On Montgomery-Like Representations for Elliptic Curves over $GF(2^k)$

Martijn Stam [⋆,⋆⋆]

Technische Universiteit Eindhoven
P.O.Box 513, 5600 MB Eindhoven, The Netherlands
stam@win.tue.nl

Abstract. This paper discusses representations for computation on non-supersingular elliptic curves over binary fields, where computations are performed on the x-coordinates only. We discuss existing methods and present a new one, giving rise to a faster addition routine than previous Montgomery-representations. As a result a double exponentiation routine is described that requires 8.5 field multiplications per exponent bit, but that does not allow easy y-coordinate recovery. For comparison, we also give a brief update of the survey by Hankerson et al. and conclude that, for non-constrained devices, using a Montgomery-representation is slower for both single and double exponentiation than projective methods with y-coordinate.

Keywords: ECC, Montgomery, point multiplication, Lucas chains

1 Introduction

Since the introduction of elliptic curve cryptography in the mid 1980s, many proposals have been made to speed up the group arithmetic. There are essentially three ways to achieve this: speed up the arithmetic in the underlying field (e.g., binary, prime, optimal extension fields [3]), pick a convenient representation of the group elements (e.g., affine, projective, Chudnovsky, 'mixed coordinates' [7]), or choose a short addition chain (e.g., non-adjacent form, Frobenius-expansions).

The effects of the three possible choices are certainly not independent as demonstrated by for instance [3, 7, 11]. This is in particular the case if the so-called Montgomery representation is used [21]. This representation was introduced in 1987 to speed up implementation of the elliptic curve integer factoring method, but has been relatively uncommon in cryptographic applications. For the Montgomery representation the general elliptic curve equation

$$E : Y^2 + a_1XY + a_3Y = X^3 + a_2X^2 + a_4X + a_6$$

[⋆] The author is sponsored by STW project EWI.4536.

[⋆⋆] Part of the work was done while visiting BRICS, Center of the Danish National Research Foundation, Aarhus University, Denmark.

over any finite field is replaced by

$$E_M : BY^2 = X^3 + AX^2 + X$$

over finite fields of odd characteristic. Because of its intended application in the elliptic curve integer factoring method, the Montgomery representation was specifically designed to speed up the calculation of the x-coordinate of nP, for large integers n and points P on the curve E_M. Montgomery's representation is characterized not so much by the particular form of the curve-equation, but mostly by the facts that to add two points their difference must be known and that the y-coordinate is absent. Furthermore, the order of the elliptic curve group must be divisible by 4 [24]. These facts have to be taken into account when one tries to take advantage of the fast Montgomery representation based computation of nP in a cryptographic context. For instance, the divisibility by 4 rules out the so-called NIST curves [23].

It is well known that for most cryptographic protocols a y-coordinate is not really needed; for instance, in Diffie-Hellman it adds only a single bit and ECDSA [10] can be run with just x-coordinates. Nevertheless, if the y-coordinate of some point P is needed, it can be computed if P's x-coordinate is known along with the x-coordinate of some other point Q and both the x and y coordinates of $P - Q$. Whether or not these data are available depends on the way P is computed. In the Montgomery representation, and assuming P is the result of a scalar multiplication, P is computed using a second order recurrence in the x-coordinate known as a Lucas chain, because to add two points their difference must be known. If a binary Lucas chain is used the difference is fixed (and known), so that the y-coordinate can be recovered [21, 25]; it has the additional benefit that the addition cost can be reduced by choosing some of the denominators as one.

The difference is not fixed (and in general not known) if a continued fraction based Lucas chain is used. As a result it is no longer possible to recover the y-coordinate in an efficient manner, but such chains give rise to a very fast double exponentiation algorithm [20, 27]. Slower double exponentiation with y-coordinate recovery can be achieved using an algorithm due to Akishita [2].

For elliptic curves over fields of characteristic two, the traditional Montgomery representation based on the curve equation E_M does not work, because the curve isomorphism requires a division by 2. Adaptation of Montgomery's representation to fields of characteristic two based on the ordinary shortened Weierstrass form for non-supersingular curves is considered in [1, 17, 30], resulting in a reasonably fast single exponentiation routine without precomputation.

In this paper we further optimize the method from [17] by introducing an alternative curve equation. We show that it saves a finite field multiplication for general point addition.[1] Compared to [17] this leads to a speedup of about 15% for double exponentiation. Furthermore, we investigate the consequences for

[1] This situation is reminiscent of [6, 9] where Lucas chains are given for x-coordinates of general Weierstrass curves (including the NIST curves). It leads to relatively slow scalar multiplication. Optimization of those formulae leads to Montgomery's results

single exponentiation with precomputation, y-coordinate recovery, and two simultaneous exponentiations. Our methods apply to all non-supersingular curves over $\mathbf{F}_{2^k}$, irrespective of the group order.

The Montgomery representation is said to have three possible advantages: it is fast, requires only few registers in memory and can serve as a hedge against timing and power analysis attacks. From our results and comparison with other work, we conclude that in the binary case the Montgomery representation is not as fast as regular methods, both for single and double exponentiation. Furthermore, the fastest Montgomery representation approach to either type of exponentiation uses continued fraction based Lucas chains; as a consequence the protection against timing and power analysis attacks is lost. So, despite the fact that our results improve on previous results in this area, we conclude that the use of Montgomery representations for elliptic curves over binary fields can hardly be recommended, unless our results can be improved upon. Only if memory usage or timing and power analysis are of serious concern, the Montgomery representations regain their attractiveness.

In Section 2 we review the traditional way of doing elliptic curve arithmetic by means of projective coordinates. In Section 3 we review known Montgomery representations and introduce a new one, that requires one field multiplication fewer for a point addition. In Section 4 exponentiation routines suitable for the Montgomery representation are analyzed and compared. In Section 5 we present our conclusions.

2 Elliptic Curves over $\mathbf{F}_{2^k}$

Before discussing curves over $\mathbf{F}_{2^k}$, we briefly discuss field arithmetic of $\mathbf{F}_{2^k}$ itself. The additive group can usually be implemented using the exclusive or of two elements. The runtime of an algorithm is typically determined by the number of multiplicative operations required. There are three important multiplicative operations, namely squaring of an element, multiplication of two elements, and inversion of an element. Squarings are so cheap that they are not counted and inversions are so expensive that they are as much as possible avoided. The cost is measured by the number of $\mathbf{F}_{2^k}$-multiplications, and it is assumed that an inversion costs the same as 10 multiplications to ease comparison with [8].

2.1 Curve Definition

An elliptic curve over $\mathbf{F}_{2^k}$ is the set of points $(X, Y) \in (\mathbf{F}_{2^k})^2$ satisfying the long Weierstrass equation

$$E : Y^2 + a_1XY + a_3Y = X^3 + a_2X^2 + a_4X + a_6 \tag{1}$$

together with a point at infinity, denoted $\mathcal{O}$. The coefficients $a_i, i \in \{1, 2, 3, 4, 6\}$, are taken from $\mathbf{F}_{2^k}$. An elliptic curve forms an abelian finite group under the

from [21] with the same restriction on the curve group order (but slightly more freedom in the curve equation).

addition operation also known as the chord-tangent process. The point at infinity, $\mathcal{O}$, serves as group identity and the negation of a point (X, Y) is given by $(X, -Y - a_1X - a_3)$.

In the literature several constants related to the a_i are defined; we recall the following three:

$$\begin{aligned} b_8 &= a_1^2 a_6 + a_1 a_3 a_4 + a_2 a_3^2 + a_4^2 \; ; \\ \Delta &= a_1^4 b_8 + a_3^4 + a_1^3 a_3^3 \; ; \\ j &= a_1^{12}/\Delta \; . \end{aligned}$$

Here Δ is called the discriminant and the j stands for j-invariant. A curve is singular iff $\Delta = 0$; henceforth we will assume $\Delta \neq 0$. The j-invariant characterizes isomorphism classes over $\bar{\mathbf{F}}_{2^k}$: Two curves are isomorphic over $\bar{\mathbf{F}}_{2^k}$ if and only if their j-invariants are the same. A curve is supersingular iff $\Delta \neq 0$ and $j = 0$.

Often the long Weierstrass equation is replaced by the following short Weierstrass equation for non-supersingular curves over fields of binary characteristic:

$$E : Y^2 + XY = X^3 + a_2X^2 + a_6 \; . \tag{2}$$

For every curve of the form (1) there is an isomorphic curve of the form (2). Moreover, if the extension degree k is odd, a_2 in (2) can be taken either 0 or 1. Hence multiplications by a_2 may be neglected. A curve of the form (2) has discriminant a_6 and j-invariant $1/a_6$.

We assume we are working in a cyclic subgroup of size q with generator P and $\log_2 q \approx k$. As customary we use additive notation for the group operation.

2.2 Curve Arithmetic

Throughout this article, we let P_i for $0 < i \leq 5$ be points on the curve and assume that $P_3 = P_1 + P_2, P_4 = P_1 - P_2$, and $P_5 = 2P_1$. Moreover, we assume that $\mathcal{O}$ is not among the P_i. This allows us to write $P_i = (X_i, Y_i)$ in affine coordinates or $P_i = (x_i, y_i, z_i)$ in projective coordinates, usually with $X_i = x_i/z_i$ and $Y_i = y_i/z_i$. Note that upper case characters are used for affine coordinates, and lower case ones for projective coordinates. The following relations hold (see e.g., [4]):

$$X_3 = (\frac{Y_1 + Y_2}{X_1 + X_2})^2 + a_1(\frac{Y_1 + Y_2}{X_1 + X_2}) + X_1 + X_2 + a_2 \tag{3}$$

$$X_4 = (\frac{Y_1 + Y_2 + a_1X_2 + a_3}{X_1 + X_2})^2 + a_1(\frac{Y_1 + Y_2 + a_1X_2 + a_3}{X_1 + X_2}) + X_1 + X_2 + a_2 \tag{4}$$

$$X_5 = \frac{X_1^4 + a_1a_3X_1^2 + b_8}{(a_1X_1 + a_3)^2} \; . \tag{5}$$

Fast arithmetic on elliptic curves has been well studied. A nice overview can be found in [8]. Skewed projective coordinates $(x/z, y/z^2)$ are the most efficient

Table 1. Expected number of field multiplications for point multiplication given a k-bit exponent

Type	Method	Given in [8]	Improvement
single	windowed NAF, $w = 4$	$5.8k + 60$	$5.6k + 60$
fixed single	fixed base comb, $w = 4$	$3.11k + 8$	$2.88k + 9$
semi-fixed double	comb+Montgomery	$9.11k + 40$	n.a.
semi-fixed double	Möller, $w = 4$	n.a.	$7.24k + 60$
double	Möller, $w = 4$	n.a.	$7.24k + 108$
double	Solinas	n.a.	$8k + 36$

representation known [16, 11]. Using skewed projective coordinates, a general additions costs 14 field multiplications. However, if one of the points is given in affine coordinates, this drops to 9 field multiplications. Doubling a point costs 4 field multiplications. King [11] notes that, if the resulting point of an addition is subsequently doubled, a multiplication can be saved at the cost of a squaring. For a large class of popular exponentiation routines, this effectively reduces the cost of an addition to 8 field multiplications.

2.3 Exponentiation (or Scalar Multiplication)

The survey [8] contains several exponentiation routines. Three different settings are examined: single exponentiation with a fixed base; single exponentiation with an arbitrary base and double exponentiation with one base fixed and the other arbitrary. This choice seems motivated by the facts that signature generation is a fixed base exponentiation and signature verification is a double exponentiation with one base fixed. However, with the introduction of fast point counting algorithms it becomes more realistic to deviate from the NIST curves, which is why we also consider double exponentiations with both bases arbitrary. Having a fixed base can ease computation by precomputing certain values.

For single exponentiation [8] proposes a windowed NAF with window size 4 (note that using Montgomery representation is reported to be faster). This results in approximately 1 point doubling and $\frac{1}{5}$ point additions per exponent bit. Moreover, 3 points have to be precomputed, namely $3P, 5P$, and $7P$. This has to be done affinely and costs 8 multiplications and 4 inversions. If the base is fixed, [8] proposes a fixed base comb of size 4. An exponentiation will cost $\frac{1}{4}$ point doublings and $\frac{15}{64}$ point additions per exponent bit on average. The number of precomputed points is 14. Note that this method does not exploit cheap point negation. The double exponentiation routine presented assumes one base is fixed and simply consists of two separate single exponentiation routines and multiplying the result (using a fixed base comb and Montgomery respectively).

A double exponentiation without any fixed bases is not addressed in [8]. However, using Solinas' trick, it will require 1 point doubling and $\frac{1}{2}$ point additions per bit of the longest exponent. Interestingly, this method already outperforms the double exponentiation routine with fixed base given in [8]. Möller [19] proposes to interleave two windowed NAF routines thereby saving one set of point doublings. For window size 4 this results in 8 points to be precomputed affinely and 1 point doubling and $\frac{2}{5}$ point additions per exponent bit.[2] Note that King's improvement does not apply in approximately $\frac{1}{25}$ of the additions. Of the 8 points to be precomputed, half can be done in advance if the base is fixed.

Table 1 summarizes these results. We give the number of field multiplications for a k-bit exponent, according to [8] and with inclusion of the known speedups just described. We use the same ratio, that one inversion costs the same as 10 multiplications. The extra costs needed at the end to convert back to affine coordinates are also included. For $k = 163$, compare with [8, Table 6].

3 The Montgomery Representation

The Montgomery representation was introduced as part of a speedup of the elliptic curve factoring method. Hence it was specifically tailored for curves over large prime fields. (Actually, over large rings $\mathbf{Z}_n^*$, where failure to invert would constitute a factoring success.) The connection with elliptic curve cryptography was made later.

We first review known methods of computing without y-coordinates for curves over binary fields. Interestingly, these methods are all based on a relationship for $X_3 + X_4$, hence the name additive Montgomery formulae. Next, we analyse methods based on a relationship for X_3X_4, called multiplicative Montgomery formulae. The terminology additive and multiplicative method is also used in [9] for curves over large prime fields. We conclude with a small word on recovering the y-coordinate based on [17].

3.1 Additive Montgomery Formulae

Agnew et al. [1] mention computing with x-coordinates only for non-supersingular curves over large extension fields of characteristic two. By using the curve equation (1) it is possible to rewrite (3) as

$$X_3 = \frac{Y_1(a_1X_2 + a_3) + Y_2(a_1X_1 + a_3) + (X_1 + X_2)(a_4 + X_1X_2)}{(X_1 + X_2)^2} . \tag{6}$$

For X_4 a similar formula can be obtained by replacing Y_2 in (6) with $Y_2 + a_1X_2 + a_3$, the y-coordinate of $-P_2$. This shows that

$$X_3 + X_4 = \frac{(a_1X_1 + a_3)(a_1X_2 + a_3)}{(X_1 + X_2)^2} . \tag{7}$$

For the shortened Weierstrass form the above is simplified by setting $a_1 = 1$ and $a_3 = 0$. The projective version presented in [1] based on these formulae is incorrect.

[2] Apparently [8] and [19] use different definitions for window sizes. We adhere to the first.

In 1999 correct projective versions for non-supersingular curves appeared in [17] and [30], both using the short Weierstrass form. In both works the following formulae are given, easily verified by (7) and (5):

$$\begin{aligned} \frac{x_3}{z_3} &= \frac{x_4(x_1z_2+x_2z_1)^2+z_4(x_1z_2)(x_2z_1)}{z_4(x_1z_2+x_2z_1)^2}\ ; \\ \frac{x_5}{z_5} &= \frac{x_1^4+a_6z_1^4}{x_1^2z_1^2}\ . \end{aligned} \tag{8}$$

Both [17] and [30] emphasize on an addition where the difference P_4 is fixed, and hence z_4 in (8) is set to 1. Such a fixed difference addition takes 4 field multiplications and 1 squaring.[3] It seems that for an ordinary addition one needs 6 field multiplications and 1 squaring. A point doubling takes 2 multiplications and, if $a_6^{1/4}$ is precomputed, 3 squarings.[4] It is also noted that having a small $a_6^{1/4}$ can reduce the cost of multiplication with $a_6^{1/4}$ considerably, and therefore of a point doubling. A similar argument holds for fixed difference addition if x_4 is small.

Affine versions are also presented, but not worked out in full since the inversions seem to deter. It is said that a point addition takes 2 multiplications, a squaring and an inversion, whereas a doubling takes one multiplication less.

3.2 Multiplicative Montgomery Formulae

Originally, Montgomery derived his formula for curves over large prime characteristic by multiplying X_3 and X_4, not by considering their difference. Not surprisingly, a multiplicative version of Montgomery's trick also proves possible for curves over binary characteristic. For a non-supersingular curve, taking $a_3 = 0$, one obtains:

$$\begin{aligned} \frac{x_3x_4}{z_3z_4} &= \frac{x_1^2x_2^2+b_8z_1^2z_2^2}{(x_1z_2+x_2z_1)^2}\ ; \\ \frac{x_5}{z_5} &= \frac{x_1^4+b_8z_1^4}{(a_1x_1z_1)^2}\ . \end{aligned}$$

In the short Weierstrass form a_1 will be 1, but b_8 can be any field element. Hence, computing x_3 and z_3 will take 6 field multiplications,[5] one less if P_4 is fixed and $z_4 = 1$. Setting $b_8 = 1$ will reduce these costs to 5 respectively 4 field multiplications, while at the same time keeping the costs for a point doubling the same. As an alternative, one could consider using the representation (x, z, xz). This requires the same number of multiplications though, so should therefore not be recommended.

[3] For some reason [30] report 2 squarings.

[4] And here [17] use 5 squarings, which is more than needed even without precomputation of $a_6^{1/4}$.

[5] Compute $(x_1z_2 + x_2z_1)$ as $(x_1 + z_1)(x_2 + z_2) - x_1x_2 - z_1z_2$.

If $a_3 = 0$, then $b_8 = a_1^2 a_6 + a_4^2$. Hence, $b_8 = 1$ can be achieved by setting $a_4 = 0$ and $a_6 = 1/a_1^2$. We therefore propose working on elliptic curves of the form

$$E : Y^2 + a_1 XY = X^3 + a_2 X^2 + 1/a_1^2 \ , \tag{9}$$

where a_1 and a_2 are in $\mathbf{F}_{2^k}$. To ensure we do not exclude any interesting curves we present the following lemma.

Lemma 1 *Any non-supersingular curve over $\mathbf{F}_{2^k}$ is isomorphic to a curve over $\mathbf{F}_{2^k}$ of the form (9).*

Proof. Recall that all non-supersingular curves have a representation of the form (2), having j-invariant $1/a_6$. A curve of the form (9) has j-invariant equal to a_1^8. Squaring is a permutation on $\mathbf{F}_{2^k}$ whence all elements have a unique eighth root in $\mathbf{F}_{2^k}$. Set $a_1 = a_6^{-1/8}$ and let $s \in \mathbf{F}_{2^k}$. Consider the admissible change of variables given by

$$\begin{aligned} x &= X/a_1^2, \\ y &= sX/a_1^2 + Y/a_1^3 \ . \end{aligned}$$

This gives an isomorphism over $\mathbf{F}_{2^k}$ of the curve given by (2) and

$$1/a_1^2 + a_1^2(a_2 + s + s^2)x^2 + x^3 = a_1 xy + y^2 \ . \tag{10}$$

It is easily verified that (10) is of the form (9).

3.3 Recovery of the y-Coordinate

The curve equation provides a way to determine the y-coordinate of a given point using a square root computation. This method is relatively expensive and one still needs to address the square root ambiguity. An alternative is presented in [17]. If two points P_1 and P_2 are given by x-coordinate only and their difference P_4 is fully specified, either unknown y-coordinate can be retrieved in a small number of field multiplications. Moreover, there is no square root ambiguity. The method from [17] to recover the y-coordinate is described in more detail below (slightly generalized).

Given X_1, X_2, X_4 and Y_4, it is possible to determine Y_2 efficiently. From formula (3) it follows how to determine Y_1 if X_1, X_2, X_3 and Y_2 are given, by using the curve equation (1) to get rid of the quadratic term Y_1^2. However, since the values $(P, Q, P - Q)$ have the same additive relation to each other as the values $(P + Q, P, Q)$, the desired result follows from a suitable re-indexing:

$$Y_1 = \frac{(X_2 + X_4)(a_4 + X_2 X_4 + X_1 X_2 + X_1 X_4) + (a_3 + a_1 X_2) Y_4}{a_3 + a_1 X_4} \ .$$

Note that $a_3 + a_1 X_4 = 0$ iff $P_4 = \mathcal{O}$, which we assumed not to be the case.

For $a_3 = 0$ and $a_4 = 0$ and projective coordinates, the formula in [17] can be generalized to

$$X_1 = \frac{a_1 z_4 x_4 z_2 x_1}{a_1 z_4 x_4 z_2 z_1} \; ; \tag{11a}$$

$$Y_1 = y_4 + \frac{(x_2 z_4 + x_4 z_2)((x_1 z_4 + x_4 z_1)(x_2 z_4 + x_4 z_2) + z_1 z_2 x_4^2 + a_1 z_1 z_2 z_4 y_4)}{a_1 z_1 z_2 x_4 z_4} \; . \tag{11b}$$

Simultaneous recovery of X_1 and Y_1 will therefore cost at most 15 multiplications and 1 inversion. If $z_4 = 1$, this reduces to 11 multiplications and 1 inversion. A further reduction to 10 multiplications and 1 inversion is achieved if $a_1 = 1$, which is the case for the shortened Weierstrass form.

4 Exponentiation, Aka Scalar Multiplication

4.1 Lucas Chains

Suppose we are given a point P and we want to determine nP, where n is a random element in $\mathbf{Z}_q$. Using a traditional curve representation this can efficiently be done using short addition-subtraction chains. An addition-subtraction chain for an integer $n > 0$ is a sequence $a_0, a_1, \ldots, a_l$ with $a_0 = 1, a_l = n$ and for all $0 < k \leq l$ there should exist $0 \leq i, j < k$ such that $a_k = a_i + a_j$ or $a_k = a_i - a_j$. There is extensive literature concerning addition chains [12] and addition-subtraction chains.

Using the Montgomery form allows us to add two points only if their difference is known. Let R_1 and R_2 be two points in the same cyclic subgroup generated by P. Without loss of generality the points can be denoted $R_1 = \kappa P$ en $R_2 = \lambda P$, their difference is $(\kappa - \lambda)P$ and their sum $(\kappa + \lambda)P$. The computation of a scalar multiple nP of P requires as intermediate values $a_0 P, a_1 P, \ldots, a_l P$ with $a_0 = 1, a_l = n$, and for all $0 < k \leq l$ there should be $0 \leq i, j < k$ such that $a_k = a_i + a_j$ and $a_i - a_j$ occurs somewhere in the chain before a_k. Such a chain is known as a Lucas chain. Although Lucas chains are much less studied than addition chains there is some literature concerning them [20, 5].

There are two types of Lucas chains, those based on a binary algorithm and those based on the extended Euclidean algorithm.

4.2 The Binary Algorithm

Single Exponentiation For ordinary exponentiation the square-and-multiply algorithm (here double-and-add) is very well known. Let $n = \sum_{i=0}^{k-1} n_i 2^i$ be an exponent and let $a = \sum_{i=j}^{k-1} n_i 2^{i-j}$ and $A = aP$ be invariant. Initialization with $j = k, a = 0$, and $A = \mathcal{O}$ poses no problems and if $j = 0$ we also have $a = n$ and hence $A = nP$ as required. Decreasing j by 1 requires replacing a by $2a + n_{j-1}$ to maintain the invariant. For A this constitutes a point doubling and, depending on whether the bit n_{j-1} is set or not, addition by P.

The binary algorithm for addition chains can easily be adapted[6] for Lucas chains if the invariant is strengthened with $b = a + 1$ and $B = bP$. Decreasing j by 1 requires replacing (a, b) by $(2a+1, 2a+2) = (a+b, 2b)$ if the bit n_{j-1} is set and by $(2a, 2+1) = (2a, a+b)$ otherwise. Note that A and B have P as known fixed difference and hence can be added.

Contrary to the binary algorithm for addition chains, in this algorithm the operations performed per step do not depend on the value of the exponent bit. This property can serve as a hedge against timing and power analysis [6, 9]. It also implies that the algorithm always requires one doubling and one fixed difference addition on the curve. In both the old and the new representation this adds up to 6 multiplications in $\mathbf{F}_{2^k}$ per exponent bit.

The binary algorithm produces not only nP, but also $(n+1)P$. Assuming that the y-coordinate of P was also known, it is easy to reconstruct the y-coordinate of nP as well, according to (11).

Double Exponentiation For ordinary addition chains there is a generalization of the binary method to multi-exponentiation due to Straus [28]. For double exponentiation, i.e., the problem of determining $nP + mQ$ given n, m, P, and Q, this method is known as Shamir's trick. Basically, one starts with $A = \mathcal{O}$ and reads both exponents from left to right simultaneously. In a single step first compute $2A$ and depending on the two bits being read, add in either nothing, P, Q, or $P + Q$.

This technique can also be exploited for Lucas-chains, as shown by Schoenmakers [26]. In this case, not only A has to be recorded, but also $A + P$ and $A + Q$ (not $A + P + Q$, since then any efficiency gain is lost). Unless both bits are set, a step will cost one point doubling and two fixed difference additions. If both bits are set a step will cost three fixed difference additions and two ordinary additions. Using the new curve representation, a double exponentiation will on average cost 13 field multiplications per exponent bit. The relatively expensive ordinary additions make this method more expensive than the straightforward method of using two single exponentiations, recovering the y-coordinates and adding the two, costing 12 field multiplications per exponent bit.

Nevertheless, an improvement to the above is possible, due to Akishita [2]. By doing some lookahead, the cost of the most expensive step is reduced to three fixed difference additions. As a result, one step takes on average $2\frac{1}{4}$ fixed difference additions and $\frac{3}{4}$ doublings. A double exponentiation will then cost 10.5 field multiplications per bit exponent for both the old and the new representation.

4.3 Montgomery's Euclidean Algorithm

Double Exponentiation The algorithm below first appeared as a single exponentiation routine in [20] under the name PRAC, the adaptation to double exponentiation was pointed to in [22]. Let R_1 and R_2 be two points in the same

[6] It is unclear who should receive credit for this adaptation, possibly Lehmer [15].

Table 2. Substitution rules for Montgomery's Euclidean Lucas algorithm

No.	Condition	Substitution(d,e)	Costs	Dual costs
M1	$d \le \frac{5}{4}e, d \equiv -e \mod 3$	$((2d-e)/3, (2e-d)/3)$	3α	3α
M2	$d \le \frac{5}{4}e, d \equiv e \mod 6$	$((d-e)/2, e)$	$\alpha+\delta$	$2\alpha+\delta$
M3	$d \le 4e$	$(d-e, e)$	α	α
M4	$d \equiv e \mod 2$	$((d-e)/2, e)$	$\alpha+\delta$	$2\alpha+\delta$
M5	$d \equiv 0 \mod 2$	$(d/2, e)$	$\alpha+\delta$	$\alpha+\delta$
M6	$d \equiv 0 \mod 3$	$(d/3-e, e)$	$3\alpha+\delta$	$4\alpha+\delta$
M7	$d \equiv -e \mod 3$	$((d-2e)/3, e)$	$3\alpha+\delta$	4α
M8	$d \equiv e \mod 3$	$((d-e)/3, e)$	$3\alpha+\delta$	$3\alpha+\delta$
M9	$e \equiv 0 \mod 2$	$(d, e/2)$	$\alpha+\delta$	$\alpha+\delta$

order q subgroup generated by P. There exist unique κ and λ in $\mathbf{Z}_q$ satisfying $R_1 = \kappa P$ and $R_2 = \lambda P$. Given two exponents n and m, one faces the task of computing $nR_1 + mR_2 = (n\kappa + m\lambda)P$. Introduce auxiliary variables a, b, d and e and auxiliary points A, B and C. The variables a and b are kept only for explanatory purposes and should not be implemented (hence κ and λ need not be known). Keep as invariant $A = aP, B = bP$, and $C = A - B$, as well as $0 < d \le e, ad + be = n\kappa + m\lambda$ and $\gcd(d,e) = \gcd(n,m)$. This invariant can be initialized with $a = \kappa, b = \lambda, d = n, e = m$, and accordingly $A = \kappa P$ and $B = \lambda P$. In the main body of the algorithm, the pair (d,e) is decreased step by step, until eventually both d and e equal the greatest common divisor of n and m. At this point $d(A+B) = nR_1 + mR_2$, so a single exponentiation routine should be called to perform the scalar multiplication $d(A+B)$.

Table 2 contains the list of substitutions for (d,e) proposed by Montgomery. One is supposed to perform the first applicable rule from the table. The costs per step are denoted in point additions (α) and point doublings (δ). As an alternative the ternary steps (M1, M2, M6, M7, and M8) can be left out to simplify the algorithm.

Simulation shows that a double exponentiation with two k-bit exponents takes on average slightly less than 1.5 ordinary additions and 0.5 doublings per exponent bit. Using the additive Montgomery version, this would yield 10 field multiplications per exponent bit for a double exponentiation. Using the new multiplicative representation, a double exponentiation costs only 8.5 field multiplications per exponent bit.

As always, single exponentiation can be sped up by precomputing $2^{k/2}P$, effectively transforming a k-bit single exponentiation into a $k/2$-bit double exponentiation. The result is a single exponentiation routine taking, on average, 4.2 field multiplications per exponent bit. A minor detail is that $(2^{k/2}-1)P$ is also required, being the difference between $2^{k/2}P$ and P. Fortunately, the binary algorithm returns both. Variations with other values instead of $2^{k/2}$ are possible.

Single Exponentiation without Precomputation Although the algorithm in [20] actually describes a double exponentiation, it was only used there for sin-

Table 3. Overview of asymptotic costs of scalar multiplication based on Montgomery representation

	Y	General	Old	New
Single exponentiation				
Binary	Yes	$\dot{\alpha} + \delta$	6	6
Montgomery	No	$1.5\alpha + 0.25\delta$	9.5	8
Precomp. Montgomery	No	$0.75\alpha + 0.25\delta$	5	4.3
Double exponentiation				
Schoenmakers	Yes	$\frac{3}{4}(2\dot{\alpha} + \delta) + \frac{1}{4}(3\dot{\alpha} + 2\alpha)$	13.5	13
Akishita	Yes	$\frac{3}{4}(2\dot{\alpha} + \delta) + \frac{3}{4}\dot{\alpha}$	10.5	10.5
Montgomery	No	$1.5\alpha + 0.5\delta$	10	8.5
Twofold exponentiation				
Montgomery	No	$1.5\alpha + 0.5\delta$	10	8.5

gle exponentiations by computing nP as $(n-r)P + rP$. Montgomery proposed setting $(n-r)/r \approx \phi$, where $\phi = \frac{1+\sqrt{5}}{2}$ is the golden ratio. This will result in $\log_\phi \sqrt{n}$ Fibonacci steps (type M3) costing one ordinary addition each, which will be followed by what looks like a random double exponentiation with exponents of magnitude about $\sqrt{n}$ [27]. Putting the pieces together, this results in approximately 8 field multiplications per exponent bit using the new representation and 9.5 field multiplications using the old. This difference is completely irrelevant, since both are outperformed by the binary algorithm (Section 4.2): it is both faster and easier; moreover it allows easy recovery of the y-coordinate and helps to thwart timing and power analysis.

Twofold Exponentiation Montgomery's algorithm can also be used to compute both nP and mP at the same time for relatively low costs by reversing the order. More precisely, go through the entire algorithm, but only keeping track of d and e. At the end, $d = 1$ and $e = 1$. Now work your way back up to $d = n$ and $e = m$ by performing the inverse of each step, but also keeping as invariant $D = dP, E = eP$, and $C = D - E$. Initialization is easy and when $(d, e) = (n, m)$ the pair (D, E) gives the desired powers. The only point of concern is whether the steps can be performed backwards. Inspection of the 9 steps shows this is indeed possible. In Table 2 the costs for performing a step backwards are listed under 'dual costs', referring to the notion of duality for addition chains [14]. Note that the costs for a step and its dual are not always the same.

Tsuruoka [29] gives a recursive version of the dual algorithm, but does not seem to realize that his algorithm is actually Montgomery's dual. Tsuruoka gives a slightly different set of transformation rules (even when taking into account duality), giving a negligible speedup (fine-tuning Montgomery's algorithm is a rather complicated business, as demonstrated by [27, Appendix A]).

Summary In Table 3 an overview is given of the various methods based on Montgomery representation. Once again, α stands for a point addition and δ

for a point doubling. The notation $\dot{\alpha}$ is used for a point addition with a fixed difference, since these are cheaper. Old refers to [17] and [30]'s additive version, new refers to the multiplicative version presented in this paper. The column Y denotes whether easy y-coordinate recovery using (11) is possible or not.

5 Conclusion

Comparing Table 1 with Table 3 shows that the Montgomery representation is considerably slower than traditional methods, both for single and for double exponentiation. However, for single exponentiation using the Montgomery form can still have two advantages. First of all, the uniformity of the steps in the binary algorithm provides a hedge against timing and power analysis. Using ordinary projective in conjunction with the Lucas binary algorithm is substantially slower. Secondly, the Montgomery method requires less memory during a computation.

For double exponentiation timing and power analysis are seldom of any concern, but if they were, the fast double exponentiation routines by Akishita and Montgomery would not provide a hedge. Of course one could run two single exponentiations, recover the y-coordinates and add the result. As for memory requirements, here the Montgomery representation clearly stands out. During computation three points have to be stored, each consisting of two $\mathbf{F}_{2^k}$-elements. Moreover d and e, elements in $\mathbf{Z}_q$, need to be stored. All in all $8k$ bits. On the other hand, Solinas' method precomputes four points of two $\mathbf{F}_{2^k}$-elements each (this includes the two bases). During computation, one point of three $\mathbf{F}_{2^k}$-elements is used. The exponents need to be recoded and stored as well. In total, this costs $13k$ bits. The method based on interleaving two windowed NAFs requires even more memory.

Acknowledgements

The author would like to thank Arjen K. Lenstra for his encouragement, advice, and careful reading of several early drafts, Berry Schoenmakers for fruitful discussions and his kind permission of inclusion of his method, and Peter Beelen and an anonymous referee for useful comments.

References

[1] G. Agnew, R. Mullin, and S. Vanstone. An implementation of elliptic curve cryptosystems over $F_{2^{155}}$. *IEEE J-SAC*, 11(5):804–813, 1993. 241, 245

[2] T. Akishita. Fast simultaneous scalar multiplication on elliptic curve with Montgomery form. *SAC'01*, LNCS **2259**, pages 255–268. 241, 249

[3] D. Bailey and C. Paar. Efficient arithmetic in finite field extensions with application in elliptic curve cryptography. *Journal of Cryptology*, 14(3):153–176, 2001. 240

[4] I. Blake, G. Seroussi, and N. Smart. *Elliptic Curves in Cryptography.* Cambridge University Press, 1999. 243

[5] D. Bleichenbacher. *Efficiency and Security of Cryptosystems based on Number Theory.* PhD thesis, ETH Zürich, 1996. 248

[6] É. Brier and M. Joye. Weierstraß elliptic curves and side-channel attacks. *PKC'02*, LNCS **2274**, pages 335–345. 241, 249

[7] H. Cohen, A. Miyaji, and T. Ono. Efficient elliptic curve exponentiation using mixed coordinates. *Asiacrypt'98*, LNCS **1514**, pages 51–65. 240

[8] D. Hankerson, J. López Hernandez, and A. Menezes. Software implementation of elliptic curve cryptography over binary fields. *CHES'00*, LNCS **1965**, pages 1–24. 242, 243, 244, 245

[9] T. Izu and T. Takagi. A fast parallel elliptic curve multiplication resistant against side channel attacks. *PKC'02*, LNCS **2274**, pages 280–296. 241, 245, 249

[10] D. Johnson and A. Menezes. The elliptic curve digital signature algorithm (ECDSA). CACR Technical report CORR 99-31, University of Waterloo, 1999. 241

[11] B. King. An improved implementation of elliptic curves over $\mathrm{GF}(2^{\mathrm{n}})$ when using projective point arithmetic. *SAC'01*, LNCS **2259**, pages 134–150. 240, 244

[12] D. E. Knuth. *Seminumerical Algorithms*, volume 2 of *The Art of Computer Programming*. Addison Wesley, 3 edition, 1997. 248

[13] D. E. Knuth, editor. *Selected papers on analysis of algorithms*. CSLI Publications, Stanford, 2000. 253

[14] D. E. Knuth and C. H. Papadimitriou. Duality in addition chains. *Bull. Eur. Assoc. Theor. Comput. Sci EATCS*, 13:2–4, 1981. Reprinted in [13, Chapter 31]. 251

[15] D. H. Lehmer. Computer technology applied to the theory of numbers. *Studies in Number Theory*, volume 6 of *MAA Studies in Mathematics*, pages 117–151, 1969. 249

[16] J. López and R. Dahab. Improved arithmetic for elliptic curve arithmetic in $\mathrm{GF}(2^m)$. *SAC'98*, LNCS **1556**, pages 201–212. 244

[17] J. López and R. Dahab. Fast multiplication on elliptic curves over $\mathrm{GF}(2^m)$ without precomputation. *CHES'99*, LNCS **1717**, pages 316–327. 241, 245, 246, 247, 248, 252

[18] A. J. Menezes and S. A. Vanstone. Elliptic curve cryptosystems and their implementation. *Journal of Cryptology*, 6(4):209–224, 1993.

[19] B. Möller. Algorithms for multi-exponentiation. *SAC'01*, LNCS **2259**, pages 165–180. 244, 245

[20] P. L. Montgomery. Evaluating recurrences of form $X_{m+n} = f(X_m, X_n, X_{m-n})$ via Lucas chains. Available from ftp.cwi.nl: /pub/pmontgom/Lucas.ps.gz, 1983. 241, 248, 249, 250

[21] P. L. Montgomery. Speeding the Pollard and elliptic curve methods of factorization. *Mathematics of Computation*, 48(170):243–264, 1987. 240, 241, 242

[22] P. L. Montgomery, Aug. 2000. Private communication: *expon2.txt, Dual elliptic curve exponentiation*. 249

[23] National Institute of Standards and Technology. *Digital Signature Standard*, 2000. FIPS Publication 186-2. 241

[24] K. Okeya, H. Kurumatani, and K. Sakurai. Elliptic curves with the Montgomery-form and their cryptographic applications. *PKC'00*, LNCS **1751**, pages 238–257. 241

[25] K. Okeya and K. Sakurai. Efficient elliptic curve cryptosystems from a scalar multiplication algorithm with recovery of the y-coordinate on a Montgomery-form elliptic curve. *CHES'01*, LNCS **2162**, pages 126–141. 241

[26] B. Schoenmakers, Aug. 2000. Personal communication. 249

[27] M. Stam and A. K. Lenstra. Speeding up XTR. *Asiacrypt'01*, LNCS **2248**, pages 125–143. 241, 251

[28] E. G. Straus. Problems and solutions: (5125) addition chains of vectors. *American Mathematical Monthly*, 71:806–808, 1964. 249

[29] Y. Tsuruoka. Computing short Lucas chains for elliptic curve cryptosystems. *IEICE Trans. Fundamentals*, E84-A(5):1227–1233, 2001. 251

[30] S. A. Vanstone, R. C. Mullin, A. Antipa, and R. Gallant. Accelerated finite field operations on an elliptic curve. WO 99/49386, Patent Cooperation Treaty, 1999. 241, 246, 252