Parallel Approaches for Efficient Scalar Multiplication over Elliptic Curve

Christophe Negre[1] and Jean-Marc Robert[1]

[1] *Equipe DALI (Université de Perpignan) and LIRMM (Univ. de Montpellier, CNRS), Perpignan, France*
{*christophe.negre, jean-marc.robert*}*@univ-perp.fr*

Keywords:
Elliptic curve, parallel implementation, scalar multiplication, prime field, binary field.

Abstract:
This paper deals with parallel implementation of scalar multiplication over an elliptic curve. We present parallel approaches which split the scalar into two parts for $E(\mathbb{F}_p)$ or three parts for $E(\mathbb{F}_{2^m})$ and perform in parallel the scalar multiplication with each part of the scalar. We present timing results of these approaches implemented over an Intel Core i7 for NIST binary curves B233, B409 and for the twisted Edwards curve Curve25519 (Bernstein, 2006). For the curves B409 and Curve25519 the proposed approaches improve by at least 10% the computation time of the scalar multiplication.

1 INTRODUCTION

Elliptic curve cryptographic protocols generally involve one or two scalar multiplications on the curve. A scalar multiplication consists in computing $kP = P + \cdots + P$ where k is an integer of several hundreds of bits and P is a point on the curve. This is generally performed through a sequence of hundreds of doublings and additions on the curve using the so-called double-and-add algorithm. It is thus a time consuming part of the cryptographic protocols.

Actual and future processors on personal computers and embedded devices will include an increasing number of cores, enabling more parallelism. The implementation of scalar multiplication might be adapted to these new platforms. A recent interesting work of Taverne *et al.* (Taverne et al., 2011) on binary elliptic curves showed that a curve scalar multiplication can be parallelized through a (double,halve)-and-add approach resulting in an interesting speed-up. GLV (Gallant et al., 2001) is another popular approach to perform a scalar multiplication in parallel fashion (Longa and Sica, 2014; Oliveira et al., 2014), but unfortunately GLV cannot be used on standard curves (P. Gallagher and Furlani, 2009).

We explore in this paper an alternative parallel approach. We split the scalar k in an upper and lower part, and we compute scalar multiplication of each part in parallel. This requires an additional sequence of doublings or halvings to finish the upper part computations before adding the two computed points. We have experimented this approach and the resulting timings show some interesting speed-up for the NIST curve B409 and also for the twisted Edward curve Curve25519 (Bernstein, 2006).

The remaining of the paper is organized as follows: in Section 2 we review basic results on elliptic curves over binary and prime fields. In Section 3 we present our approaches for the parallelization of scalar multiplication. In Section 4 we provide experimental results and we end the paper with some concluding remarks in Section 5.

2 REVIEW OF ELLIPTIC CURVE SCALAR MULTIPLICATION

An elliptic curve over a finite field $\mathbb{F}_q$ is the set of points $(x, y) \in \mathbb{F}_q^2$ satisfying a smooth equation of degree 3 plus a point at infinity $\mathcal{O}$. A commutative group law can be defined on this set of points using the chord and tangent method. This provides doubling and addition formulas consisting

in a number of field operations on the coordinates of the points. The complexity of these formulas can be reduced by choosing appropriate curve equation and system of coordinates. In the remaining of this section, we review curve equation and point operation formulas in the case of elliptic curve defined over binary field and prime field. We also review classical algorithms for scalar multiplication.

2.1 Point operations in $E(\mathbb{F}_p)$

An elliptic curve $E(\mathbb{F}_p)$ over a prime field $\mathbb{F}_p$ is generally defined by a short Weierstrass equation:

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

In this case addition and doubling formulas given by the chord and tangent method are as follows: let $P_1 = (x_1, y_1), P_2 = (x_2, y_2)$ and $P_3 = (x_3, y_3)$, be three points on $E(\mathbb{F}_p)$ such that $P_3 = P_1 + P_2$, then we have:

$$\text{where } \begin{cases} x_3 = \lambda^2 - x_1 - x_2, \\ y_3 = \lambda(x_1 - x_3) - y_1, \end{cases} \quad \begin{cases} \lambda = \frac{y_2 - y_1}{x_2 - x_1} \text{ if } P_1 \neq P_2, \\ \lambda = \frac{3x_1^2 + a}{2y_1} + x_1 \text{ if } P_1 = P_2. \end{cases} \tag{1}$$

In the sequel we will use the two following alternative elliptic curve equations:

- *Twisted Edwards (Hisil et al., 2008).* The twisted Edwards elliptic curve equation is as follows:

  $$ax^2 + y^2 = 1 + dx^2y^2$$

  where a and d are two elements in $\mathbb{F}_p$. In this case there is an unified addition and doubling formula: if $P_1 = (x_1, y_2)$ and $P_2 = (x_2, y_2)$ are in $E(\mathbb{F}_p)$, then the affine coordinates (x_3, y_3) of $P_3 = P_1 + P_2$ are as follows:

  $$\begin{array}{l} x_3 = (x_1y_2 + y_1x_2)/(1 + dx_1x_2y_1y_2), \\ y_3 = (y_1y_2 - ax_1x_2)/(1 - dx_1x_2y_1y_2). \end{array} \tag{2}$$

- *Montgomery curves (Montgomery, 1987).* We consider Montgomery elliptic curves which have the following form:

  $$E : by^2 = x^3 + ax^2 + x \text{ with } (a, b) \in \mathbb{F}_p^2.$$

  Montgomery noticed in (Montgomery, 1987) that for $Q = (x_Q, y_Q)$ the x-coordinate of $2Q$ depends only on x_Q:

  $$x_{2Q} = \frac{(x_Q^2 - 1)^2}{4x_Q(x_Q^2 + ax_Q + 1)}. \tag{3}$$

  Moreover, Montgomery noticed that given two points $Q = (x_Q, y_Q)$ and $P = (x_P, y_P)$ and if we know the difference $R = Q - P$, then we can compute $Q + P$ as follows:

  $$x_{Q+P} = \frac{(x_Qx_P - 1)^2}{x_R(x_P - x_Q)^2}. \tag{4}$$

  Montgomery curves and twisted Edwards curves are isomorphic. A conversion of the point coordinates between these two curves requires only a few field operations.

The formulas in (1), (2) and (3) involve a field inversion which is generally a costly operation. Alternative coordinate systems have been proposed in the case of Weierstrass, Montgomery, twisted Edwards and Jacobi quartic in order to avoid field inversions and reduce the number of field multiplications:

- Extended coordinate system where a quadruplet (X, Y, Z, T) represents the points $x = X/Z, y = Y/Z$ and $T/Z = xy$.
- Regular projective coordinates where a triplet (X, Y, Z) represents a point $x = X/Z, y = Y/Z$.
- Inverted coordinates : a triplet (X, Y, Z) corresponds to a point $x = Z/X$ and $y = Z/Y$.

The cost of a doubling, an addition and a mixed addition in these coordinate systems are given in Table 1.

2.2 Operations in binary elliptic curves

An elliptic curve over a binary field is generally defined by the following short Weierstrass equation:

$$E : y^2 + xy = x^3 + ax^2 + b \text{ with } a, b \in \mathbb{F}_{2^m}. \tag{5}$$

The formulas for addition, doubling and halving on $E(\mathbb{F}_{2^m})$ are as follows:

- *Addition and doubling.* We consider two points $P_1 = (x_1, y_1)$ and $P_2 = (x_2, y_2)$ on the curve $E(\mathbb{F}_{2^m})$. The coordinates (x_3, y_3) of $P_3 = P_1 + P_2$ are given by the following formula:

  $$\begin{array}{l} \begin{cases} x_3 = \lambda^2 + \lambda + x_1 + x_2 + a, \\ y_3 = (x_1 + x_3)\lambda + x_3 + y_1, \end{cases} \\ \text{where } \lambda = \begin{cases} \frac{y_1 + y_2}{x_1 + x_2} & \text{if } P_1 \neq P_2, \\ \frac{y_1}{x_1} + x_1 & \text{if } P_1 = P_2. \end{cases} \end{array} \tag{6}$$

- *Point halving.* In (Knudsen, 1999), Knudsen noticed that if a point $Q = (u, v)$ on $E(\mathbb{F}_{2^m})$ is of odd order, then the point $P = \frac{1}{2}Q$ in

Table 1 – Complexity of curve operations in $E(\mathbb{F}_p)$

Curve form	Coordinates	Doubling	mixed Addition	Addition
Weierstrass ($a=-3$)	Jacobian	$4M+4S$	$8M+3S$	$12M+4S$
Twisted Edwards	Inverted	$3M+4S+M_a+M_{2d}$	$8M+1S+M_a+M_d$	$9M+1S+M_a+M_d$
Jacobi quartic	$XXYZZ$ coord.	$3M+4S$	$6M+3S+M_{a-1}$	$7M+4S+M_{a-1}$
Montgomery	Projective	$2M+M_{(a+2)/4}+2S$	-	$4M+2S$

Table 2 – Complexity of curve operation in $E(\mathbb{F}_{2^m})$ when $a=1$

Coordinates	Doubling	Halving	Mixed-addition	Doubling-mixed-addition	Addition
Affine	$2M+I$	$1M+1SqRt+1QS$	-	$4M+2I$	$2M+I$
Lopez-Dahab	$4M+5S$	-	$9M+4S$	$13M+4S$	$13M+9S$
Kim-Kim	$4M+5S$	-	$8M+4S$	$12M+9S$	$12M+4S$
Lambda proj.	$4M+4S$	-	$8M+2S$	$10M+6S$	$11M+4S$

the subgroup generated by Q is well defined. Knudsen also noticed that the computation of the coordinates of P in terms of the coordinates of Q can be computed efficiently. Indeed, since $Q=2P$ and we can use (6) to derive the coordinates of $P=(x,y)$ in terms of $Q=(u,v)$ as follows:

$$\lambda = x + y/x \quad (7)$$
$$u = \lambda^2 + \lambda + a \quad (8)$$
$$v = x^2 + u(\lambda+1) \quad (9)$$

Knudsen proposed to first solve the quadratic equation (8) in λ, then to compute $x = \sqrt{v+u(\lambda+1)}$ from (9) and to finally compute $y=\lambda x+x$ from (7). A point halving on $E(\mathbb{F}_{2^m})$ has thus a cost of $2M$ plus one square root ($SqRt$) and one quadratic solver (QS). It becomes more efficient if P and Q are represented in lambda coordinates $P=(x_P,\lambda_P=\frac{y_P}{x_P}+x_P)$ and $Q=(y_Q,\lambda_Q=\frac{y_Q}{x_Q}+x_Q)$ since, in this case, a point halving requires only $1M+1SqRt+1QS$.

The following projective coordinate systems are proposed in the literature in order to efficiently implement point operations in $E(\mathbb{F}_{2^m})$:

- *Lopez-Dahab coordinate system (López and Dahab, 1998).* A point is given by a triplet (X,Y,Z) corresponding to the affine point $x=X/Z, y=X/Z^2$.
- *Kim-Kim coordinate system (Kim and Kim,).* A point is given by a quadruplet (X,Y,Z,T), where $T=Z^2$, which corresponds to the affine point $x=X/Z, y=X/T$.
- *Lambda projective coordinate system (Oliveira et al., 2014) .* A point is represented by a triplet (X,L,Z) such that $x=X/Z$, $\lambda=L/Z=y/x+x$ and $y=(L/Z+X/Z)X/Z$.

For explicit point addition and doubling formulas in these projective coordinate systems the reader may refer to (Hankerson et al., 2004; Kim and Kim, ; Oliveira et al., 2014). In Table 2 we report the corresponding cost of each point operation on the curve when $a=1$. We do not report the complexity of a point halving in projective coordinate systems, since in this case the quadratic solver requires a field inversion and makes the halving inefficient. We can notice that Lambda coordinates provide the most efficient curve operations.

2.3 Scalar multiplication algorithms

Double-and-add scalar multiplication. A scalar multiplication kP is generally performed with a sequence of doublings and additions. The scalar k is first recoded with the NAF_w recoding as $k=\sum_{i=0}^{\ell} k_i 2^i$ where k_i is 0 or an odd integer in $[-2^{w-1}, 2^{w-1}]$ and w is the window size. The 2^{w-2} points k_iP for k_i odd in $[0, 2^{w-1}]$ are computed at the beginning of the algorithm and then $R=kP$ is computed through a sequence of doublings and additions $R \leftarrow 2R+k_iP$ for $i=\ell, \ell-1, \ldots, 0$. This method is reported in Algorithm 1. The complexity of this approach is, on average, $\ell+1$ doublings and $\ell/(w+1)+2^{w-2}-1$ additions (see (Hankerson et al., 2004) for a detailed analysis).

Halve-and-add scalar multiplication. In the case of binary elliptic curve, the halving operation makes it possible to use a halve-and-add approach instead of a double-and-add approach. We first recode the scalar as $k=\sum_{i=0}^{\ell} k'_i 2^{-i}$ where k'_i is odd in $[-2^{w-1}, 2^{w-1}]$. Then the scalar multipli-

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)

cation kP is computed by first setting $R \leftarrow P$ and $Q_j \leftarrow \mathcal{O}, j \in \{1, 3, \ldots, 2^{w-1} - 1\}$. Then we perform sequence of halvings and additions showns in steps 4-9 in Algorithm 2. and at the end the result is $Q = \sum_{j \in \{1,3,\ldots,2^{w-1}-1\}} jQ_j$. This method is depicted in Algorithm 2.

The post-computation in the **return** statement (Step 10) is computed with the technique credited to Knuth: $Q_i \leftarrow Q_i + Q_{i+2}$ for i from $2^{w-1} - 3$ to 1, then Q is given by $Q \leftarrow Q_1 + 2\sum_{j \in \{3,\ldots,2^{w-1}-1\}} Q_j$. The total complexity of the halve-and-add scalar multiplication is on average $2^{w-1} + \ell/(w+1)$ point additions and $\ell - 1$ halvings and one doubling. Using the cost of Table 2, we obtain the following complexity in terms of field operations:

$$\begin{aligned} \#Op. = \; & \ell(M + SqRt + QS) \\ & +(8M + 4S)(\ell/(w+1) + 2^{w-1}) \\ & +4M + 4S \end{aligned} \quad (10)$$

Algorithm 2 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: Recode $k = \sum_{i=0}^{\ell} k'_i 2^{-i}$ with k'_i is an odd integer in $[-2^{w-1}, 2^{w-1}]$
2: **for** $j \in J = \{1, 3, \ldots, 2^{w-1} - 1\}$ **do** $Q_j \leftarrow \mathcal{O}$
3: $R \leftarrow P$
4: **for** i from 0 to ℓ **do**
5: $\quad$ **if** $k'_i \neq 0$ **then**
6: $\quad\quad Q_{|k'_i|} \leftarrow Q_{|k'_i|} + sign(k'_i)R$
7: $\quad$ **end if**
8: $\quad R \leftarrow R/2$
9: **end for**
10: **return** $Q \leftarrow \sum_{j \in J} jQ_j$

Parallel (double,halve)-and-add scalar multiplication. In the case of binary elliptic curve, the halve-and-add and double-and-add methods can be used in parallel in order to speed-up the computation of the scalar multiplication. The scalar k is recoded as

$$k = \underbrace{\left(\sum_{i=0}^{s} k'_i 2^i\right)}_{k'} + \underbrace{\left(\sum_{i=1}^{\ell-s} k''_i 2^{-i}\right)}_{k''}.$$

Then the computation can be split into two threads: one double-and-add thread computing $Q' = k'P$ and one halve-and-add thread computing $Q'' = k''P$, the result is obtained with a final addition $Q = Q' + Q''$. For further details on this method the reader may refer to (Taverne et al., 2011).

3 PROPOSED PARALLEL SCALAR MULTIPLICATION

We consider the case of curves where the parallelization based on GLV (Gallant et al., 2001) is not applicable. This is the case of the NIST curves B233, B409 and this is also the case of twisted Edwards curve Curve25519 (Bernstein, 2006).

3.1 Two-thread parallelization over $E(\mathbb{F}_p)$

The proposed approach to parallelize part of the computations involved in double-and-add scalar multiplication is the following: we split the scalar into two parts $k = k_1 + k_2 2^s$ with $k_1 < 2^s$ and then we divide the computations of $kP = k_1 P + 2^s k_2 P$ into two threads:

- Thread1 computes $Q_1 = k_1 P$ using a double-and-add approach of length s.
- Thread2 computes $2^s k_2 P$ by first performing $k_2 P$ using the double-and-add algorithm and then a sequence of doublings $2(k_2P), 4(k_2P), \ldots, 2^s(k_2P)$ and finally it outputs $Q_2 = 2^s(k_2P)$.

At the end, the two points Q_1 and Q_2 are added and then $Q = Q_2 + Q_1$ is output. This method is depicted in Fig. 1.

Complexity. The proposed parallelization is optimal if the computation load of the two threads are the same. This means that

$$\underbrace{\frac{s}{w+1}\mathrm{A} + s\mathrm{D}}_{\text{Thread1}} \cong \underbrace{\frac{\ell - s}{w+1}\mathrm{A} + \ell\mathrm{D}}_{\text{Thread2}}$$

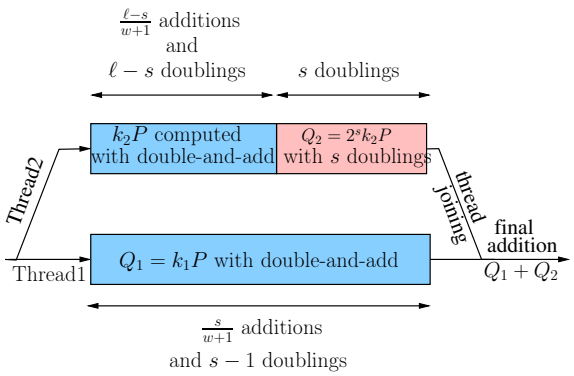

Figure 1 – Two-thread parallelization of double-and-add approach

where A and D represent an addition and a doubling, respectively. This implies that, for a balanced computation load, the split s of the scalar k have to be as follows:

$$s \cong \frac{A + (w+1)D}{2A + (w+1)D}\ell$$

In other words, the proposed parallelization reduces the computation time by a ratio of $\alpha = \frac{A+(w+1)D}{2A+(w+1)D}$. We use the complexity of the curve operations given in Table 1 to derive explicit values of the ratio $\alpha = \frac{A+(w+1)D}{2A+(w+1)D}$ for the three cases $w = 2, 3$ and 4. For the sake of simplicity, we assume that M_a and M_{a-1}, M_d are negligible compared to M and $S = 0.8M$. We report the resulting values of α in Table 3.

Table 3 – Estimated values for the ratio α

	Value of α		
	$w = 2$	$w = 3$	$w = 4$
Weierstrass	0.74	0.77	0.80
Twisted Edwards	0.75	0.79	0.81
Jacobi quartic	0.76	0.79	0.82

Table 3 shows that the timing of the execution is expected to be reduced by a factor of 25% for $w = 2$ and for the two other cases the improvement might be around 20%.

3.2 Optimized two-thread parallelization over $E(\mathbb{F}_p)$

We present in this subsection an optimized version of the approach of Subsection 3.1 when the curve is a twisted Edwards curves. A twisted Edwards curve can alternatively be set in a Montgomery form, and then we can use the efficiency of the doubling on a Montgomery curve: from Table 1 a doubling on a Montgomery curve requires only $2M + 2S$. Specifically, we propose to perform the sequence of s doublings of Thread2, in the parallelization of Subsection 3.1, with the Montgomery doubling. This results in the following optimized parallelization:

- Thread1 computes $Q_1 = k_1P$ using a double-and-add approach of length s on the Edward curve.
- Thread2 first computes the coordinates of P in the Montgomery curve. Then it computes $P_s = 2^sP$ on the Montgomery curve with a sequence of s doublings. Then it computes the coordinates of P_s in the Edwards curves. Finally it computes $k_2P_s = 2^sk_2P$ using the double-and-add algorithm on the twisted Edwards curve.

This modified two-thread parallelization induces a problem: we do not know the coordinate y_s of $P_s = 2^sP$ at the end of the sequence of doublings in the Montgomery curve. Indeed the Montgomery doubling formula computes only the x coordinate of a point (or, equivalently, X and Z projective coordinates). We can compute $\pm y_s$ by solving a quadratic equation in y given be the curve equation. But we do not know the sign of the solution of the quadratic equation, i.e., we are left with the two possible values $\pm y_s$. We propose to deal with this problem as follows:

- We pick an arbitrary sign for y_s and we let Thread2 compute $k_2 \cdot (\pm P_s)$.
- We use a right-to-left double-and-add scalar multiplication in Thread1. At some point, Thread1 computes $P_s = 2^sP$ and then at this point we know the correct sign for y_s.

When the two threads are joined, knowing the correct sign for y_s enables us to pick the correct value of $Q_s = 2^sk_2P$ and then we can compute $k_1P + 2^sk_2P$.

This proposed optimized two-thread parallelization is depicted in Fig. 2.

Complexity. We denote D_M the cost of a Montgomery doubling and A_E and D_E the respective costs of the addition and doubling on a twisted Edwards curve and A'_E the cost of a non mixed addition (involved in the right-to-left double-and-add). Then the work load of the two threads are equal if the following identity holds

$$\underbrace{\frac{s}{w+1}A'_E + sD_E}_{\text{Thread1}} \cong \underbrace{\frac{\ell - s}{w+1}A_E + (\ell - s)D_E + sD_M}_{\text{Thread2}}.$$

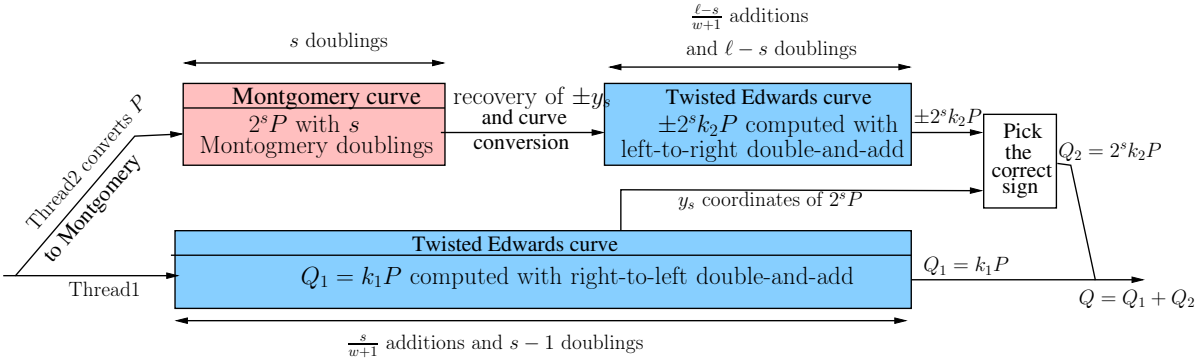

Figure 2 – Optimized two-thread parallelization of double-and-add approach

This implies that

$$s \cong \underbrace{\frac{A_E + (w+1)D_E}{A'_E + A_E + (w+1)(2D_E - D_M)}}_{\alpha} \ell.$$

With the complexity of the curve operations given in Table 1 we derive explicit values for the ratio α: for $w = 2$ we obtain $\alpha = 0.62$, for $w = 3$ we have $\alpha = 0.63$ and for $w = 4$ we have $\alpha = 0.64$.

In other words, with the proposed optimization, the theoretical saving is around 35% of the computation time compared to the non-parallelized version.

3.3 Three-thread parallelization over $E(\mathbb{F}_{2^m})$

Our goal in this subsection is to increase the level of parallelism of the two-thread (double,halve)-and-add scalar multiplication in $E(\mathbb{F}_{2^m})$ (reviewed in Subsection 2.3). To reach this goal, we recode and split the scalar k as follows:

$$k = \underbrace{\sum_{i=0}^{s'} k'_i 2^i}_{k'} + \underbrace{\sum_{i=1}^{s} k''_i 2^{-i}}_{k''} + 2^{-s} \underbrace{\left(\sum_{i=1}^{\ell-s'-s} k'''_i 2^{-i} \right)}_{k'''}$$

where k'_i, k''_i and k'''_i are in $\{\pm 1, \pm 3, \pm 5, \ldots, \pm 2^{w-2} - 3\}$.

We split the computation of the scalar multiplication into three threads:

- Thread1 computing $k'P$ using a double-and-add method of length s'.
- Thread2 computing $k''P$ using a halve-and-add method of length s.
- Thread3 computing $2^{-s}k'''P$ by performing a halve-and-add scalar multiplication followed by a sequence of s halvings to obtain $2^{-s}k'''P$.

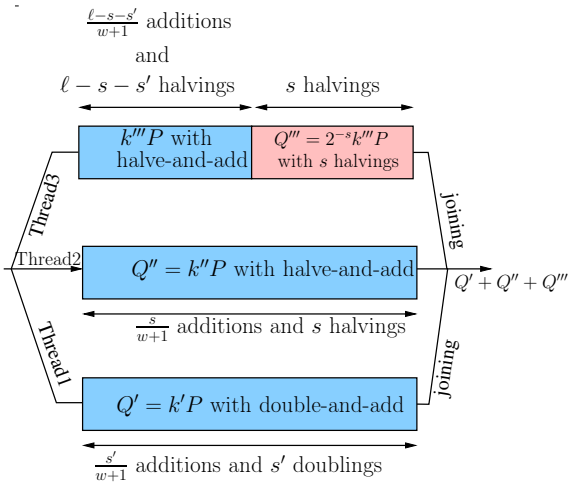

Figure 3 – Three-thread version of the (double,halve)-and-add approach

This approach is depicted in Fig. 3.

Complexity. Let us evaluate the reducing ratio of the computations in this case. The work load of the three threads are well-balanced if the following equations hold

$$\begin{aligned} \tfrac{s}{w+1}A + sH &\cong \tfrac{s'}{w+1}A + s'D, \\ \tfrac{s}{w+1}A + sH &\cong \tfrac{\ell-s'-s}{w+1}A + (\ell - s')H, \end{aligned}$$

where A represents an addition, D a doubling and H a halving on the curve $E(\mathbb{F}_{2^m})$. We solve the above equations and we obtain that the work load is well balanced when $s = \alpha\ell$ where

$$\alpha = \frac{(A + (w+1)H)(A + (w+1)D)}{((2A + (w+1)H)(A + (w+1)D) + (A + (w+1)H)^2)}.$$

The resulting values for the ratio α is given below in the cases $w = 2, 3$ and 4

- for $w = 2$ we have $\alpha \cong 0.46$,
- for $w = 3$ we have $\alpha \cong 0.37$,
- for $w = 4$ we have $\alpha \cong 0.37$.

4 IMPLEMENTATION RESULTS

In this section, we present our experimental results for the parallel approaches of in Section 3.

The platform used for the experimentations is an Optiplex 990 DELL running an Ubuntu 12.04. The processor is an Intel Core i7-2600 Sandy Bridge 3.4GHz which has four physical cores. Our code is written in C language and compiled with `gcc 4.6.3`. The timings are obtained with turbo mode and hyperthreading deactivated as recommended in (Bernstein and Lange, 2012).

4.1 Implementations for $E(\mathbb{F}_p)$

We consider the twisted Edwards curve Curve25519 introduced by Bernstein in (Bernstein, 2006) defined over the prime field $\mathbb{F}_p$, with $p = 2^{255} - 19$. For field operations, we reuse the publicly available code of Adam Langley in (Langley, 2008). In this code, a field element is stored in a array of five 64 bit words, each word containing 51 bits of the 255 bit field element. This allows a better management of carries in field addition and subtraction operations. The multiplications and squarings are performed with schoolbook method. Squaring is optimized with the usual trick which reduces the number of word multiplications. The reduction modulo $p = 2^{255} - 19$ consists in multiplying by 19 the 255 most significant bits and then adding the result to the lower 255 bits. For the inversion of a field element we use the extended Euclidean algorithm with the lower level function of the GMP library (gmp,). Curve operations simply follow the formulas provided in (hyp,) corresponding to inverted projective coordinates in twisted Edwards curves.

4.2 Implementations for $E(\mathbb{F}_{2^m})$

Our implementations deal with NIST curves B233 and B409 defined over the fields $\mathbb{F}_{2^{233}} = \mathbb{F}[x]/(x^{233}+x^{74}+1)$ and $\mathbb{F}_{2^{409}} = \mathbb{F}[x]/(x^{409}+x^{87}+1)$, respectively. For a field multiplication, we apply a small number of recursions of the Karatsuba algorithm which breaks the m bit polynomial multiplication into several 64 bit polynomial multiplications. Such 64 bit multiplication are computed with the `PCLMUL` instruction, available on Intel Core i7 processors. Due to the special form of the irreducible polynomials, the reduction is done with a small number of shifts and bitwise `XOR`s on 64 bit words. We compute the field inversion with the Itoh-Tsujii algorithm, that is a sequence of field multiplications and multisquarings performed with look-up table. For field squaring, square root and quadratic solver (needed in halvings), we also use a look-up table method, which is the fastest way according to our tests. For the curve operations, we use the projective lambda coordinates with the corresponding formulas provided in (Oliveira et al., 2014).

4.3 Timing results for the proposed parallel approaches

Table 4 reports the timings obtained for the three parallel approaches discussed in Section 3. We provide also the timings of the two-thread parallel (double,halve)-and-add approach with $w = 4$ for B233 and B409 and the timings of non-parallelized double-and-add approach with $w = 2, 3$ and 4 for $E(\mathbb{F}_p)$. For each parallel scalar multiplication we give the split value s (and s' for the three-thread case). Additionally we provide timings found in the literature over the same processor and for similar curves and fields.

Table 4 – Timings (in 10^3 clock-cycles (CC)) of parallel approaches over $E(\mathbb{F}_{2^m})$ ad $E(\mathbb{F}_p)$

	Curve	Method	NAF size w	$\frac{\#CC}{10^3}$	splits s	splits s'	nb of core
proposed	B233	three-thread	4	106	110	83	3
our code	B233	(db,hv)-&-add	4	104	98	–	2
Taverne et al.	B233	(db,hv)-&-add	4	100	-	-	2
Negre et al.	B233	(db,hv)-&-add	4	117	-	-	2
proposed	B409	three-thread	4	303	187	143	3
our code	B409	(db,hv)-&-add	4	338	175	–	2
Taverne et al.	B409	(db,hv)-&-add	4	349	-	-	2
Negre et al.	B409	(db,hv)-&-add	4	452	-	-	2
proposed	C25519	two-thread	2	186	185	-	2
proposed	C25519	opt-two-thd	2	180	168	-	2
our code	C25519	db-&-add	4	239	-	-	1
our code	C25519	db-&-add	3	219	-	-	1
our code	C25519	db-&-add	2	221	-	-	1
Langley$^{(\star)}$	C25519	Montg. ladder	-	229	-	-	1
Bernsetin	C25519	Montg. ladder	-	194	-	-	1
Hamburg	Mtg251	Montg. ladder	-	153	-	-	1

($\star$) Compiled and run on our platform

Concerning the curve B233, the proposed parallelization does not show any speed-up compared to the two-thread (double,halve)-and-add approach. This could be explained by the cost induced by the thread management. On the other hand, the approach is clearly effective for the

curve B409: it even shows a timing which is better than all timings found in the literature for (double,halve)-and-add approach (cf. Section 2).

In the case of Curve25519, the proposed optimizations behave as expected: the two-thread with $w = 2$ is 14% faster than the double-and-add with $w = 2$, the optimized-two-thread has a speed-up of 17%. The speed-up is smaller than the one expected provided by the value of α. But this might be due to the thread managements and to the penalty of the costly square-root computation in the case of the optimized-two-thread approach. Our approach compares favorably with the code of Langley and Bernstein, but it does not compare favorably with the timings of Hamburg. But the approach of Hamburg involves a smaller field and also a smaller key length.

5 CONCLUSION

We have presented in this paper parallel approaches to speed-up the scalar multiplication in $E(\mathbb{F}_{2^m})$ and $E(\mathbb{F}_p)$. The proposed parallelization split the scalar into two parts or three parts. Then each part of the scalar multiplication is performed in parallel, the upper part requiring an additional sequence of doublings or halving. These approaches have been implemented on an Intel Core i7 and the resulting timings shows that the proposed parallelizations is effective for curves for NIST curve B409 and for curve the twisted Edwards curve Curve25519 defined over $\mathbb{F}_p$ with $p = 2^{255} - 19$.

REFERENCES

Explicit formula database. http://www.hyperelliptic.org/EFD/.

The GNU Multiple Precision Arithmetic Library (GMP). http://gmplib.org/.

Bernstein, D. (2006). Curve25519: New Diffie-Hellman Speed Records. In *PKC 2006*, volume 3958 of *LNCS*, pages 207–228.

Bernstein, D. and Lange, T. (2012). eBACS: ECRYPT Benchmarking of Cryptograhic Systems. http://bench.cr.yp.to/. accessed May 25th, 14.

Gallant, R., Lambert, R., and Vanstone, S. (2001). Faster Point Multiplication on Elliptic Curves with Efficient Endomorphisms. In *CRYPTO 2001*, volume 2139 of *LNCS*, pages 190–200.

Hamburg, M. Fast and compact elliptic-curve cryptography. Cryptology ePrint Archive, Report 2012/309. http://eprint.iacr.org/.

Hankerson, D., Menezes, A., and Vanstone, S. (2004). *Guide to Elliptic Curve Cryptography*. Springer.

Hisil, H., Wong, K. K.-H., Carter, G., and Dawson, E. (2008). Twisted Edwards Curves Revisited. In *ASIACRYPT*, pages 326–343.

Kim, K. and Kim, S. A New Method for Speeding Up Arithmetic on Elliptic Curves over Binary Fields. Technical report. http://eprint.iacr.org/.

Knudsen, E. W. (1999). Elliptic Scalar Multiplication Using Point Halving. In *ASIACRYPT'99*, volume 1716 of *LNCS*, pages 135–149.

Langley, A. (2008). C25519 code. http://code.google.com/p/curve25519-donna/.

Longa, P. and Sica, F. (2014). Four-Dimensional Gallant-Lambert-Vanstone Scalar Multiplication. *J. Cryptology*, 27(2):248–283.

López, J. and Dahab, R. (1998). Improved Algorithms for Elliptic Curve Arithmetic in GF(2^n). In *SAC'98*, volume 1556 of *LNCS*, pages 201–212. Springer.

Montgomery, P. (1987). Speeding up the Pollard and elliptic curve methods of factorization. *Math. of Comp.*, 48:263–264.

Nègre, C. and Robert, J.-M. (2013). Impact of Optimized Field Operations ab, ac and $ab + cd$ in Scalar Multiplication over Binary Elliptic Curve. In *AFRICACRYPT*, pages 279–296.

Oliveira, T., López, J., Aranha, D., and Rodríguez-Henríquez, F. (2014). Two is the fastest prime: lambda coordinates for binary elliptic curves. *J. Crypt. Eng.*, 4(1):3–17.

P. Gallagher, D. D. and Furlani, C. (2009). Digital Signature Standard (DSS). In *FIPS Publications*, volume FIPS 186-3, page 93. NIST.

Taverne, J., Faz-Hernández, A., Aranha, D. F., Rodríguez-Henríquez, F., Hankerson, D., and López, J. (2011). Speeding Scalar Multiplication over Binary Elliptic Curves using the New Carry-Less Multiplication Instruction. *J. Crypt. Eng.*, 1(3):187–199.