftarrow x_4 + (x_1 + x_4)\lambda_2 + y_1$	*SM*
if $(x_3 = x_4)$ **then return** $\boldsymbol{O}$	
$\lambda_3 \leftarrow (y_3 + y_4)/(x_3 + x_4)$	*IM*
$x_5 \leftarrow \lambda_3^2 + \lambda_3 + x_3 + x_4$; $y_5 \leftarrow x_5 + (x_3 + x_5)\lambda_3 + y_3$	*SM*
return (x_5, y_5)	

Figure 5. Triple and add algorithm for binary curves (3).

4. Radix-4 Computations

Computing $4\boldsymbol{P}$ *for* GF(p) *curves.* In [19], the authors gave a method to compute $4\boldsymbol{P}$ in 1 inversion, 9 squarings and 9 (field) multiplications. The algorithm is given in Figure 6. One multiplication has a_4 as an operand — if the curve is chosen so that a_4 is numerically small, then this multiplication can be replaced by additions.

Input: $\boldsymbol{P} = (x_1, y_1) \neq \boldsymbol{O}$ Output: $\boldsymbol{T} = 4\boldsymbol{P}$	
$A_1 \leftarrow x_1$; $B_1 \leftarrow 3x_1^2 + a_4$; $C_1 \leftarrow y_1$; $D_1 \leftarrow 12A_1C_1^2 - B_1^2$	*SMSS*
$A_2 \leftarrow B_1^2 - 8A_1C_1^2$; $B_2 \leftarrow 3A_2^2 + 16a_4C_1^4$	*SMS*
$C_2 \leftarrow B_1(4A_1C_1^2 - A_2) - 8C_1^4$; $D_2 \leftarrow 12A_2C_2^2 - B_2^2$	*MSMS*
if $(C_1C_2 = 0)$ **then return** $\boldsymbol{O}$	
$I \leftarrow (4C_1C_2)^{-1}$	*MI*
$x_4 \leftarrow (B_2^2 - 8A_2C_2^2)I^2$; $y_4 \leftarrow (B_2D_2 - 8C_2^4)I^2I$	*SMSMMM*
return (x_4, y_4)	

Figure 6. Quadrupling algorithm for GF(p) curves (2).

Computing $4\boldsymbol{P}+\boldsymbol{Q}$ *over* $\mathrm{GF}(p)$ *fields.* We compute $4\boldsymbol{P}+\boldsymbol{Q}$ as $2\,(2\boldsymbol{P})+\boldsymbol{Q}$ using our new formulæ for $2\boldsymbol{P}+\boldsymbol{Q}$. This is done with 2 inversions, 4 squarings and 11 (field) multiplications.

Total cost. The density of a such signed expansion is 3/5 (see [7]), and the length of the expansion is half that of NAF. The cost per bit is thus

$$\boxed{0.8\mathsf{I}+3\mathsf{S}+(51/10)\mathsf{M}}\ .$$

Computing $4\boldsymbol{P}$ *for binary curves.* In this subsection we propose an improvement of formulæ presented in [8]. The method proposed by Guajardo and Paar gives $4\boldsymbol{P}$ with $1\mathsf{I}+6\mathsf{S}+9\mathsf{M}$, whereas repeated doubling has complexity $2\mathsf{I}+4\mathsf{S}+4\mathsf{M}$. In characteristic two, if normal bases are used, field squarings can be neglected.

Let E be a curve with the short binary form (3) over a field of characteristic 2. Let $\boldsymbol{P}=(x_1,y_1),\boldsymbol{Q}=(x_2,y_2)\in E\backslash\{\boldsymbol{O}\}$. The negative of $\boldsymbol{P}$ is $-\boldsymbol{P}=(x_1,x_1+y_1)$. If $\boldsymbol{P}\neq-\boldsymbol{Q}$ then the sum of $\boldsymbol{P}$ and $\boldsymbol{Q}$ is given by $\boldsymbol{R}=(x_3,y_3)$ with

$$x_3=\lambda^2+\lambda+x_1+x_2+a_2,\quad y_3=\lambda(x_1+x_3)+x_3+y_1$$

where $\lambda=(y_2+y_1)/(x_2+x_1)$ if $\boldsymbol{P}\neq\boldsymbol{Q}$, or $\lambda=x_1+(y_1/x_1)$ if $\boldsymbol{P}=\boldsymbol{Q}$.

Let $\boldsymbol{P}=(x_1,y_1)$. Then $2\boldsymbol{P}=(x_2,y_2)$ is given by

$$x_2=(x_1+\frac{y_1}{x_1})^2+(x_1+\frac{y_1}{x_1})+a_2,\quad y_2=x_1^2+(x_1+\frac{y_1}{x_1})x_2+x_2\ ,$$

and $4\boldsymbol{P}=(x_3,y_3)$ is then given by

$$x_3=(x_2+\frac{y_2}{x_2})^2+(x_2+\frac{y_2}{x_2})+a_2,\quad y_3=x_2^2+(x_2+\frac{y_2}{x_2})x_3+x_3\ .$$

That means that $\dfrac{1}{x_1}$ and $\dfrac{1}{x_2}$ are needed. However, it is simple to see that

$$\frac{1}{x_2}=\frac{x_1^2}{x_1^4+a_6}\ . \tag{4}$$

Let λ_c be defined as

$$\lambda_c:=\frac{1}{x_1(x_1^4+a_6)}\ . \tag{5}$$

Then $\lambda_1:=x_1+\dfrac{y_1}{x_1}$ and $\lambda_2:=x_2+\dfrac{y_2}{x_2}$ can be obtained as

$$\lambda_1=\lambda_c\cdot(x_1^4+a_6)\cdot y_1+x_1,\quad \lambda_2=x_1\cdot y_2\cdot x_1^2\cdot\lambda_c+x_2\ .$$

Input: $\boldsymbol{P} = (x_1, y_1) \neq \boldsymbol{O}$
Output: $\boldsymbol{T} = 4\boldsymbol{P}$

if $(x_1(x_1^4 + a_6) = 0)$ **then return** $\boldsymbol{O}$
$\lambda_c \leftarrow (x_1(x_1^4 + a_6))^{-1}$ *SSMI*
$\lambda_1 \leftarrow \lambda_c(x_1^4 + a_6)y_1 + x_1$ *MM*
$x_2 \leftarrow \lambda_1^2 + \lambda_1 + a_2$; $y_2 \leftarrow x_1^2 + \lambda_1 x_2 + x_2$ *SM*
$\lambda_2 \leftarrow x_1 y_2 x_1^2 \lambda_c + x_2$ *MMM*
$x_3 \leftarrow \lambda_2^2 + \lambda_2 + a_2$; $y_3 \leftarrow x_2^2 + \lambda_2 x_3 + x_3$ *SSM*
return (x_3, y_3)

Figure 7. Quadrupling algorithm over binary field using (3).

Finally, the computation of λ_1 and λ_2 requires 1 inversion, 6 (field) multiplications and 2 squarings. This means that computation of $4\boldsymbol{P}$ requires $1\mathsf{I}+5\mathsf{S}+8\mathsf{M}$. If squarings are neglected, one (field) multiplication has been saved, and the break-even point is now $\mathsf{I} > 4\mathsf{M}$.

5. Scalar Multiplication

The fact that tripling a point is cheaper than a double and add using our techniques suggests using the operation of tripling more often while performing scalar multiplication of a point on an elliptic curve.
Table III summarizes the results from Sections 2 through 4, using the short form (2) or (3).

Table III. Table of costs for different operations.

Operation	GF(p) cost	Binary field cost
$\boldsymbol{P}+\boldsymbol{Q}$	1I + 1S + 2M	1I + 1S + 2M
$2\boldsymbol{P}$	1I + 2S + 2M	1I + 1S + 2M
$2\boldsymbol{P}+\boldsymbol{Q}$	1I + 2S + 9M	1I + 2S + 9M
$3\boldsymbol{P}$	1I + 4S + 7M	1I + 4S + 7M
$3\boldsymbol{P}+\boldsymbol{Q}$	2I + 4S + 9M	2I + 3S + 9M
$4\boldsymbol{P}$	1I + 9S + 9M	1I + 5S + 8M
$4\boldsymbol{P}+\boldsymbol{Q}$	2I + 4S + 11M	

We propose elliptic curve scalar multiplication algorithms for the situation where we want speed and aren't worried about timing attacks on the exponent (perhaps the exponent is public). Examples occur during the ECM method of factorization and while verifying an ECDSA signature.

5.1. Ternary/binary approach

The proposed algorithms evaluate expressions of the form $6\boldsymbol{P} \pm \boldsymbol{Q}$. We can compute this as $2(3\boldsymbol{P}) \pm \boldsymbol{Q}$ or $3(2\boldsymbol{P}) \pm \boldsymbol{Q}$. When using (2), the latter takes an extra inversion but saves 5 (field) multiplications. We assume $2(3\boldsymbol{P}) \pm \boldsymbol{Q}$ is better. For binary curves, the costs are $3\mathsf{I} + 4\mathsf{S} + 11\mathsf{M}$ and $2\mathsf{I} + 6\mathsf{S} + 16\mathsf{M}$, so the trade-off is 1 inversion for 2 squarings and 5 (field) multiplications.

Suppose you want $n\boldsymbol{P}$ where $\boldsymbol{P}$ is a point and $n > 0$. A possible recursive algorithm is given in Figure 8.

if $n = 1$ **then return** $\boldsymbol{P}$
switch $(n \bmod 6)$
  cases $0 \bmod 6,\ 3 \bmod 6$: **return** $3((n/3)\boldsymbol{P})$
  cases $2 \bmod 6,\ 4 \bmod 6$: **return** $2((n/2)\boldsymbol{P})$
  case $1 \bmod 6,\ n = 6m + 1$: **return** $2((3m)\boldsymbol{P}) + \boldsymbol{P}$
  case $5 \bmod 6,\ n = 6m - 1$: **return** $2((3m)\boldsymbol{P}) - \boldsymbol{P}$

Figure 8. Possible ternary/binary algorithm.

5.2. Example

As an example, compare the cost to form $314159\boldsymbol{P}$ using this ternary/binary approach as opposed to the standard binary NAF method. Note that for these comparisons, the costs for various operations are taken from Table III.

Using the combined ternary/binary mod 6 approach from Figure 8:

$314159 = 6 \cdot 52360 - 1$	triple, double-subtract
$52360 = 8 \cdot 6545$	3 doublings
$6545 = 6 \cdot 1091 - 1$	triple, double-subtract
$1091 = 12 \cdot 91 - 1$	triple, double, double-subtract
$91 = 18 \cdot 5 + 1$	triple, triple, double-add
$5 = 6 - 1$	triple, double-subtract
	6T, 4D, 5DA

Total cost is 15 inversions, 42 squarings, 95 (field) multiplications when working over GF(p). Compare this to the binary NAF method:

$314159 = 16 \cdot 19635 - 1$
$19635 = 4 \cdot 4909 - 1$
$4909 = 4 \cdot 1227 + 1$
$1227 = 4 \cdot 307 - 1$
$307 = 4 \cdot 77 - 1$
$77 = 4 \cdot 19 + 1$
$19 = 4 \cdot 5 - 1$
$5 = 4 + 1$

Using that the cost to compute $4\boldsymbol{P}+\boldsymbol{Q}$ is 2 inversions, 4 squarings, and 11 (field) multiplications, the total cost is 17 inversions, 41 squarings, 97 (field) multiplications.

The combined ternary/binary gives a 5% savings over the binary NAF method, window size 2, if one inversion costs the same as six (field) multiplications.

Remark. The combined ternary/binary can be improved by computing $5\boldsymbol{P}$ as $2(2\boldsymbol{P})+\boldsymbol{P}$. Another improvement computes the intermediate $6545\boldsymbol{P}$ using $6545 = 16(409) + 1$ and $409 = 24(17) + 1$, costing: (9I, 41S, 65M) instead of the (10I, 30S, 73M) from above.

Remark. For $17\boldsymbol{P}$, $16\boldsymbol{P}+\boldsymbol{P}$ (3I, 13S, 20M) comes out slightly better than $18\boldsymbol{P}-\boldsymbol{P}$, (3I, 10S, 23M), trading 3 multiplies for 3 squarings.

6. Conclusion

In this paper, we have proposed various strategies for efficiently evaluating $2\boldsymbol{P}+\boldsymbol{Q}$ on an elliptic curve. This outperforms a previous proposal by Eisenträger *et al.* whenever a field inversion is more expensive than six field multiplications. From this, a fast algorithm for tripling a point on an elliptic curve was derived. Furthermore, a fast algorithm for quadrupling a point was presented, improving an earlier proposal by Guajardo and Paar. Finally, we have introduced a mixed ternary/binary representation to take advantage of the aforementioned improvements, resulting in efficient methods for elliptic curve scalar multiplication, as used in ECDSA or ECDH.

References

1. IEEE Std 1363-2000. *IEEE Standard Specifications for Public-Key Cryptography*, IEEE Computer Society, August 29, 2000.
2. Michael Brown, Darrel Hankerson, Julio López, and Alfred Menezes. Software implementation of the NIST elliptic curves over prime fields. In D. Naccache, editor, *Topics in Cryptology – CT-RSA 2001*, vol. 2020 of *Lecture Notes in Computer Science*, pp. 250–265. Springer-Verlag, 2001.
3. Ian F. Blake, Gadiel Seroussi, and Nigel P. Smart. *Elliptic Curves in Cryptography*, vol. 265 of *London Mathematical Society Lecture Note Series.* Cambridge University Press, Cambridge, 2000.
4. Kirsten Eisenträger, Kristin Lauter, and Peter L. Montgomery. Fast elliptic curve arithmetic and improved Weil pairing evaluation. In M. Joye, editor, *Topics in Cryptology – CT-RSA 2003*, vol. 2612 of *Lecture Notes in Computer Science*, pp. 343–354. Springer-Verlag, 2003.
5. Robert P. Gallant, Robert J. Lambert, and Scott A. Vanstone. Faster point multiplication on elliptic curves with efficient endomorphisms. In J. Kilian, editor, *Advances in Cryptology – CRYPTO 2001*, vol. 2139 of *Lecture Notes in Computer Science*, pp. 190–200. Springer-Verlag, 2001.
6. Taher El Gamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Transactions on Information Theory*, 31(4):469–472, 1985.
7. Daniel M. Gordon. A survey of fast exponentiation methods. *Journal of Algorithms*, 27(1):129–146, 1998.
8. Jorge Guajardo and Christof Paar. Efficient algorithms for elliptic curve cryptosystems. In B.S. Kaliski Jr., editor, *Advances in Cryptology – CRYPTO '97*, vol. 1294 of *Lecture Notes in Computer Science*, pp. 342–356. Springer-Verlag, 1997.
9. Alfred J. Menezes, Paul C. van Oorschot, and Scott A. Vanstone. *Handbook of Applied Cryptography*, CRC Press Series on Discrete Mathematics and its Applications. CRC Press, Boca Raton, FL, 1997.
10. Burton S. Kaliski Jr., The Montgomery inverse and its applications, *IEEE Transactions on Computers*, 44(8):1064–1065, 1995.
11. Neal Koblitz. Elliptic curve cryptosystems. *Mathematics of Computation*, 48:203–209, 1987.
12. Çetin K. Koç and Erkay Savaş. Architectures for unified field inversion with applications in elliptic curve cryptography. In *9th IEEE International Conference on Electronics, Circuits and Systems (ICECS 2002)*, Dubrovnik, Croatia, September 15–18, 2002, vol. 3, pp. 1155–1158.
13. Chae Hoon Lim and Pil Joong Lee. More flexible exponentiation with precomputation. In Y.G. Desmedt, editor, *Advances in Cryptology – CRYPTO '94*, vol. 839 of *Lecture Notes in Computer Science*, pp. 95–107. Springer-Verlag, 1994.
14. Róbert Lórencz. New algorithm for classical modular inverse. In B.S. Kaliski Jr., Ç.K. Koç, and C. Paar, editors, *Cryptographic Hardware and Embedded Systems – CHES 2002*, vol. 2523 of *Lecture Notes in Computer Science*, pp. 57–70. Springer-Verlag, 2003.
15. Victor S. Miller. Use of elliptic curves in cryptography. In H.C. Williams, editor, *Advances in Cryptology – CRYPTO '85*, vol. 218 of *Lecture Notes in Computer Science*, pp. 417–426. Springer-Verlag, 1986.
16. Bodo Möller, private communication.

17. Peter L. Montgomery. Modular multiplication without trial division. *Mathematics of Computation*, 44(170):519–521, 1985.
18. Peter L. Montgomery. Speeding the Pollard and elliptic curve methods of factorization. *Mathematics of Computation*, 48(177):243–264, 1987.
19. Yasuyuki Sakai and Kouichi Sakurai, Efficient scalar multiplications on elliptic curves with direct computations of several doublings. *IEICE Transactions Fundamentals*, E84-A(1):120–129, 2001.
20. Erkay Savaş and Çetin K. Koç. The Montgomery modular inverse - revisited, *IEEE Transactions on Computers*, 49(7):763–766, 2000.
21. Jerome A. Solinas. Low-weight binary representations for pairs of integers. *Tech. Report CORR 2001/41*, CACR, Waterloo, 2001.
22. Ernst G. Straus. Addition chains of vectors (problem 5125). *American Mathematical Monthly*, 70:806–808, 1964.

Appendix

A. Pseudo-code

Let (x_1, y_1) and (x_2, y_2) be two points on the short GF(p) curve (2). The following algorithm updates (x_1, y_1) with $2(x_1, y_1) + (x_2, y_2)$. Registers are denoted by T_i.

$T_1 \leftarrow x_1$; $T_2 \leftarrow y_1$; $T_3 \leftarrow x_2$; $T_4 \leftarrow y_2$

$T_5 \leftarrow 2T_1$; $T_5 \leftarrow T_5 + T_3$	$(= 2x_1 + x_2)$
$T_1 \leftarrow T_3 - T_1$	$(= x_2 - x_1)$
$T_6 \leftarrow {T_1}^2$	$(= (x_2 - x_1)^2)$
$T_5 \leftarrow T_5 \cdot T_6$	$(= (2x_1 + x_2)(x_2 - x_1)^2)$
$T_6 \leftarrow T_1 \cdot T_6$	$(= (x_2 - x_1)^3)$
$T_4 \leftarrow T_4 - T_2$	$(= y_2 - y_1)$
$T_7 \leftarrow {T_4}^2$	$(= (y_2 - y_1)^2)$
$T_5 \leftarrow T_5 - T_7$	$(= d)$
$T_7 \leftarrow T_5 \cdot T_1$; $T_7 \leftarrow {T_7}^{-1}$	$(= I)$
$T_5 \leftarrow T_5 \cdot T_7$; $T_5 \leftarrow T_5 \cdot T_4$	$(= \lambda_1)$
$T_6 \leftarrow T_6 \cdot T_7$	$(= (x_2 - x_1)^3 I)$
$T_7 \leftarrow 2T_2$; $T_7 \leftarrow T_7 \cdot T_6$; $T_7 \leftarrow T_7 - T_5$	$(= \lambda_2)$
$T_4 \leftarrow T_3 - T_1$	$(= x_1)$
$T_6 \leftarrow T_7 - T_5$	$(= \lambda_2 - \lambda_1)$
$T_5 \leftarrow T_7 + T_5$	$(= \lambda_2 + \lambda_1)$
$T_1 \leftarrow T_6 \cdot T_5$; $T_1 \leftarrow T_1 + T_3$	$(= x_4)$
$T_4 \leftarrow T_4 - T_1$; $T_4 \leftarrow T_4 \cdot T_7$	
$T_2 \leftarrow T_4 - T_2$	$(= y_4)$

It is worth noticing that only seven registers are needed. This count omits registers needed internally by the field arithmetic codes.

Let (x_1, y_1) be a point on the short GF(p) curve (2). The following algorithm updates registers with $3(x_1, y_1)$.

$$
\begin{array}{ll}
T_1 \leftarrow x_1;\ T_2 \leftarrow y_1;\ T_5 \leftarrow a_4 & \\
T_3 \leftarrow 2\,T_2;\ T_3 \leftarrow T_3^2 & (=X) \\
T_4 \leftarrow T_1^2;\ T_4 \leftarrow 3\,T_4;\ T_4 \leftarrow T_4 + T_5 & (=Z) \\
T_5 \leftarrow T_4^2 & (=Y) \\
T_6 \leftarrow 3\,T_1;\ T_6 \leftarrow T_6 \cdot T_3;\ T_5 \leftarrow T_5 - T_6 & (=-d) \\
T_4 \leftarrow T_4 \cdot T_5;\ T_6 \leftarrow 2\,T_2;\ T_5 \leftarrow T_5 \cdot T_6 & (=-D) \\
T_5 \leftarrow T_5^{-1} & (=-I) \\
T_4 \leftarrow T_4 \cdot T_5 & (=\lambda_1) \\
T_3 \leftarrow T_3^2;\ T_5 \leftarrow T_3 \cdot T_5;\ T_3 \leftarrow T_5 + T_4 & (=-\lambda_2) \\
T_4 \leftarrow (T_3 + T_4);\ T_4 \leftarrow T_4 \cdot T_5; T_1 \leftarrow T_4 + T_1 & (=x_4) \\
T_3 \leftarrow T_4 \cdot T_3;\ T_2 \leftarrow T_3 - T_2 & (=y_4)
\end{array}
$$

Tripling a point is done with only six intermediate registers.

B. Break-even Point

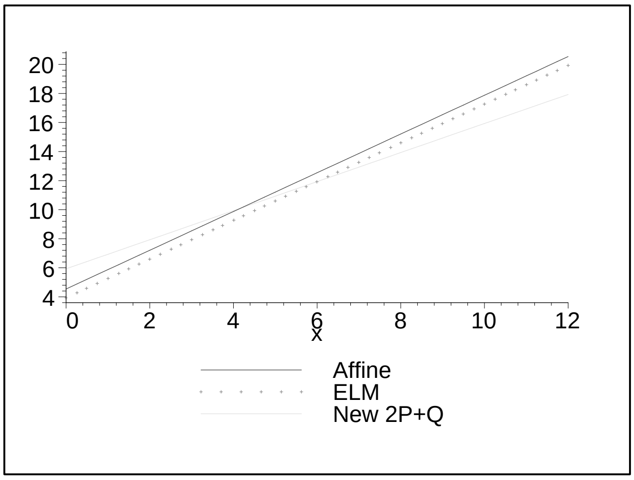

Figure 9. Comparison for NAF.

C. Radix-4 Computation: Right-to-left

Assume we are using (2). As illustrated in the above technique for computing $3\boldsymbol{P}+\boldsymbol{Q}$, a point addition $\boldsymbol{P}+\boldsymbol{Q}$ and a doubling $2\boldsymbol{P}$ can be done simultaneously, exchanging two inversions for 1 inversion and 3 (field) multiplications. This was pointed out in [18], [4], and in [16]. In this way, computing both $\boldsymbol{P}+\boldsymbol{Q}$ and $2\boldsymbol{Q}$ can be done in 1 inversion, 3 squarings and 7 (field) multiplications. Then, the cost per bit is

$$\boxed{1\mathsf{I}+(7/3)\mathsf{S}+(11/3)\mathsf{M}}\ .$$

However, this does not take into account the fact that we have a NAF. This especially implies that the update of $\boldsymbol{Q}$ into $2\boldsymbol{Q}$ can be replaced by updating $\boldsymbol{Q}$ into $4\boldsymbol{Q}$ and then not jumping to the next bit but the following. Then, the following cost per bit is obtained

$$\boxed{2/3\mathsf{I}+4\mathsf{S}+(16/3)\mathsf{M}}\ .$$

If we consider that $\mathsf{S}=0.8\mathsf{M}$, the break-even point is $\mathsf{I}>9\mathsf{M}$.

Remark. This is not surprising since we use the results of Sakai-Sakurai [19] to compute $4\boldsymbol{Q}$ and the break-even point compared with repeated doubling is $\mathsf{I}>9\mathsf{M}$.

D. Inversion over a Finite Field

This section briefly deals with inversion of a finite field element. Let a be a nonzero element of GF(p), where p is prime. Let a^{-1} denote its multiplicative inverse. There are several ways to compute this inverse.

One method uses a table of length $p-1$. This is feasible only for small p. It can be fast if the table fits in cache.

Another is based on Fermat's theorem: $a^{-1}=a^{p-2}$. At first glance this 'trivial' method seems to be much too costly. However, it has some interesting aspects. No extra routine is needed. Moreover, p can be a Mersenne or generalized Mersenne prime for increased efficiency of modular reduction [1]. Further, if we suppose that p is a generalized Mersenne prime, say $p=2^{\kappa_1}-2^{\kappa_2}-1$, then $a^{-1}=a^{2^{\kappa_1}-2^{\kappa_2}-3}$ and smart-card routines can be used to speed-up repeated squarings.

A third method is based on the extended Euclidean algorithm, which given two integers a and p, outputs u and v such that $au+pv=\gcd(a,p)$. If a is invertible modulo p and if $0\le u<p$, then $\gcd(a,p)=1$ and $a^{-1}=u$. An improvement to the extended Euclidean algorithm due to Lehmer is explained in [9, p. 607].

A fourth method proceeds in two steps and is based on the well-known Montgomery multiplication. Let a and b be two integers between 0 and $p-1$. Montgomery multiplication fixes an exponent k such that

$p < 2^k$ and returns $a\,b\,2^{-k} \bmod p$. The Montgomery inverse is defined (by Kaliski in [10] based on [17], see also [20]) as

$$x := a^{-1}2^k \ .$$

The regular inverse a^{-1} is obtained by computing the Montgomery product of x and 1 (see [20] for variants), see also [14]. If one has an algorithm for a^{-1}, then one can get $x = (a2^{-k})^{-1}$ by inverting the Montgomery product of x and 1.

Estimates for the cost of a field inversion in terms of field multiplications dramatically depend on the architecture used and the size and type of the field. Equivalences for field element inversion vary between 4 field multiplications in [4] and [12] to 80 field multiplications in [2]. The ratio of 80 takes into account the use of special modular reduction routines to speed multiplication in prime fields where the prime is of a special form (generalized Mersenne prime), and does not take into account Lehmer's method for speeding modular inversion. A discussion of the ratio in various contexts can also be found in [3, p. 72].