New Parallel Approaches for Scalar Multiplication in Elliptic Curve over Fields of Small Characteristic

Christophe Negre, Jean-Marc Robert

New Parallel Approaches for Scalar Multiplication in Elliptic Curve over Fields of Small Characteristic

Christophe Negre[1,2,3] and Jean-Marc Robert[1,2,3]

[1] Équipe DALI, Université de Perpignan, France
[2] LIRMM, UMR 5506, Université Montpellier 2, France
[3] LIRMM, UMR 5506, CNRS, France

Abstract

We present two new strategies for parallel implementation of scalar multiplication over elliptic curves. We first introduce a Montgomery-halving algorithm which is a variation of the original Montgomery point multiplication. This Montgomery-halving can be run in parallel with the original Montgomery point multiplication in order to concurently compute part of the scalar multiplication. We also present two point thirding formulas in some subfamily of curves $E(\mathbb{F}_{3^m})$. We use these thirding formulas to implement the scalar multiplication through a Third-and-add approach and a parallel Third-and-add and Double-and-add or Triple-and-add approaches. We also provide some implementation results on an Intel Core i7 of the presented two strategies which show a speed-up of 5%-13% compared to non-parallelized approaches.

1 INTRODUCTION

Cryptographic protocols based on elliptic curves necessitate efficient implementation of scalar multiplication on the curve. This operation is generally performed through a sequence of point doublings and point additions combined with a recoding of the scalar which reduces the number of additions. In the case of curves defined over extended binary fields Knudsen [10] and Schroeppel [14] independently proposed an alternative approach in 1999-2000. This approach is based on the halving operation, which multiplies a point of the curve by the inverse of 2. Schroeppel and Knudsen showed that this operation is really efficient on curves defined over extended binary fields. This approach was then used in [16] to parallelize the scalar multiplication, since half of the scalar multiplication can be performed through a Double-and-add approach and the other half can be performed concurrently through a Halve-and-add approach. Some recent software implementations [16] on Intel Core i5 and i7 processors show that this parallelization provides a significant speed up of the scalar multiplication.

In this paper we investigate two new directions for the parallelization of the scalar multiplication. The first direction concerns the parallelization of Montgomery point multiplication on curves $E(\mathbb{F}_{2^m})$. The method of Montgomery for scalar multiplication is very regular : the same set of field operations are performed at each iteration of the main loop. This increases the resistance to timing attack and simple power analysis (SPA). We propose a halving version of the approach of Montgomery, the Montgomery-halving, which replaces the point doublings with point halvings in the main loop of the algorithm. This leads to a parallelization of the Montgomery point multiplication into two threads: one thread performing the original Montgomery point multiplication and a second thread performing the Montgomery-halving approach.

The second direction consists in adapting the halving approach to non-supersingular curve over $\mathbb{F}_{3^m}$. To achieve this goal, we propose a thirding operation on a sub-family of non-supersingular elliptic curves $E(\mathbb{F}_{3^m})$. Thirding a point consists of a multiplication by the inverse of 3 modulo the point order. We use this point thirding in a Third-and-add approach of the scalar multiplication and then combine this approach with the regular Double-and-add or Triple-and-add approaches to obtain a parallelization of the scalar multiplication on $E(\mathbb{F}_{3^m})$.

We provide implementation results based on the proposed parallel approaches. In the case of the parallelized Montgomery point multiplication in $E(\mathbb{F}_{2^m})$, we have implemented the operations in $\mathbb{F}_{2^m}$ using the approaches of [16]. Implementation strategies for field multiplications and field additions in $\mathbb{F}_{3^m}$ are based on [1]. For the other field operations in $\mathbb{F}_{3^m}$ we adapt the methods used for $\mathbb{F}_{2^m}$ to the case of characteristic three fields. The timings obtained provide a speed up between 5% and 10% for the Montgomery-parallel approach and 5% and 13% for the parallel scalar multiplication $E(\mathbb{F}_{3^m})$ compared to non-parallel approach.

The paper is organized as follows. In Section 2 we review some background on elliptic curve over extended binary field and related scalar multiplication methods. In Section 3 we present the Montgomery-halving approach and the parallelized Montgomery approach for scalar multiplication and related implementation results. In Section 4 we review the best known methods for the implementation of the scalar multiplication in $E(\mathbb{F}_{3^m})$. Then, in Section 5, we present our proposed thirding point formula and Third-and-add and parallel approaches for scalar multiplication, along with implementation strategies and timings. Finally, in Section 6, we give some concluding remarks.

2 Binary Elliptic Curve Scalar Multiplication

In this section, we consider an extended binary field $\mathbb{F}_{2^m} = \mathbb{F}_2[t]/(f(t))$, where $f(t) \in \mathbb{F}_2[t]$ is an irreducible polynomial of degree m, and an elliptic curve $E(\mathbb{F}_{2^m})$ defined by the following Weierstrass equation:

$$E : y^2 + xy = x^3 + ax^2 + b \text{ with } a, b \in \mathbb{F}_{2^m}. \quad (1)$$

Let $P_1 = (x_1, y_1)$ and $P_2 = (x_2, y_2)$ be two points on $E(\mathbb{F}_{2^m})$. Then the addition of two points on the curve can be computed as follows: if (x_3, y_3) are the coordinates of the point $P_3 = P_1 + P_2$, we have:

$$\begin{cases} x_3 &= \lambda^2 + \lambda + x_1 + x_2 + a, \\ y_3 &= (x_1 + x_3)\lambda + x_3 + y_1, \end{cases} \text{ where } \lambda = \begin{cases} \frac{y_1+y_2}{x_1+x_2} & \text{if } P_1 \neq P_2, \\ \frac{y_1}{x_1} + x_1 & \text{if } P_1 = P_2. \end{cases} \quad (2)$$

The above point doubling and point addition formulas involve an inversion in $\mathbb{F}_{2^m}$ which is a costly operation. It is generally preferable to use a projective coordinate system which provides curve operation formulas with a few more field multiplications but without any field inversion. The most used coordinate systems are: the standard projective coordinate system where the projective coordinates $P = (X : Y : Z)$ correspond to the affine coordinates $(X/Z, Y/Z)$ satisfying the curve equation (1), and the López-Dahab projective coordinates $P = (X : Y : Z)$ which corresponds to the affine coordinates $(X/Z, Y/Z^2)$.

Double-and-add scalar multiplication.

The scalar multiplication consists of the computation

$$k \cdot P = \overbrace{P + P + \cdots + P}^{k \; times}$$

for a point $P \in E(\mathbb{F}_{2^m})$ and a scalar $k \in \mathbb{N}$. The most used method to implement scalar multiplication is the classical *Double-and-add* approach combined with the NAF_w recoding of the scalar k. This recoding consists in rewriting k as $k = \sum_{i=0}^{\ell} k_i 2^i$ such that each k_i is an odd integer in $\{\pm 1, \ldots, \pm 2^{w-1} - 1\}$. These coefficients k_i are generated through the following loop:

while $k > 0$ **do**
  if $k \equiv 0 \mod 2$ **then**
    $k_i \leftarrow 0, k \leftarrow k/2$
  else
    $k_i \leftarrow k \bmod_s 2^w, \; k \leftarrow (k - k_i)/2$
  end if
  $i \leftarrow (i + 1)$
end while

The term $\bmod_s$ above represents a signed modular reduction. The NAF_w representation of k is sparse since the proportion of non-zero coefficients k_i is $\cong 1/(w+1)$. This is advantageous since this reduces the number of additions in the Double-and-add algorithm. Indeed, the Double-and-add approach computes $k \cdot P$ by first precomputing $T[i] = i \cdot P$ for odd positive integers $i \in [0, 2^{w-1}]$, and then performs a sequence of doubling and addition $R \leftarrow 2 \cdot R + sign(k_i)T[\,|k_i|\,]$ for $i = \ell, \ell - 1, \ldots, 0$ where ℓ is the length of $NAF_w(k)$. As stated in [7], the total complexity of the Double-and-add scalar multiplication is $w + \ell$ doublings and $\frac{\ell}{w+1} + 2^{w-1}$ additions.

Algorithm 1 Double-and-add

Require: $P \in E(\mathbb{F}_{2^m})$ and a scalar $k \in [0, N-1]$ where N is the order of P.

Ensure: $Q = k \cdot P$

1: Compute $NAF_w(k) = \sum_{i=0}^{\ell} k_i 2^i$

2: Compute $T[i] = i \cdot P$ for all odd positive integers $i \in [0, 2^{w-1}]$

3: $Q \leftarrow \mathcal{O}$

4: **for** i **from** ℓ **downto** 0 **do**

5: $\quad Q \leftarrow 2 \cdot Q + sign(k_i) T[\,|k_i|\,]$

6: **end for**

7: **return** (Q)

Montgomery point multiplication.

An another popular approach to implement scalar multiplication is the Montgomery point multiplication. This algorithm uses two points Q_0 and Q_1 which have a constant difference $Q_1 - Q_0 = P$ during the whole run of the algorithm. The point Q_0 successively takes the values $k^{(i)} \cdot P$ where

$$k^{(i)} = [k_\ell, k_{\ell-1}, \ldots, k_i]_2 = \left(\sum_{j=i}^{\ell} k_j 2^{j-i} \right) \text{ with } i = \ell, \ell-1, \ldots, 1, 0,$$

and $[k_\ell, k_{\ell-1}, \ldots, k_1, k_0]_2$ is the binary representation of the scalar k. This approach is described in Algorithm 2. We can notice that the loop operations on the point Q_0 are $Q_0 \leftarrow Q_0 + Q_1 = 2Q_0 + P$ if $k_i = 1$ or $Q_0 \leftarrow 2Q_0$ if $k_i = 0$ which are equivalent to the point doubling and point addition of the regular Double-and-add approach. The other loop operations $Q_1 \leftarrow 2Q_1$ if $k_i = 1$ and $Q_1 \leftarrow Q_0 + Q_1$ if $k_i = 0$ maintain the difference $Q_1 - Q_0 = P$ during the whole computations. An interesting property of this approach is its regularity during each iteration. Such property is important to provide some resistance against some side channel attacks like simple power analysis or timing attacks. In the case of binary elliptic curves Lopez and Dahab in [11] showed that the use of *standard projective coordinates* makes this approach almost as efficient as Double-and-add approach combined with NAF_w recoding. This optimized approach requires $(6\ell + 10)M + I$ where ℓ is the bit length of k and M and I represent a field multiplication and a field inversion, respectively.

Halve-and-add scalar multiplication.

In the case of elliptic curve over extended binary field, the halving operation, originally presented in [10], [14], makes possible the implementation of the scalar multiplication through a Halve-and-add approach in place of a Double-and-add approach. We assume that the degree m of $\mathbb{F}_{2^m}$ is odd and that $Trace(a) = 1$ where a is the coefficient of the equation (1) which defines $E(\mathbb{F}_{2^m})$ and where $Trace(a)$ is defined as $Trace(a) = \sum_{i=0}^{m-1} a^{2^i}$.

The point doubling over $E(\mathbb{F}_{2^m})$ is a one to one application when it is restricted to the subgroup of

Algorithm 2 Montgomery point multiplication

Require: $P \in E(\mathbb{F}_{2^m})$ and a scalar $k \in [0, ord(P)]$

Ensure: $Q = k \cdot P$

1: $Q_0 \leftarrow \mathcal{O}$
2: $Q_1 \leftarrow P$
3: **for** i from $t-2$ downto 0 **do**
4: **if** $(k_i = 0)$ **then**
5: $T \leftarrow Q_0, Q_0 \leftarrow 2T, Q_1 \leftarrow T + Q_1$
6: **else**
7: $T \leftarrow Q_1, Q_1 \leftarrow 2T, Q_0 \leftarrow Q_0 + T,$
8: **end if**
9: **end for**
10: **return** Q_0

$E(\mathbb{F}_{2^m})$ formed by the points of odd order. We consider only such points of odd order. The halving formula is derived from the doubling formula : let $P = (x, y)$ and $Q = (u, v)$ be two points on $E(\mathbb{F}_{2^m})$ such that $Q = 2 \cdot P$. If N is the order of Q and s is such that $2 \times s = 1 \mod N$, we have $P = s \cdot Q$ which is equivalent to $P = \left[\frac{1}{2}\right] \cdot Q$. The doubling formula (2) provides the following relations between (x, y) and (u, v):

$$\lambda = x + y/x \tag{3}$$

$$u = \lambda^2 + \lambda + a \tag{4}$$

$$v = x^2 + u(\lambda + 1) \tag{5}$$

We first notice that (4) implies that $Trace(u + a) = 0$ since $Trace(\lambda^2 + \lambda) = 0$ for any λ. We can also show that $Trace(x + a) = 0$ since P is also of odd order and can be written as $P = 2 \cdot R$ for some point $R \in E(\mathbb{F}_{2^m})$. Based on this fact, the authors of [10], [14] have then derived the following method to compute the halving of Q.

1) Compute the solution $\lambda_0 = HalfTrace(u + a)$ of the equation $u = \lambda^2 + \lambda + a$ where

$$HalfTrace(u + a) = \sum_{i=0}^{\frac{m-1}{2}} (u + a)^{2^{2i}}.$$

This λ_0 is a valid solution:

$$\begin{aligned} \lambda_0^2 + \lambda_0 &= \left(\sum_{i=0}^{\frac{m-1}{2}} (u + a)^{2^{2i}}\right)^2 + \left(\sum_{i=0}^{\frac{m-1}{2}} (u + a)^{2^{2i}}\right) \\ &= \left(\sum_{i=0}^{m-1} (u + a)^{2^i}\right) + (u + a)^{2^m} \\ &= Trace(u + a) + u + a \\ &= u + a. \end{aligned}$$

2) Compute the second solution $\lambda_1 = \lambda_0 + 1$ of the equation $u = \lambda^2 + \lambda + a$.
3) Using (5), we obtain the two possible values x_0 and x_1 of the x-coordinate of P, each corresponding to λ_0 or λ_1:

$$\begin{aligned} x_0 &= \sqrt{v + u(\lambda_0 + 1)} = \sqrt{v + u\lambda_0 + u}, \\ x_1 &= \sqrt{v + u(\lambda_1 + 1)} = \sqrt{v + u\lambda_0}. \end{aligned}$$

4) The x-coordinate of P is then $x = x_0$ if $Trace(x_0) = Trace(a)$ otherwise $x = x_1$ since P must have an x-coordinate satisfying $Trace(x) = Trace(a)$. The fact that only one element among $\{x_0, x_1\}$ satisfies $Trace(x_i) = Trace(a)$ is proven in [5].
5) From (3), the y-coordinate is then derived from x and λ as $y = x(x + \lambda)$.

The point halving is then reduced to the following computations: one $HalfTrace$ and one $Trace$ plus two multiplications, one square root and few additions. Consequently, when the $HalfTrace$, square root and $Trace$ operations are performed efficiently, this makes the halving operation competitive compared to doubling operation.

Algorithm 3 Halve-and-add

Require: $P \in E(\mathbb{F}_{2^m})$ of odd order N and a scalar $k \in [0, N-1]$ and $\ell = \lceil \log_2(N) \rceil + 1$ and w a window size.

Ensure: $Q = k \cdot P$.

1: $k' \leftarrow 2^\ell k \mod N$
2: Compute $NAF_w(k') = \sum_{i=0}^{\ell} k'_i 2^i$
3: **for** $j \in J = \{1, 3, \ldots, 2^{w-1} - 1\}$ **do** $Q_j \leftarrow \mathcal{O}$
4: $Q \leftarrow P$
5: **for** i from ℓ downto 0 **do**
6: $\quad Q_{|k'_i|} \leftarrow Q_{|k'_i|} + sign(k'_i)Q$
7: $\quad Q \leftarrow Q/2$
8: **end for**
9: **return** $Q \leftarrow \sum_{j \in J} jQ_j$

The Halve-and-add algorithm proposed in [5] is similar to the Double-and-add approach. Preliminary, we need to recode the scalar k in base $1/2$ representation: we first compute

$$k' = 2^\ell \times k \mod N \tag{6}$$

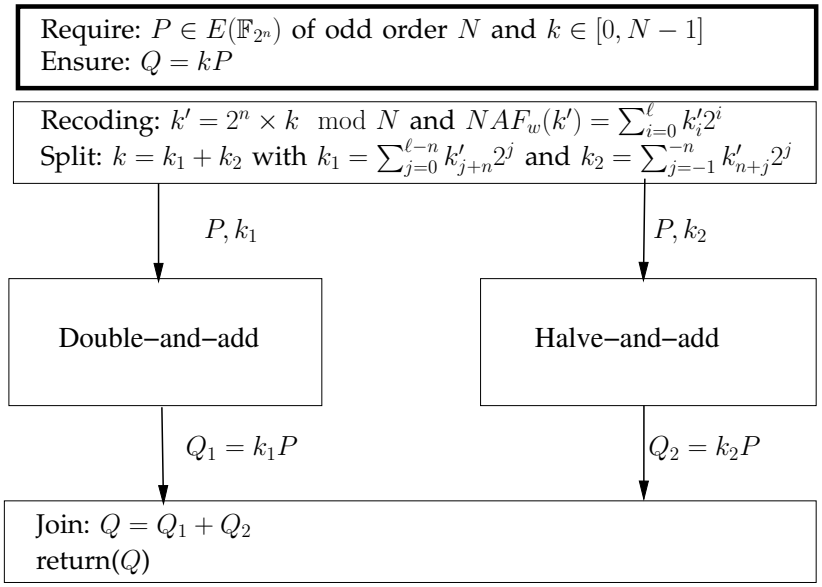
where ℓ is the bit length of N and then we derive $NAF_w(k') = \sum_{i=0}^{\ell} k'_i 2^i$. Thus, using (6), we obtain the following base $1/2$ representation of k:

$$k = k' \times 2^{-\ell} = \sum_{i=0}^{\ell} k'_i 2^{i-\ell}$$

The scalar multiplication can then be computed through a sequence of halvings and additions. This sequence of operations is slightly different from the Double-and-add approach since the approach is left-to-right instead of right-to-left. In a point Q we store sequentially the different values $\left[\frac{1}{2^i}\right] \cdot P, i = 0, \ldots, \ell$ which makes possible to keep Q in affine coordinate during the whole algorithm : no conversion are then required. The values $\left[\frac{1}{2^i}\right] \cdot P$ are added to Q_j where $j \in J = \{1, 3, \ldots, 2^{w-1} - 1\}$ if $k'_i = j$. The final result is obtained by adding the points Q_j multiplied by their respective coefficient j. This approach is detailed in Algorithm 3. This algorithm require $\ell/(w+1) + 2^{w-1} + 1$ Additions and ℓ halvings.

Parallel Double-and-add/Halve-and-add scalar multiplication.

Figure 1. Parallel Double-and-add and Halve-and-add



We review the parallel approach for scalar multiplication in $E(\mathbb{F}_{2^m})$ presented in [16]. We consider a point $P \in E(\mathbb{F}_{2^m})$ of odd order N of bit length ℓ and a scalar $k \in [0, N-1]$. The scalar k is recoded by first computing

$$k' = 2^n \times k \mod N$$

where n is the fixed split generally taken close to $\ell/2$. Afterwards we compute $NAF_w(k') = \sum_{i=0}^{\ell} k'_i 2^i$ and we obtain the following expression of k:

$$k = k' \times 2^{-n} \mod N = (k'_\ell 2^{\ell-n} + \ldots + k'_n) + (k'_{n-1} 2^{-1} + \ldots + k'_0 2^{-n}) \mod N.$$

The scalar multiplication $k \cdot P$ of P is then split in two parts:

$$k \cdot P = \underbrace{(k'_\ell 2^{\ell-n} + \ldots + k'_n)}_{k_1} \cdot P + \underbrace{(k'_{n-1} 2^{-1} + \ldots + k'_0 2^{-n})}_{k_2} \cdot P,$$

the first part $k_1 \cdot P$ is computed with the Double-and-add algorithm and the second part $k_2 \cdot P$ is performed through a Halve-and-add approach. This parallel method is described in Fig. 1.

3 PARALLELIZED MONTGOMERY POINT MULTIPLICATION

In the previous section we have seen that the scalar multiplication can be performed in parallel fashion by concurrently perform the Halve-and-add and Double-and-add algorithms. We present in this section an adaptation of this strategy in order to parallelize the Montgomery point multiplication.

3.1 Montgomery-halving

We consider an elliptic curve $E(\mathbb{F}_{2^m})$ over $\mathbb{F}_{2^m}$, a point $P \in E(\mathbb{F}_{2^m})$ with an odd order N and a scalar $k \in [0, N-1]$. We further assume that $Trace(a) = 1$ and m is odd which makes possible the use of a point halving on $E(\mathbb{F}_{2^m})$. Our goal is to modify the Montgomery point multiplication in order to use the halving operation in place of the doubling operation. We present a new algorithm with the following properties:

- regularity of the operations performed in each iteration of the main loop in order to keep the SPA resistance properties of the original Montgomery point multiplication;
- constant difference between Q_1 and Q_0, the two points computed at each iteration all along the whole point multiplication, which provides some resistance against fault attacks.

As in the case of the Halve-and-add approach we have to recode the scalar k by first computing $k' = 2^{\ell-1} \cdot k \mod N = \sum_{i=0}^{\ell-1} k'_i 2^i$ which gives $k = \sum_{i=0}^{\ell-1} k'_i 2^{i-(\ell-1)}$ where $k'_i \in \{0, 1\}$ and ℓ is the bit length of N. The proposed approach involves two points Q_0 and Q_1 such that:

- the point Q_0 takes successively the values $k^{(i)} \cdot P$ for $i = 0, 1, \ldots, \ell-1, \ell$ where $k^{(i)} = (\sum_{j=0}^{i} k'_j 2^{j-i})$;
- the point Q_1 satisfies $Q_1 = Q_0 - 2P$ during the whole run of the algorithm.

The proposed Montgomery-halving point multiplication is described in Algorithm 4.

Lemma 1. *Let $Q_{0,i}$ and $Q_{1,i}$ be the respective values of Q_0 and Q_1 after the i-th loop iteration and let $k^{(i)} = \sum_{j=0}^{i} k'_j 2^{j-i}$. Then the following identities hold:*

$$\begin{cases} Q_{0,i} &= k^{(i)} \cdot P, \\ Q_{1,i} &= (k^{(i)} - 2) \cdot P. \end{cases}$$

Proof: We prove the lemma by induction on i. We first consider the case $i = 0$: since the initial value of Q_0 and Q_1 are $Q_0 = k_0 \cdot P$ and $Q_1 = (k_0 - 2) \cdot P$, the induction hypothesis is satisfied in this case.

Algorithm 4 Montgomery-halving

Require: $k \cdot P \in E(\mathbb{F}_{2^m})$ of odd order N and a scalar $k \in [0, N-1]$ and let $\ell = \lfloor \log_2(N) \rfloor$.

Ensure: $Q = k \cdot P$.

1: Compute $k' = 2^{\ell-1} \cdot k \mod N = \sum_{i=0}^{\ell-1} k'_i 2^i$ with $\ell = \lfloor \log_2(N) \rfloor + 1$
2: $Q_0 \leftarrow k_0 \cdot P, Q_1 \leftarrow k_0 \cdot P - 2 \cdot P$
3: **for** i from 1 to $\ell - 1$ **do**
4: **if** $(k'_i = 1)$ **then**
5: $T \leftarrow Q_0/2, Q_0 \leftarrow T, Q_1 \leftarrow Q_1 - T$
6: **else**
7: $T \leftarrow Q_1/2, Q_1 \leftarrow T, Q_0 \leftarrow Q_0 - T$
8: **end if**
9: **end for**
10: **return** (Q_0)

We now assume that the induction hypothesis is satisfied up to i and we prove it for $i+1$. By induction hypothesis we have

$$\begin{cases} Q_{0,i} &= k^{(i)} \cdot P, \\ Q_{1,i} &= k^{(i)} \cdot P - 2 \cdot P, \end{cases} \quad \text{and} \quad Q_{1,i} - Q_{0,i} = -2 \cdot P.$$

Now we consider the two following cases:

- If $k'_{i+1} = 0$, which means that $k^{(i+1)} = k^{(i)}/2$. In this case the following operations are performed:

$$\begin{cases} Q_{0,(i+1)} &= Q_{0,i}/2 = (k^{(i+1)}/2) \cdot P, \\ Q_{1,(i+1)} &= Q_{1,i} - Q_{0,i}/2 = (Q_{1,i} - Q_{0,i}) + Q_{0,i}/2. \end{cases}$$

  By induction hypothesis $Q_{1,i} - Q_{0,i} = -2P$ and $Q_{0,i} = k^{(i)}P$ which implies $Q_{0,i}/2 = (k^{(i)}/2) \cdot P = k^{(i+1)}P$. We thus obtain the required values

$$\begin{cases} Q_{0,(i+1)} &= k^{(i+1)} \cdot P, \\ Q_{1,(i+1)} &= -2 \cdot P + k^{(i+1)}P. \end{cases}$$

- If $k'_{i+1} = 1$ we have $k^{(i+1)} = k^{(i)}/2 + 1$ and the two points $Q_{0,i+1}$ and $Q_{1,i+1}$ are computed as follows:

$$\begin{cases} Q_{0,(i+1)} &= Q_{0,i} - Q_{1,i}/2 = Q_{0,i}/2 - (Q_{1,i} - Q_{0,i})/2, \\ Q_{1,(i+1)} &= Q_{1,i}/2 \end{cases}$$

  We use again the induction hypothesis which gives $Q_{1,i} - Q_{0,i} = -2P, Q_{0,i} = k^{(i)}P$ and $Q_{1,i} = (k^{(i)} - 2)P$ and we obtain

$$\begin{cases} Q_{0,(i+1)} &= (k^{(i)}/2)P + P = k^{(i+1)}P, \\ Q_{1,(i+1)} &= \frac{(k^{(i)}-2)}{2}P = (k^{(i)}/2 + 1)P - 2P = k^{(i+1)}P - 2P. \end{cases}$$

This ends the proof of the lemma. □

The proposed algorithm still have a regularity in the operations of the main loop iteration, i.e., the same type of operation is performed independently of the value taken by the bits k'_i. The difference $Q_{1,i} - Q_{0,i} = -2 \cdot P$ is preserved during the whole computations, this can be used to detect some fault injection during the computation. Consequently, the proposed algorithm conserves the main properties of the original Montgomery point multiplication.

Unfortunately, in terms of efficiency, the use of point halving requires to use affine coordinates. Indeed, we could not find a projective form of the Montgomery-halving approach which would save the inversions involved in the ℓ point additions of the algorithm. But the approach still makes possible to develop a parallel version of the Montgomery point multiplication. The next section deals with this version.

3.2 Parallel version

In this parallelized version, we use a split technique similar to the one used by the authors in [16]. Let P be the point to be multiplied with a scalar k and we assume that P has an odd order N. The method to parallelize the computations is similar to the one reviewed in Section 2. The scalar k is recoded by first computing

$$k' = 2^n \times k \mod N = \sum_{i=0}^{\ell} k'_i 2^i$$

and is then split in two parts $k = k_1 + k_2$

$$k = k' \times 2^{-n} \mod N = \underbrace{(k'_t 2^{\ell-n} + \ldots + k'_n)}_{k_1} + \underbrace{(k'_{n-1} 2^{-1} + \ldots + k'_0 2^{-n})}_{k_2} \mod N.$$

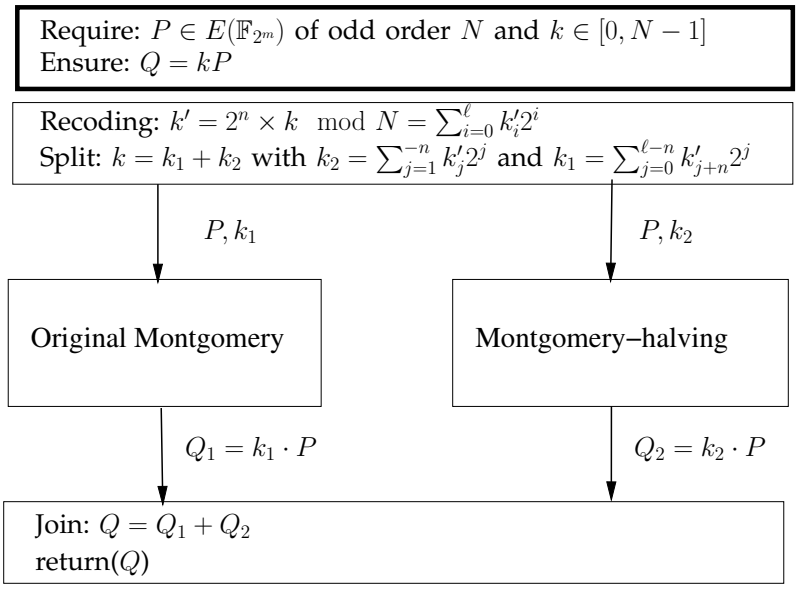
Then the scalar multiplication $k_1 \cdot P$ is computed with the original Montgomery algorithm and the second part $k_2 \cdot P$ is performed in parallel through a Montgomery-halving approach. The final result is obtained with a final addition $k \cdot P = (k_1 \cdot P) + (k_2 \cdot P)$. This parallel method is described in Fig. 2.

3.3 Implementations results

The platform used for the implementations is an Optiplex 990 DELL equipped with an Intel Core i7-2600 and with an Ubuntu 12.04 operating system. The code was written in C language and compiled with gcc 4.6.3. Following the recommendations of [2] the Hyperthreading and Turbo-boost options have been disabled on our platform in order to measure accurately the performances of the considered algorithms.

The considered curves are the NIST B233 defined over $\mathbb{F}_{2^{233}} = \mathbb{F}_2[t]/(t^{233} + t^{87} + 1)$ and the NIST B409 curve defined over $\mathbb{F}_{2^{409}} = \mathbb{F}_2[t]/(t^{409} + t^{79} + 1)$. Both curves are given by a curve equation of the form $y^2 = x^3 + x^2 + b$ where b is a non-sparse element in the field $\mathbb{F}_{2^m}$. The conditions $Trace(a) = 1$ and m odd are fulfilled in order to have an efficient halving operation on the curve.

Figure 2. Parallelized Montgomery



3.3.1 Implementation strategies for field operations

We used the following strategies for implementing field operations in $\mathbb{F}_{2^{233}}$ and $\mathbb{F}_{2^{409}}$:

- *Polynomial multiplication:* we follow the same strategy as the one presented in [16]. The multiplication is performed using a combination of Karatsuba approach for binary polynomial and carry-less instruction of Intel Core i7 processor. Specifically, the recursion of Karatsuba is performed twice for $\mathbb{F}_{2^{233}}$ resulting in nine multiplications of degree 63 polynomials. For the case of $\mathbb{F}_{2^{409}}$, the recursion is performed three times which results in 25 multiplications of degree 63 polynomials. A multiplication of polynomial of degree 63 is performed with the Intel core i7 carry-less multiplication (`PCLMUL` instruction). This instruction is part of the MMX instruction set and we use it with the intrinsic function `_mm_clmulepi64_si128`.
- *Squaring:* Let $a(t) = \sum_{i=0}^{m-1} a_i t^i$ be a binary polynomial, the square of a is $a(t)^2 = \sum_{i=0}^{m-1} a_i t^{2i}$. Consequently, squaring a binary polynomial consists in inserting zeros between each bit of a. This insertion of zeros is performed with the method presented in [3]. Specifically, there exists an MMX instruction, the `byte_shuffle`, which applies a fixed S-box $\{0,1\}^4 \to \{0,1\}^8$ simultaneously to the least significant nibbles of each byte of a 128 bit word. This makes possible to perform the squaring of a polynomial $a(t)$ by choosing S as the function which inserts zeros between each bit of a nibble. The resulting squaring implementation consists in a few word maskings and byte shufflings.
- *Reduction:* the irreducible polynomials which defines the NIST fields $\mathbb{F}_{2^{233}}$ and $\mathbb{F}_{2^{409}}$ are trinomials. We then use the usual approach to perform the polynomial reduction modulo $f(t)$ which consists

of a few word shifts of the unreduced part of the polynomial followed with XOR operations.

- *Square-root:* let $a(t) = \sum_{i=0}^{m-1} a_i t^i$ be an element of $\mathbb{F}_{2^m}$, then the square-root of a can be expressed as follows:

$$\sqrt{a} = \left(\sum_{i=0}^{\frac{m-1}{2}} a_{2i} t^i \right) + \sqrt{t} \left(\sum_{i=0}^{\frac{m-1}{2}} a_{2i+1} t^i \right).$$

  We thus have to separate the bits of a into two parts: one part containing bits with odd subscript and the second part with bits with even subscript. This can be done by a few maskings and shiftings. Then we have to remove the zeros between the bits of the resulting two parts. We use the `byte_shuffle` instruction to remove these zeros as it was used in the squaring implementation. Then we have to multiply with $\sqrt{t}$, but $\sqrt{t}$ has sparse expression in the fields $\mathbb{F}_{2^{233}}$ and $\mathbb{F}_{2^{409}}$, consequently this multiplication consists of a few shifts and XORs.

- *Field inversion:* We use the approach of Itoh-Tsujii [8] which is derived from the following expression of the inverse of a

$$a^{-1} = a^{2^m - 2} = \left(a^{2^{m-1}-1} \right)^2.$$

  The method of Itoh-Tsujii performs this exponentiation through a short sequence of field multiplications and multi-squarings. Indeed, in the case of $\mathbb{F}_{2^{233}}$ a sequence of a^{2^e-1} is computed where e follows the addition chain $1 \to 2 \to 3 \to 6 \to 7 \to 14 \to 28 \to 29 \to 58 \to 116 \to 232$ using the following relation:

$$\left(a^{2^e-1} \right)^{2^{e'}} \times a^{2^{e'}-1} = a^{2^{e+e}-1}.$$

  The same method is applied for $\mathbb{F}_{2^{409}}$.

- *Half-trace:* This operation is defined as $HalfTrace(a) = \sum_{i=0}^{(m-1)/2} a^{2^{2i}}$. One important fact is that it is a linear operation. Consequently we precompute and store in a table $T[\cdot][\cdot]$ the terms

$$T[i][c_0, c_1, c_2, c_3] = HalfTrace((c_0 + c_1 t + c_2 t^2 + c_3 t^3) t^{4i})$$

  for $i = 0, \ldots, m/4$ and $[c_0, c_1, c_2, c_3] \in \{0, 1\}^4$. Then the polynomial a is split in a sequence of nibbles; we use the table $T[\cdot][\cdot]$ to compute the half-trace of each nibble and then accumulate their value to obtain $HalfTrace(a)$. As shown in [5], this approach can be optimized by removing the even bits of a: this divides by two the number of $HalfTrace$ computation on the nibbles of a. We have implemented this optimized version.

3.3.2 Implementation results

In Table 1 we report the timings obtained for the three considered approaches: original Montgomery, Montgomery-halving and Montgomery-parallel. These timings are average of hundreds of run with input point P and scalar k taken at random. The values given in the *split* column corresponds the split value n defined in Subsection 3.2 which minimizes the timing of the parallel approach. The optimal split value is

39 for the curve B233 and 59 for the curve B409. Fig. 3 shows the behavior of the timings of the parallel algorithm when the split value n varies from 20 to 60. We notice that the curve has a 'V' shape: this means that the timings depends linearly to $\max(n, 233 - n)$.

We also notice that our proposed parallelized version of Montgomery method provides a speed-up of 5%-10% compared to the non-parallelized original Montgomery approach.

Table 1
Timings in clock cycles of Montgomery approaches.

Algorithm	NIST Curve B233			NIST Curve B409		
	#CC	ms	split	#CC	ms	split
Regular Montgomery	157307	0.04	-	734256	0.21	-
Montgomery-Halving	707385	0.20	-	4212043	1.23	-
Montgomery-Parallel	149117	0.04	39	659460	0.19	59
Relative speed-up Montgomery-Parallel/Montgomery-Doubling	5.09 %			10.5 %		

Below we provide some recently published implementation results for several Montgomery approaches for field sizes close to 256.

Taverne *et al.* [16] on Curve2251 : 225 000,
Bernstein [2] on Curve25519 : 194 000,
Hamburg [6] Montgomery over $\mathbb{F}_p$: 153 000.

The implementation results are all on an Intel Core i7-2600 with 3.4 GHz. This shows that our implementation is competitive compared to these results even if the comparison is not fair since most of the considered field have field size slightly larger than 233, but in counterparts they generally take advantage of sparse curve coefficients.

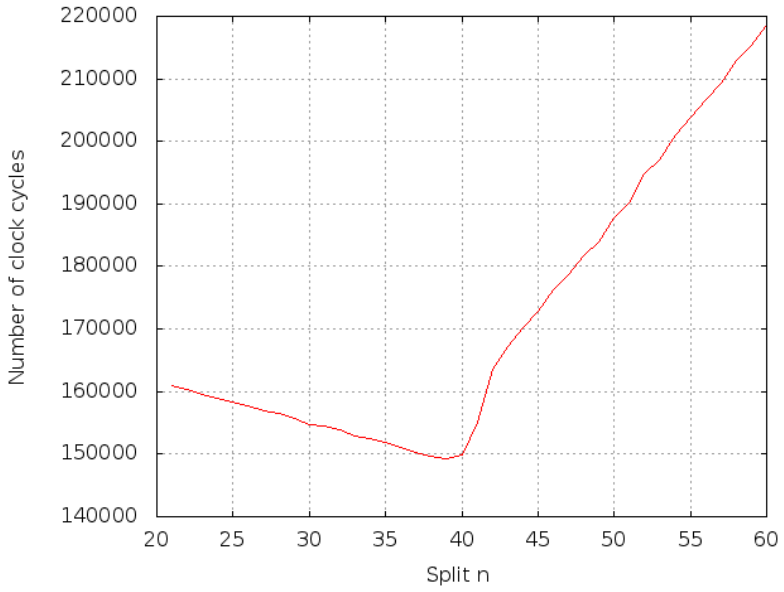
4 Elliptic curves defined over characteristic three fields

In this section, we focus on elliptic curves defined over characteristic three fields $\mathbb{F}_{3^m}$. Such elliptic curves are generally separated into two kinds of curve : supersingular elliptic curves and ordinary elliptic curves. Here we will focus on ordinary elliptic curves. Such curves can be defined by a Weierstrass equation:

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

where $a, b \in \mathbb{F}_{3^m}$ and $a, b \neq 0$ since the curve equation must be non-singular.

Figure 3. Timings of the Montgomery-parallel approach in terms of the split n



4.1 Curve operations

Let $P_1 = (x_1, y_1)$ and $P_2 = (x_2, y_2)$ be two points on $E(\mathbb{F}_{3^m})$. The addition, doubling and tripling formulas on this curve are as follows:

- *Addition.* The coordinates (x_3, y_3) of the point $P_3 = P_1 + P_2$ are:

$$\begin{cases} x_3 &= \lambda^2 - a - x_1 - x_2, \\ y_3 &= \lambda(x_1 - x_3) - y_1, \end{cases} \quad \text{where } \lambda = \frac{y_2 - y_1}{x_2 - x_1}.$$

- *Doubling.* The coordinates (x_3, y_3) of $P_3 = 2 \cdot P_1$ are:

$$\begin{cases} x_3 &= \lambda^2 - a + x_1, \\ y_3 &= \lambda(x_1 - x_3) - y_1. \end{cases} \quad \text{where } \lambda = \frac{ax_1}{y_1}.$$

- *Tripling.* The coordinates (x_3, y_3) of $P_3 = 3 \cdot P_1$ are computed as follows

$$\begin{cases} x_3 &= \frac{y_1^6}{a^2(x_1^3+b)^2} - \frac{ax_1^3}{x_1^3+b}, \\ y_3 &= \frac{y_1^9}{a^3(x_1^3+b)^3} - \frac{y_1^3}{x_1^3+b}. \end{cases} \tag{7}$$

A common strategy to improve the efficiency of the addition, doubling and tripling operations consists in using a projective coordinate system in order to remove field the inversions which are costly operations. The following set of projective coordinates were proposed in the context of curve defined over field of characteristic three:

- The Jacobian projective coordinates of a point $P = (X, Y, Z)$ (cf. [7]) corresponds to the affine coordinates of P as $(x, y) = (X/Z^2, Y/Z^3)$.

- The ML-projective coordinates (X, Y, Z, T) of a point P defined in [9] corresponds to the affine coordinates of P as $(x, y) = (X/T, Y/Z^3)$.
- The scaled projective coordinate system (X, Y, T) was proposed in [4]. The affine coordinates (x, y) of a point P can be derived from its scaled projective coordinates (X, Y, T) as $x = X/bT$ and $y = Y/bT$.

The complexities of the curve operations in each coordinate system are given in Table 2. Farashahi *et al.* showed in [4] that the most advantageous coordinate system is the scaled projective system for curves admitting a point of order three. Smart and Westwood proved in [15] that these curves are isomorphic to curves given by a Weierstrass equation of the form

$$y^2 = x^3 + x^2 + b \text{ with } b \in \mathbb{F}_{3^m} \text{ and } b \neq 0.$$

In other words, an elliptic curve which admits a point of order 3 has a curve equation with $a = 1$. We also notice from Table 2 that for the other king of ordinary curves the Jacobian coordinate system provides the best set of curve operations.

Table 2

Complexity of curve operations in $E(\mathbb{F}_{3^m})$

Coordinate system	Eq. form	Mixed addition	Doubling	Tripling
Jacobian	any a	7M+3S+2C+1D	5M+2S+3C	3M+2S+5C +1D
ML-Coordinate system	$a = 1$	8M+2C	5M+3S+3C	6M+6C
Scaled projective [4]	$a = 1$	8M+1D+1C	3M+2C	4M+4C+1D

M=Multiplication, S=Squaring, C=Cubing, D=Multiplication with a constant.

4.2 Scalar multiplication in $E(\mathbb{F}_{3^m})$

Now we review the best known approaches for scalar multiplication for elliptic curves over characteristic three field. The well known Double-and-add approach has already been presented in Section 2. This approach applies also in the case of a curve $E(\mathbb{F}_{3^m})$. But for elliptic curves over characteristic three fields, the tripling operation on the curve is really efficient (cf. Table 2). This motivates the use of the Triple-and-add variation of the Double-and-add approach which replaces doubling operations with tripling operations.

Triple-and-add. The Triple-and-add approach uses a scalar recoded with the signed window representation ($SWR_{3,w}$) of [13]. The $SWR_{3,w}$ extends the NAF_w to base three integer representation. Specifically, the scalar k is recoded as a sequence $k_0, \ldots, k_\ell$ of elements $k_i \in [-\frac{3^w-1}{2}, \frac{3^w-1}{2}]$ as specified in Algorithm 5.

Note that, in Algorithm 5, the operation $\bmod_s$ is a signed modular reduction, i.e., integers are reduced in the set $\{-(3^w-1)/2, \ldots, -1, 0, 1, \ldots, (3^w-1)/2\}$. From [13] we know that the length ℓ of this $SWR_{3,w}(k)$

Algorithm 5 $SWR_{3,w}$ - Signed window representation in base 3

Require: An integer $k \geq 0$ and a window length w

Ensure: The $SWR_{3,w}$ of k

while $k > 0$ **do**
  if $k \equiv 0 \mod 3$ **then**
    $k_i \leftarrow 0, k \leftarrow k/3$
  else
    $k_i \leftarrow k \bmod_s 3^w,\ k \leftarrow (k - k_i)/3$
  end if
  $i \leftarrow (i + 1)$
end while

satisfies $\ell \leq \lfloor \log_3(k) \rfloor + 1$ and that the number of non-zero k_i is approximately $\cong \frac{\ell}{w+\frac{1}{2}}$. The Triple-and-add method for scalar multiplication $k \cdot P$ uses the $SWR_{3,w}$ method to recode the scalar k, it then precomputes the points $i \cdot P$ for $0 < i < \frac{3^w}{2}$ and $i \not\equiv 0 \mod 3$ and stores them in a table $T[\cdot]$. Then $k \cdot P$ is computed through a sequence of tripling and addition $3 \cdot P + sign(k_i)T[\,|k_i|\,]$ for $i = \ell, \ell - 1, \ldots, 1, 0$.

Algorithm 6 Triple-and-add with $SWR_{3,w}$

Require: A curve $E(\mathbb{F}_{3^m})$, a point $P \in E(\mathbb{F}_{3^m})$ and a scalar k .

Ensure: $Q = k \cdot P$

$(k_{\ell-1}, \ldots, k_0) \leftarrow SWR_{3,w}(k)$

Precomputations. **for** $i = 0, \ldots, \frac{3^w-1}{2}$ **and** $i \not\equiv 0 \mod 3$ **do** $T[i] \leftarrow i \cdot P$

$Q \leftarrow \mathcal{O}$

for $i = \ell - 1$ **to** 0 **do**
  $Q \leftarrow 3Q$
  $Q \leftarrow Q + sign(k_i)T[\,|k_i|\,]$
end for

The complexity of this approach is equal to the cost of the precomputations which is 3^{w-1} additions plus $(2w - 3)$ triplings, plus the cost of the main loop which requires $\frac{\ell}{w+\frac{1}{2}}$ additions plus ℓ triplings where ℓ is the length of $SWR_{3,w}(k)$ (for further details the reader may refer to [13]).

Comparison of the complexity of the Double-and-add and Triple-and-add approaches. Based on the complexity of the Double-and-add and Triple-and-add methods combined with the complexities of the curve operations given in Table 2, we can derive the number of multiplications (M) for some cryptographic field sizes m. For simplicity, we neglect cubings, cube roots and additions in $\mathbb{F}_{3^m}$ which are generally assumed to be neatly faster than multiplications and inversions. We also assume that a squaring has the same cost

as a multiplication, i.e., $S = M$. The resulting complexities are given in Table 3. We notice that, in the case $a = 1$, the Double-and-add method with $w = 4$ in scaled projective coordinates provides the best complexity results. But we also notice that the Triple-and-add approach with $w = 3$ in scaled projective coordinates has a complexity of the same order of magnitude. For the case $a \neq 1$ the Triple-and-add approach with $w = 3$ gives the best complexity results.

Table 3

Complexity of Double-and-add and Triple-and-add for a few cryptographic sizes

	Coordinates	Cost											
		$m = 127$			$m = 147$			$m = 187$			$m = 251$		
		DA $w = 4$	TA $w = 2$	TA $w = 3$	DA $w = 4$	TA $w = 2$	TA $w = 3$	DA $w = 4$	TA $w = 2$	TA $w = 3$	DA $w = 4$	TA $w = 2$	TA $w = 3$
any a	Jacobian	2033	1400	1390	2318	1607	1573	2907	2023	1938	3836	2689	2523
$a = 1$	ML	2124	1249	1289	2422	1433	1455	3036	1801	1786	4006	2390	2316
	Scaled proj.	1139	1172	1195	1288	1344	1347	1595	1688	1649	2080	2238	2134

DA=Double-and-add, TA=Triple-and-add

5 PARALLEL SCALAR MULTIPLICATION IN $E(\mathbb{F}_{3^m})$

We present in this section a Third-and-add approach to perform a scalar multiplication in $E(\mathbb{F}_{3^m})$. This method uses a thirding operation on the curve which consists in a multiplication of a point P by the inverse of 3. We will then take advantage of this new Third-and-add approach to parallelize the scalar multiplication by concurrently performing the Third-and-add and Triple-and-add or Double-and-add algorithms.

5.1 Thirding when $a = 1$

We consider two points $P = (x_P, y_P)$ and $Q = (x_Q, y_Q)$ on an elliptic curve $E(\mathbb{F}_{3^m})$ given by an equation $y^2 = x^3 + x^2 + b$, i.e., with $a = 1$. The tripling formula in affine coordinates is as follows: if $P = 3 \cdot Q$, the expression of (x_P, y_P) in terms of (x_Q, y_Q) is

$$x_P = \frac{y_Q^6}{(x_Q^3+b)^2} - \frac{x_Q^3}{x_Q^3+b}, \qquad y_P = \frac{y_Q^9}{(x_Q^3+b)^3} - \frac{y_Q^3}{x_Q^3+b}. \tag{8}$$

If we set

$$B = \frac{y_Q^3}{x_Q^3 + b} \text{ and } A = \frac{x_Q^3}{x_Q^3 + b}, \tag{9}$$

the previous equation (8) rewrites as

$$x_P = B^2 - A, \qquad y_P = B^3 - B. \tag{10}$$

Computing the thirding of P, i.e., $Q = \left[\frac{1}{3}\right] \cdot P$, consists of computation of the coordinates x_Q and y_Q in terms of x_P and y_P. Before proceeding to the presentation of the thirding formula, we first need to state some results concerning the solutions of the equation in the variable B of the form $B^3 - B = u$ where u is a fixed element of $\mathbb{F}_{3^m}$ satisfying $Trace(u) = 0$.

Lemma 2 (ThirdTrace). *We assume that the field $\mathbb{F}_{3^m}$ has a degree satisfying $m \neq 0 \mod 3$. The three solutions of the equation $B^3 - B = u$ where $u \in \mathbb{F}_{3^m}$ satisfies $Trace(u) = 0$ are:*

$$B_0 = \begin{cases} -\sum_{i=0}^{(m-1)/3-1} (u^3 - u)^{3^{3i+1}} & \text{if } n \equiv 1 \mod 3, \\ \sum_{i=0}^{(m-2)/3} (u^3 - u)^{3^{3i}} & \text{if } n \equiv 2 \mod 3, \end{cases} \tag{11}$$

and $B_1 = B_0 + 1$ and $B_2 = B_0 + 2$.

Proof: It is clear that if B_0 is a solution of $B^3 - B = u$ then B_1 and B_2 are also solutions of this equation:

$$(B_0 + 1)^3 - (B_0 + 1) = B_0^3 - B_0 = u \text{ and } (B_0 + 2)^3 - (B_0 + 2) = B_0^3 - B_0 = u.$$

There is no other solution since the equation is of degree three in B. Now, we check that the expression of B_0 given in (11) when $m \equiv 1 \mod 3$ is a solution of the equation $B^3 - B = u$. We have

$$\begin{aligned} B_0^3 - B_0 &= -\left(\sum_{i=0}^{(m-1)/3-1} (u^3 - u)^{3^{3i+2}} - \sum_{i=0}^{(m-1)/3-1} (u^3 - u)^{3^{3i+1}}\right) \\ &= -\left(\sum_{i=0}^{(m-1)/3-1} u^{3^{3i+3}} + \sum_{i=0}^{(m-1)/3-1} u^{3^{3i+2}} + \sum_{i=0}^{(m-1)/3-1} u^{3^{3i+1}}\right) \\ &= -\left(\sum_{i=1}^{m-1} u^{3^i}\right). \end{aligned}$$

Now since $Trace(u) = \sum_{i=0}^{m-1} u^{3^i} = 0$, we have $B_0^3 - B_0 = u^{3^0} - Trace(u) = u$. This means that B_0 is a solution of $B^3 - B = u$. We now consider the second case, i.e., $m \equiv 2 \mod 3$:

$$\begin{aligned} B_0^3 - B_0 &= \sum_{i=0}^{(m-2)/3} (u^3 - u)^{3^{3i+1}} - \sum_{i=0}^{(m-2)/3} (u^3 - u)^{3^{3i}} \\ &= \sum_{i=0}^{(m-2)/3} u^{3^{3i+2}} + \sum_{i=0}^{(m-2)/3} u^{3^{3i+1}} + \sum_{i=0}^{(m-2)/3} u^{3^{3i}} \\ &= \sum_{i=0}^{m} u^{3^i}. \end{aligned}$$

and since $Trace(u) = \sum_{i=0}^{m-1} u^{3^i} = 0$ we have $B_0^3 - B_0 = u^{3^m} = u$ as required. □

In the sequel, for m an odd integer, we will call the third-trace of $B \in \mathbb{F}_{3^m}$ the element

$$ThirdTrace(B) = \begin{cases} -\sum_{i=0}^{(m-1)/3-1} B^{3^i} \text{ if } m \equiv 1 \mod 3, \\ \sum_{i=0}^{(m-2)/3} B^{3^i} \text{ if } m \equiv 2 \mod 3. \end{cases}$$

Lemma 3. *We consider an elliptic curve $E(\mathbb{F}_{3^m})$ defined by $y^2 = x^3 + x^2 + b$ over $\mathbb{F}_{3^m}$ such that $m \not\equiv 0 \mod 3$. We assume that the order of the curve N is such that $N = 3N'$ and $N' \not\equiv 0 \mod 3$. Then the two following assertions hold:*

i) Let $P = (x_P, y_P) \in E(\mathbb{F}_{3^m})$ satisfying $Trace(y_P) = 0$ then there are three points $Q_0 = (x_0, y_0), Q_1 =$

(x_1, y_1) and $Q_2 = (x_2, y_2)$ on the curve such that $3 \cdot Q_i = P$. Their coordinates can be computed as follows:

$$\boxed{\begin{array}{l} B_0 \leftarrow ThirdTrace(y_P^3 - y_P), \quad B_1 \leftarrow B_0 + 1, \quad B_2 \leftarrow B_2 + 2. \\ A_0 \leftarrow B_0^2 - x_P, \quad A_1 \leftarrow A_0 + 2B_0 + 1, \quad A_2 \leftarrow A_0 + 4B_0 + 4. \\ x_0 \leftarrow \sqrt[3]{\frac{bA_0}{1-A_0}}, \quad x_1 \leftarrow \sqrt[3]{\frac{bA_1}{1-A_1}}, \quad x_2 \leftarrow \sqrt[3]{\frac{bA_2}{1-A_2}}. \\ y_0 \leftarrow \sqrt[3]{\frac{bB_0}{1-A_0}}, \quad y_1 \leftarrow \sqrt[3]{\frac{bB_1}{1-A_1}}, \quad y_2 \leftarrow \sqrt[3]{\frac{bB_2}{1-A_2}}. \end{array}} \tag{12}$$

ii) If $P = (x_P, y_P) \in E(\mathbb{F}_{3^m})$ satisfies $Trace(y_P) = 0$ it has an order N', and if $P = (x_P, y_P) \in E(\mathbb{F}_{3^m})$ satisfies $Trace(y_P) \neq 0$ then it has an order equal to $3N'$.

Proof:

- We proceed to the proof of i): we first prove that the points $(x_0, y_0), (x_1, y_1)$ and (x_2, y_2) satisfy (8). We look for the solutions of (8): from Lemma 2 and since $Trace(y_P) = 0$ we deduce that $B_0 = ThirdTrace(y_P), B_1 = B_0 + 1, B_2 = B_0 + 2$ are the three solutions of the degree three polynomial equation $y_P = B^3 - B$. From (10) we get the three corresponding values A_0, A_1 and A_2 associated to B_0, B_1 and B_2 respectivelly:

$$A_0 \;=\; B_0^2 - x_P, \quad A_1 \;=\; B_1^2 - x_P, \quad A_2 \;=\; B_2^2 - x_P.$$

Finally, we deduce the possible solutions $(x_0, y_0), (x_1, y_1)$ and (x_2, y_2) of (10) in terms of A_0, A_1, A_2 and B_0, B_1, B_2. We first remark that

$$A_i = \frac{x_i^3}{x_i^3 + b} \Leftrightarrow x_i = \sqrt[3]{\frac{bA_i}{1 - A_i}}.$$

This implies the respective expressions of x_0, x_1, x_2 in terms of A_0, A_1 and A_2 given in (12). Now, we compute the values for y_0, y_1 and y_2:

$$y_i \;=\; \sqrt[3]{B_i}(x_i + \sqrt[3]{b}) = \sqrt[3]{B_i}\left(\sqrt[3]{\tfrac{bA_i}{1-A_i}} + \sqrt[3]{b}\right) = \sqrt[3]{\tfrac{bB_i}{1-A_i}}$$

This means that the three points $(x_0, y_0), (x_1, y_1)$ and (x_2, y_2) of (12) are the solutions of (8). We now prove that these three points $(x_0, y_0), (x_1, y_1)$ and (x_2, y_2) are on $E(\mathbb{F}_{3^m})$. We take $i \in \{0, 1, 2\}$ and then we use the fact that (x_P, y_P) satisfy the curve equation:

$$y_P^2 = x_P^3 + x_P^2 + b.$$

We now replace y_P and x_P by their respective expression in terms of x_i and y_i:

$$\begin{array}{rcl} \left(\frac{y_i^9}{(x_i^3+b)^3} - \frac{y_i^3}{x_i^3+b}\right)^2 &=& \left(\frac{y_i^6}{(x_i^3+b)^2} - \frac{x_i^3}{x_i^3+b}\right)^3 + \left(\frac{y_i^6}{(x_i^3+b)^2} - \frac{x_i^3}{x_i^3+b}\right)^2 + b \\ \Longleftrightarrow \quad \frac{y_i^{18}}{(x_i^3+b)^6} + \frac{y_i^{12}}{(x_i^3+b)^4} + \frac{y_i^6}{(x_i^3+b)^2} &=& \frac{y_i^{18}}{(x_i^3+b)^6} - \frac{x_i^9}{(x_i^3+b)^3} + \frac{y_i^{12}}{(x_i^3+b)^4} + \frac{y_i^6 x_i^3}{(x_i^3+b)^3} + \frac{x_i^6}{(x_i^3+b)^2} + b \end{array}$$

Now the terms $\frac{y_i^{18}}{(x_i^3+b)^6}$ and $\frac{y_i^{12}}{(x_i^3+b)^4}$ which appear on each side of the equation cancel. We then move the term $\frac{y_i^6 x_i^3}{(x_i^3+b)^3}$ to the left side of the equation and this gives:

$$\begin{array}{rcl} \frac{y_i^6}{(x_i^3+b)^2} - \frac{y_i^6 x_i^3}{(x_i^3+b)^3} &=& -\frac{x_i^9}{(x_i^3+b)^3} + \frac{x_i^6}{(x_i^3+b)^2} + b \\ \Longleftrightarrow \quad \frac{y_i^6(x_i^3+b-x_i^3)}{(x_i^3+b)^3} &=& -\frac{x_i^9}{(x_i^3+b)^3} + \frac{x_i^6}{(x_i^3+b)^2} + b. \end{array}$$

Now, we multiply the equation by $(x_i^3+b)^3$ and we obtain:

$$\begin{aligned} & by_i^6 &=& -x_i^9 + x_i^6(x_i^3+b) + b(x_i^3+b)^3 \\ \iff & by_i^6 &=& bx_i^9 + bx_i^6 + b^4. \end{aligned}$$

We finally obtain that $y_i^2 = x_i^3 + x_i^2 + b$ after a division by b and taking the cube root of each side of the equation. This concludes the proof of i).

- We proceed to the proof of ii). Let $P = (x_P, y_P) \in E(\mathbb{F}_{3^m})$ of order N'. Then since $\gcd(N', 3) = 1$ there exists s such that $3s = 1 \mod N'$. Now, if we set $Q = s \cdot P$ this point satisfies $3 \cdot Q = 3s \cdot P = P$. Let $B = \frac{y_Q^3}{x_Q^3+b}$ then since $3 \cdot Q = P$ we have from (10) that $y_P = B^3 - B$ which implies

$$Trace(y_P) = Trace(B^3) - Trace(B) = 0$$

since $Trace(B^3) = Trace(B)$. We now assume that $P = (x_P, y_P) \in E(\mathbb{F}_{3^m})$ is of order $3N'$, then it cannot satisfies $Trace(y_P) = 0$, otherwise using i) it would exists Q_1, Q_2 and Q_3 satisfying $3Q_i = P$. This means that Q_i would have an order equal to $9N'$ which contradicts the fact that $E(\mathbb{F}_{3^m})$ has order $3N'$. This ends the proof of ii).

□

The previous lemma tells us that for a given point P of order N' we compute the point $Q = \left[\frac{1}{3}\right] \cdot P$ of order N' by first computing $Q_0 = (x_0, y_0), Q_1 = (x_1, y_1)$ and $Q_2 = (x_2, y_2)$ given by (12) and then selecting the point Q_i such that $Trace(y_i) = 0$.

To implement the thirding formula (12) efficiently we can use the following strategies:

- The three inversions $(1-A_0)^{-1}, (1-A_1)^{-1}$ and $(1-A_2)^{-1}$ are costly operation. These three inversions can be performed through only one inversion plus few multiplications. Indeed, we use the strategy proposed by Montgomery: we first compute the product $D = (1-A_0)(1-A_1)(1-A_2)$, then compute its inverse $D^{-1} = (1-A_0)^{-1}(1-A_1)^{-1}(1-A_2)^{-1}$ and finally deduce $(1-A_0)^{-1} = D^{-1}(1-A_1)(1-A_2)$ and $(1-A_1)^{-1} = D^{-1}(1-A_0)(1-A_2)$ and $(1-A_2)^{-1} = D^{-1}(1-A_0)(1-A_1)$.
- In order to avoid some trace computation and some multiplications in the computations of $(x_i, y_i), i = 0, 1, 2$, we can proceed by first computing y_0 and then we compute $Trace(y_0)$ and if it is equal to zero then we compute x_0 and return (x_0, y_0) otherwise we compute y_1 and its trace $Trace(y_1)$. Again if $Trace(y_1) = 0$ we compute x_1 and return (x_1, y_1) otherwise we compute y_2 and x_2 and return (x_2, y_2).

5.2 Thirding when $a = -1$

We consider the case of elliptic curves $E(\mathbb{F}_{3^m})$ defined by $y^2 = x^3 - x^2 + b$, i.e., with $a = -1$. Let $P = (x_P, y_P)$ and $Q = (x_Q, y_Q)$ be two points on $E(\mathbb{F}_{3^m})$. Based on (7), if $P = 3 \cdot Q$, the expression of (x_P, y_P) in terms of (x_Q, y_Q) is as follows:

$$x_P = \frac{y_Q^6}{(x_Q^3+b)^2} + \frac{x_Q^3}{x_Q^3+b}, \qquad y_P = -\frac{y_Q^9}{(x_Q^3+b)^3} - \frac{y_Q^3}{x_Q^3+b}.$$

So if we define $B = \frac{y_Q^3}{x_Q^3+b}$ and $A = \frac{x_Q^3 a}{x_Q^3+b}$, then the previous equation rewrites as

$$x_P = B^2 - A, \qquad y_P = -(B^3 + B). \tag{13}$$

The process to derive a thirding formula in this case is similar to the case $a = 1$ of Subsection 5.1. We first need to solve the equation $y_P = -(B^3 + B)$ in the variable B, and then we will derive the corresponding solution for A, x_Q and y_Q. We only deal with the case where m is odd which is the case in practice.

Lemma 4. *Let $u \in \mathbb{F}_{3^m}$ and we assume that m is odd. Then the equation $B^3 + B = u$ has a unique solution which is*

$$B = \sum_{i=0}^{(m-1)/2} u^{3^{2i}} + Trace(u).$$

In the sequel we will denote $HalfTrace(B) = \sum_{i=0}^{(m-1)/2} u^{3^{2i}}$ the half-trace of B.

Proof: We first check that $B = \sum_{i=0}^{(m-1)/2} u^{3^{2i}} + Trace(u)$ is a solution of the equation $B^3 + B = u$. We have

$$\begin{aligned} B^3 + B &= \left(\sum_{i=0}^{(m-1)/2} u^{3^{2i}} + Trace(u)\right)^3 + \left(\sum_{i=0}^{(m-1)/2} u^{3^{2i}} + Trace(u)\right) \\ &= \sum_{i=0}^{(m-1)/2} u^{3^{2i+1}} + \sum_{i=0}^{(m-1)/2} u^{3^{2i}} + Trace(u)^3 + Trace(u) \\ &= \sum_{i=0}^{m} u^{3^i} + Trace(u)^3 + Trace(u) \end{aligned}$$

But, now, we notice that $\sum_{i=0}^{m} u^{3^i} = Trace(u) + u^{3^m} = Trace(u) + u$ and since $Trace(u) \in \mathbb{F}_3$ we have $Trace(u)^3 = Trace(u)$. This implies :

$$B^3 + B = Trace(u) + u + 2Trace(u) = u.$$

We now prove that the solution is unique. Indeed if B_1 and B_2 are two solutions of $B^3 + B = u$ then $C = B_1 - B_2$ satisfies $C^3 + C = (B_1^3 + B_1) - (B_2^3 + B_2) = u - u = 0$. But this means that if $C \neq 0$ then $C^2 + 1 = 0$ but such element are in $\mathbb{F}_{3^2} \backslash \mathbb{F}_3$ and since m is assumed to be odd such element are not in $\mathbb{F}_{3^m}$. In other words C must be equal to zero and thus B_1 and B_2 are equal. This concludes the proof. □

We are now able to generate the thirding formula in the case $a = -1$ and m odd.

Lemma 5. *We consider an elliptic curve $E(\mathbb{F}_{3^m})$ given by $y^2 = x^3 - x^2 + b$ over $\mathbb{F}_{3^m}$ with m odd. Let $P = (x_P, y_P)$ and $Q = (x_Q, y_Q)$ be two points on $E(\mathbb{F}_{3^m})$. The order N' of the curve satisfies $N' \not\equiv 0 \mod 3$ and if $3 \cdot Q = P$ then the coordinates of Q can be computed in terms of (x_P, y_P) as follows:*

$$\boxed{\begin{aligned} &B \leftarrow HalfTrace(-y_P) + Trace(-y_P), \\ &A \leftarrow B^2 - x_P, \\ &x_Q \leftarrow \sqrt[3]{\tfrac{bA}{1-A}}, \\ &y_Q \leftarrow \sqrt[3]{\tfrac{bB}{1-A}}. \end{aligned}} \tag{14}$$

Proof: To check that $N' \not\equiv 0 \mod 3$ we follow the same outline as in the proof of Lemma 1 in [15]. If $N' \equiv 0 \mod 3$ there exists a non trivial point of order three $T = (x_T, y_T)$ on the curve. If we look at the

tripling formula (7) and since $3 \cdot T = \mathcal{O}$, we must have $a^2(x_T^3 + b) = 0$. This implies that $x_T = \sqrt[3]{-b}$ and also that $y_T^2 = x_T^3 + ax_T^2 + b = -1 \cdot \sqrt[3]{b}^2$. But such y_T exists in $\mathbb{F}_{3^m}$ if and only if -1 is a square in $\mathbb{F}_{3^m}$ but this is not possible since m is odd and, thus, $\mathbb{F}_{3^m}$ does not contain the square roots of -1. The thirding formula is a straightforward consequence of Lemma 4 which provides the expression of B in terms of y_P. The expressions of A, x_Q and y_Q in terms of x_P and y_P are obtained using the same approach as in the proof of Lemma 3. □

5.3 Parallel scalar multiplication in $E(\mathbb{F}_{3^m})$

We consider a curve $E(\mathbb{F}_{3^m})$ given by $y^2 = x^3 + ax^2 + b$ where $a \in \{1, -1\}$. The thirding formulas presented in the two previous subsections provide some new approaches to implement of the scalar multiplication on $E(\mathbb{F}_{3^m})$. Indeed, let k be the scalar and let P be a point on $E(\mathbb{F}_{3^m})$ of order $N < 3^\ell$ which is assumed to be prime. If we denote $k' = 3^{\ell-1} \times k \mod N$, then if we write $k' = \sum_{i=0}^{\ell-1} k'_i 3^i$ in base 3, i.e., with $k'_i \in \{0, 1, 2\}$ we have

$$\begin{aligned} k &= 3^{-(\ell-1)}k' \mod N \\ &= \textstyle\sum_{i=0}^{\ell-1} k'_i 3^{i-\ell+1} \mod N \\ &= \textstyle\sum_{i=0}^{-(\ell-1)} k'_{i+\ell-1} 3^i \mod N. \end{aligned}$$

Consequently, the scalar multiplication $k \cdot P$ can be performed through a sequence of thirdings and additions:

$$k \cdot P = \sum_{i=0}^{-(\ell-1)} k'_{i+\ell-1} 3^i \cdot P.$$

We extend this idea to a Third-and-add approach which uses $SWR_{3,w}$ to recode k'. Details of this approach are given in Algorithm 7.

Algorithm 7 Third-and-add with $SWR_{3,w}$

Require: A curve $E(\mathbb{F}_{3^m})$, a point $P \in E(\mathbb{F}_{3^m})$ of prime order $N < 3^\ell$ and a scalar $k \in [1, N]$.

Ensure: $Q = k \cdot P$

$k' \leftarrow k \cdot 3^\ell \mod N$

$(k'_\ell, \ldots, k'_0) \leftarrow SWR_{3,w}(k')$.

Precomputations. **for** $i = 0, \ldots, \frac{3^w-1}{2}$ and $i \not\equiv 0 \mod 3$ **do** $T[i] \leftarrow iP$.

$Q \leftarrow \mathcal{O}$

for $i = \ell$ **to** 0 **do**

  $Q \leftarrow \left[\frac{1}{3}\right] \cdot Q$.

  $Q \leftarrow Q + sign(k_i)T[\,|k_i|\,]$

end for

The Third-and-add approach is not interesting in practice since the thirding operation appears to be quite costly compared to tripling formula. But we can take advantage of the Third-and-add approach to

implement the scalar multiplication in parallel fashion. Indeed, we can split the integer into two parts k_1 and k_2 as follows : we set a split value $0 < n < \ell = \lceil \log_3(N) \rceil$ and we compute $k' = k \cdot 3^n \mod N$. Then if we write $k' = \sum_{i=0}^{\ell} k'_i 3^i$ in base 3 we can rewrite k as follows:

$$\begin{aligned} k &= 3^{-n}k' \mod N \\ &= \left(\sum_{i=0}^{n-1} k'_i 3^{i-n}\right) + \left(\sum_{i=n}^{\ell} k'_i 3^{i-n}\right) \mod N \\ &= \underbrace{\left(\sum_{i=1}^{n} k'_{n-i} 3^{-i}\right)}_{k_1} + \underbrace{\left(\sum_{i=0}^{\ell-n} k'_{i+n} 3^{i}\right)}_{k_2} \mod N. \end{aligned}$$

Then the scalar multiplication can be split into two concurrent algorithms: a Triple-and-add algorithm which performs $k_2 \cdot P$ and a Third-and-add algorithm which performs $k_1 \cdot P$. The result is obtained after a final addition $k \cdot P = k_1 \cdot P + k_2 \cdot P$.

5.4 Implementation and timing results

The platform used for our experimentation is the same as in Subsection 3.3: an Intel Core i7-2400 with Ubuntu 12.04 and gcc 4.6.3. The two fields considered in our implementations are $\mathbb{F}_{3^{127}} = \mathbb{F}_3[t]/(f(t))$ with $f(t) = (t^{128}+t^{77}+1)/(t-1)$ and $\mathbb{F}_{3^{251}} = \mathbb{F}_3[t]/(t^{251}+t^{26}-1)$. A field element $a = \sum_{i=0}^{m-1}(a_{i,0}+2a_{i,1})t^i$ where $a_{i,0}, a_{i,1} \in \{0,1\}$ is decomposed as $a = a_0 + 2a_1$ where $a_0 = \sum_{i=0}^{m-1} a_{i,0}t^i$ which is stored in one (resp. two) 128 bit word for $m = 127$ (resp. $m = 251$) and $a_1 = \sum_{i=0}^{m-1} a_{i,1}t^i$ which is also stored in one (resp. two) 128 bit word for $m = 127$ (resp. $m = 251$).

- *Addition.* We use the approach presented in [1]: let a and b be two elements in $\mathbb{F}_{3^m}$ and let a_0 and a_1 and b_0 and b_1 be the decomposition of a and b as described above. The formula used to compute the addition $c = a + b$ in $\mathbb{F}_{3^m}$ is the following

$$u = (a_0 \,|\, a_1) \,\&\, (b_0 \,|\, b_1), \text{ and } c_0 = u \wedge (a_0 \,|\, b_0), \text{ and } c_1 = u \wedge (a_1 \,|\, b_1),$$

  and $c = c_0 + 2c_1$ where $|, \&$ and $\wedge$ are the bitwise logical operations OR, AND and XOR, respectively.
- *Multiplication.* Again, we follow the strategy used in [1]: we adapt the Lopez-Dahab algorithm for the multiplication in $\mathbb{F}_{2^m}$ of [12] to the case of $\mathbb{F}_{3^m}$. Indeed, we use a shift-and-add method with window size $w = 4$. To compute $a \times b$, we first precompute all the products $(a_0 + a_1 t + a_2 t^2 + a_3 t^3) \times b$ for all nibbles $[a_0, a_1, a_2, a_3] \in \{0,1\}^4$ and store these products in a table $T[\,\cdot\,]$. Then a is split in nibbles and the product is computed by a sequence of table-look-ups, additions and shifts. A shift by 8 is used preferably to shift by 4 since it is cheaper on 128 bit registers.
- *Cubing.* The cubing of an element $a = \sum_{i=0}^{m-1} a_i t^i$ is $a^3 = \sum_{i=0}^{m-1} a_i t^{3i} \mod f(t)$. We then have to insert two zeros between each coefficient a_i and reduce the result modulo $f(t)$. We use a strategy inspired from the implementation approach for the squaring in $\mathbb{F}_{2^m}$ of [3]. Indeed, the instruction `byte_shuffle` makes possible to apply in parallel an S-box $S : \{0,1\}^4 \to \{0,1\}^8$ to the least significant four bits of each byte of a 128 bit word. So, we use it to replace the least significant nibble

of each byte of a 128 bit word with the corresponding nibble with inserted zeros. The reduction is performed with a few shifts and additions due to the specific form of the used irreducible polynomials $f(t)$.

- *Cube Root.* The cube root of an element $a = \sum_{i=0}^{m-1} a_i t^i$ can be expressed as follows

$$\sqrt[3]{a} = \left(\sum_{i=0}^{\lceil m/3 \rceil - 1} a_{3i} t^i \right) + \sqrt[3]{t} \left(\sum_{i=0}^{\lceil m/3 \rceil - 1} a_{3i+1} t^i \right) + \sqrt[3]{t^2} \left(\sum_{i=0}^{\lceil m/3 \rceil - 1} a_{3i+2} t^i \right) \mod f(t).$$

  We then need to separate the coefficients a_i into three parts: one containing a_i with $i \equiv 0 \mod 3$, the second with a_i with $i \equiv 1 \mod 3$, and the third part containing a_i with $i \equiv 2 \mod 3$. We perform the separation by performing some masking and by using the `byte_shuffle` instruction to remove the remaining zeros. The element $\sqrt[3]{t}$ is precomputed and is sparse for the considered fields so the multiplications by $\sqrt[3]{t}$ and $\sqrt[3]{t}^2$ are easy to implement and fast.

- *ThirdTrace, HalfTrace and Multi-cubing.* The third-trace is a linear function, i.e., $ThirdTrace(a+b) = ThirdTrace(a) + ThirdTrace(b)$ and the same is true for the half-trace $HalfTrace(a+b) = HalfTrace(a) + HalfTrace(b)$ and the multi-cubing $(a+b)^{3^i} = a^{3^i} + b^{3^i}$ for $i \geq 0$. We describe the strategy we employed for the implementation of the $ThirdTrace$ operation, the $HalftTrace$ and multi-cubing were implemented in similar approach. For the computation of $ThirdTrace$ we first precompute and store in a table tab_TT the value

$$tab\_TT[i][a_{4i} + 2a_{4i+1} + 4a_{4i+2} + 8a_{4i+3}] = ThirdTrace\left((a_{4i} + a_{4i+1}t + a_{4i+2}t^2 + a_{4i+3}t^3)t^{4i}\right)$$

  where $(a_{4i}, a_{4i+1}, a_{4i+2}, a_{4i+3}) \in \{0,1\}^4$. To compute $ThirdTrace(a)$ for an element $a = \sum_{i=0}^{m-1} \in \mathbb{F}_{3^m}$, we decompose a into a sequence of nibbles $[a_{4i}, a_{4i+1}, a_{4i+2}, a_{4i+3}]$ for $0 \leq i < m/4$, and then sum the $m/4$ values $tab\_TT[i][a_{4i} + 2a_{4i+1} + 4a_{4i+2} + 8a_{4i+3}]$.

- *Inversion.* To perform the inversion we use the approach of Itoh-Tsujii [8]. The inverse of $a \in \mathbb{F}_{3^m}$ is given by:

$$a^{-1} = a^{3^{\sum_{i=1}^{m-1} 3^i}} \times a^{3^{\sum_{i=0}^{m-1} 3^i}}.$$

  This expression can be computed through a short sequence of multi-cubings and multiplications. Indeed, if we denote $e_k = \sum_{i=0}^{k-1} 3^i$, then the following identity holds

$$(a^{e_k})^{3^{k'}} \times a^{e_{k'}} = a^{e_{k+k'}},$$

  since $e_k \times 3^{k'} + e_{k'} = \sum_{i=0}^{k+k'-1} 3^i = e_{k+k'}$. This property enables to compute $a^{-1} = (a^{e_{m-2}})^3 \times e_{m-1}$ using an addition chain. Indeed for $m = 127$ the sequence of e_k where k follows the addition chain $1 \to 2 \to 3 \to 6 \to 7 \to 14 \to 15 \to 31 \to 62 \to 63 \to 126$ can be used to compute a^{-1}. For $m = 251$ we use the chain $1 \to 2 \to 3 \to 6 \to 7 \to 14 \to 15 \to 31 \to 62 \to 124 \to 125 \to 250 \to 251$.

We have implemented the Double-and-add, Triple-and-add and Third-and-add algorithms for the scalar multiplication along with their parallel counterparts in $E(\mathbb{F}_{3^{127}})$ and $E(\mathbb{F}_{3^{251}})$. The curves chosen have

either $a = 1$ and order $N = 3p$ where p is prime, either $a = -1$ and a prime order N. The resulting timings are shown in Table 4.

Table 4

Timings (in clock cycles and millisecond) for scalar multiplication in $E(\mathbb{F}_{3^{127}})$ and $E(\mathbb{F}_{3^{251}})$

Curve Type	Method	Window size of NAF_w	$m = 127$			$m = 251$		
			#CC	ms	split	#CC	ms	split
$a = 1$	Double-and-add	4	615346	0.18	-	3478306	1.02	-
	Triple-and-add	2	699185	0.20	-	4128228	1.21	-
	Triple-and-add	3	697243	0.20	-	3876161	1.14	-
	Third-and-add	2	2774417	0.81	-	12315146	3.62	-
	Third-and-add	3	2781878	0.81	-	12325814	3.62	-
	Parallel (Triple-and-add,Third-and-add)	(2,2)	628639	0.18	24	3735006	1.09	34
	Parallel (Triple-and-add,Third-and-add)	(3,3)	645469	0.18	24	3573207	1.05	32
	Parallel (Double-and-add,Third-and-add)	(4,3)	586731	0.17	20	3282243	0.96	26
$a = -1$	Double-and-add	4	1323311	0.38	-	8938046	2.62	-
	Triple-and-add	2	845028	0.24	-	5528848	1.62	-
	Triple-and-add	3	846891	0.24	-	5068188	1.49	-
	Third-and-add	2	3052728	0.89	-	11727985	3.44	-
	Third-and-add	3	2750622	0.80	-	11010147	3.23	-
	Parallel (Triple-and-add,Third-and-add)	(2,2)	747425	0.21	27	4712970	1.38	48
	Parallel (Triple-and-add,Third-and-add)	(3,3)	744853	0.21	30	4383200	1.28	46

The timings shown in Table 4 are coherent to the complexity shown in Table 2:

- For $a = 1$ the best non-parallelized approach is the Double-and-add scalar multiplication while the Triple-and-add approaches are slower. The improvement provided by the parallelization is of 4.88% for $m = 127$ and 5.73% for $m = 251$.
- For $a = -1$ the best non-parallelized approach is the Triple-and-add approach with $w = 2$ for $m = 127$ and $w = 3$ for $m = 251$. The parallelization with $w = 3$ provides a speed-up of 12.3% for $m = 127$ and 13.8% for $m = 251$ compared to the best non-parallelized approaches.

It appears that we could not find in the literature implementation results sufficiently recent for elliptic curve scalar multiplication on $E(\mathbb{F}_{3^m})$. Specifically, most of the published results in characteristic three concern pairing implementation on $E(\mathbb{F}_{3^m})$ where m is much larger. So we could compare the results presented above with some related timings for similar curves $E(\mathbb{F}_{3^m})$.

6 Conclusion

We have presented in this paper two new strategies for parallel implementation of scalar multiplication. The first one concerns elliptic curves over binary fields $\mathbb{F}_{2^m}$: we have proposed a halving form of the

Montgomery point multiplication. This approach can be used in parallel to the original Montgomery multiplication in order to concurrently computes part of the scalar multiplication. The second strategy concerns implementation of elliptic curve scalar multiplication in $E(\mathbb{F}_{3^m})$: we provide point thirding formulas on two sub-family of curves. Point thirding is the analog in $E(\mathbb{F}_{3^m})$ to point halving in binary elliptic curves $E(\mathbb{F}_{2^m})$. This leads to a Third-and-add approach for scalar multiplication and also a parallel approach which concurrently performs Third-and-add and Triple-and-add or Double-and-add algorithms.

We have implemented these two new parallel strategies. The timings obtained shows a speed-up of 5-10% compared to the original non parallelized version for the case of $E(\mathbb{F}_{2^m})$ and a speed-up of 5-13% for scalar multiplication $E(\mathbb{F}_{3^m})$.

REFERENCES

[1] O. Ahmadi, D. Hankerson, and A. Menezes. Software implementation of arithmetic in F_{3^m}. In *WAIFI*, volume 4547 of *LNCS*, pages 85–102, 2007.

[2] D.J. Bernstein and T. Lange. eBACS: ECRYPT Benchmarking of Cryptographic Systems, November 2013.

[3] D.F. Aranha and J. López and D. Hankerson. Efficient Software Implementation of Binary Field Arithmetic Using Vector Instruction Sets. In *Progress in Cryptology - LATINCRYPT 2010*, volume 6212 of *LNCS*, pages 144–161. Springer, 2010.

[4] R.R. Farashahi, H. Wu, and C. Zhao. Efficient Arithmetic on Elliptic Curves over Fields of Characteristic Three. In *Selected Areas in Cryptography (SAC 2012)*, volume 7707 of *LNCS*, pages 135–148. Springer, 2013.

[5] K. Fong, D. Hankerson, J. López, and A. Menezes. Field Inversion and Point Halving Revisited. *IEEE Trans. Computers*, 53(8):1047–1059, 2004.

[6] M. Hamburg. Fast and compact elliptic-curve cryptography. Technical Report 2012/309, Cryptology ePrint Archive, 2012.

[7] D. Hankerson, J. López Hernandez, and A. Menezes. Software Implementation of Elliptic Curve Cryptography over Binary Fields. In *CHES 2000*, volume 1965 of *LNCS*, pages 1–24. Springer, 2000.

[8] T. Itoh and S. Tsujii. A Fast Algorithm for Computing Multiplicative Inverses in GF(2^m) Using Normal Bases. *Inf. Comput.*, 78(3):171–177, 1988.

[9] K.-H. Kim, S.I. Kim, and J.S. Choe. New fast algorithms for arithmetic on elliptic curves over finite fields of characteristic three. Technical Report Report 2007/179, Cryptology ePrint Archive, 2007.

[10] E.W. Knudsen. Elliptic Scalar Multiplication Using Point Halving. In *Advances in Cryptology - ASIACRYPT '99*, 1999.

[11] J. Lopez and R. Dahab. Fast Multiplication on Elliptic Curves Over GF(2^m) without Precomputation. In *CHES 99*, volume 1717, pages 316–327, 1999.

[12] Julio López and Ricardo Dahab. High-Speed Software Multiplication in $\mathbb{F}_{2^m}$. In *INDOCRYPT 2000*, volume 1977 of *LNCS*, pages 203–212. Springer, 2000.

[13] C. Negre. Scalar Multiplication on Elliptic Curves Defined over Fields of Small Odd Characteristic. In *Progress in Cryptology - INDOCRYPT 2005*, volume 3797 of *LNCS*, pages 389–402, 2005.

[14] R. Schroeppel. Elliptic Curve Point Halving Wins Big. In *Second Midwest Arithmetical Geometry in Cryptography Workshop*, Nov. 2000.

[15] N.P. Smart and E.J. Westwood. Point Multiplication on Ordinary Elliptic Curves over Fields of Characteristic Three. *Appl. Algebra Eng. Commun. Comput.*, 13(6):485–497, 2003.

[16] J. Taverne, A. Faz-Hernández, D.F. Aranha, F. Rodríguez-Henríquez, D. Hankerson, and J. López. Speeding scalar multiplication over binary elliptic curves using the new carry-less multiplication instruction. *J. Cryptographic Engineering*, 1(3):187–199, 2011.