Atomicity Improvement for Elliptic Curve Scalar Multiplication

Christophe Giraud[1] and Vincent Verneuil[2*]

[1] Oberthur Technologies,
4, allée du doyen Georges Brus, 33 600, Pessac, France.
c.giraud@oberthur.com
[2] Inside Contactless
41, Parc Club du Golf
13 856 Aix-en-Provence, Cedex 3, France.
vverneuil@insidefr.com

Abstract. In this paper we address the problem of protecting elliptic curve scalar multiplication implementations against side-channel analysis by using the atomicity principle. First of all we reexamine classical assumptions made by scalar multiplication designers and we point out that some of them are not relevant in the context of embedded devices. We then describe the state-of-the-art of atomic scalar multiplication and propose an atomic pattern improvement method. Compared to the most efficient atomic scalar multiplication published so far, our technique shows an average improvement of up to 10.6%.

Keywords: Elliptic Curves, Scalar Multiplication, Atomicity, Side-Channel Analysis.

1 Introduction

1.1 Preamble

We consider the problem of performing scalar multiplication on elliptic curves over $\mathbb{F}_p$ in the context of embedded devices such as smart cards. In this context, efficiency and side-channel resistance are of utmost importance. Concerning the achievement of the first requirement, numerous studies dealing with scalar multiplication efficiency have given rise to efficient algorithms including sliding-window and signed representation based methods [19].

Regarding the second requirement, side-channel attacks exploit the fact that physical leakages of a device (timing, power consumption, electromagnetic radiation, etc) depend on the operations performed and on the variables manipulated. These attacks can be divided into two groups: the *Simple Side-Channel Analysis* (SSCA) [25] which tries to observe a difference of behavior depending on the value of the secret key by using a single measurement, and the *Differential Side-Channel Analysis* (DSCA) [26] which exploits data value leakages by performing

* A part of this work has been done while at Oberthur Technologies.

statistical treatment over several hundreds of measurements to retrieve information on the secret key. Since 1996, many proposals have been made to protect scalar multiplication against these attacks [7, 12, 23]. Amongst them, *atomicity* introduced by Chevallier-Mames et al. in [9] is one of the most interesting methods to counteract SSCA. This countermeasure has been widely studied and Longa recently proposed an improvement for some scalar multiplication algorithms [27].

In this paper we present a new atomicity implementation for scalar multiplication, and we detail the atomicity improvement method we employed. This method can be applied to minimize atomicity implementation cost for sensitive algorithms with no security loss. In particular our method allows the implementation of atomic scalar multiplication in embedded devices in a more efficient way than any of the previous methods.

The rest of this paper is organized as follows. We finish this introduction by describing the scalar multiplication context which we are interested in and by mentioning an important observation on the cost of field additions. In Section 2 we recall some basics about Elliptic Curves Cryptography. In particular we present an efficient scalar multiplication algorithm introduced by Joye in 2008 [21]. Then we recall in Section 3 the principle of atomicity and we draw up a comparative chart of the efficiency of atomic scalar multiplication algorithms before this work. In Section 4, we propose an improvement of the original atomicity principle. In particular, we show that our method, applied to Joye's scalar multiplication, allows a substantial gain of time compared to the original atomicity principle. Finally, Section 5 concludes this paper.

1.2 Context of the study

We restrict the context of this paper to practical applications on embedded devices which yields the constraint of using standardized curves over $\mathbb{F}_p$[3]. As far as we know, NIST curves [17] and Brainpool curves [14,15] cover almost all curves currently used in the industry. We thus exclude from our scope Montgomery curves [32], Hessian curves [20], and Edwards curves[4] [16] which do not cover NIST neither Brainpool curves.

Considering that embedded devices – in particular smart cards – have very constrained resources (i.e. RAM and CPU), methods requiring heavy scalar treatment are discarded as well. In particular it is impossible to store scalar precomputations for some protocols such as ECDSA [1] where the scalar is randomly generated before each scalar multiplication. Most of the recent advances in this

[3] The curves over $\mathbb{F}_p$ are generally recommended for practical applications [33, 34].

[4] An elliptic curve over $\mathbb{F}_p$ is expressible in Edwards form only if it has a point of order 4 [6] and is expressible in twisted Edwards form only if it has three points of order 2 [4]. Since NIST and Brainpool curves have a cofactor of 1 there is not such equivalence. Nevertheless, for each of these curves, it is possible to find an extension field $\mathbb{F}_{p^q}$ over which the curve has a point of order 4 and is thus birationally equivalent to an Edwards curve. However the cost of a scalar multiplication over $\mathbb{F}_{p^q}$ is prohibitive in the context of embedded devices.

field cannot thus be taken into account: Double Base Number System [13, 31], multibase representation [28], Euclidean addition chains and Zeckendorf representation [30].

1.3 On the cost of field additions

In the literature, the cost of additions and subtractions over $\mathbb{F}_p$ is generally neglected compared to the cost of field multiplication. While this assumption is relevant in theory, we found out that these operations were not as insignificant as predicted for embedded devices. Smart cards for example have crypto-coprocessors in order to perform multi-precision arithmetic. These devices generally offer the following operations: addition, subtraction, multiplication, modular multiplication and sometimes modular squaring. Modular addition (respectively subtraction) must therefore be carried out by one classical addition (resp. subtraction) and one conditional subtraction (resp. addition) which should always be performed – i.e. the effective operation or a dummy one – for SSCA immunity. Moreover every operation carried out by the coprocessor requires a constant extra software processing δ to configure the coprocessor. As a result, the cost of field additions/subtractions is not negligible compared to field multiplications. Fig. 1 is an electromagnetic radiation measurement during the execution on a smart card of a 192-bit modular multiplication followed by a modular addition. Large amplitude blocks represent the 32-bit crypto-coprocessor activity while those with smaller amplitude are only CPU processing. In this case the time ratio between modular multiplication and modular addition is approximately 0.3.

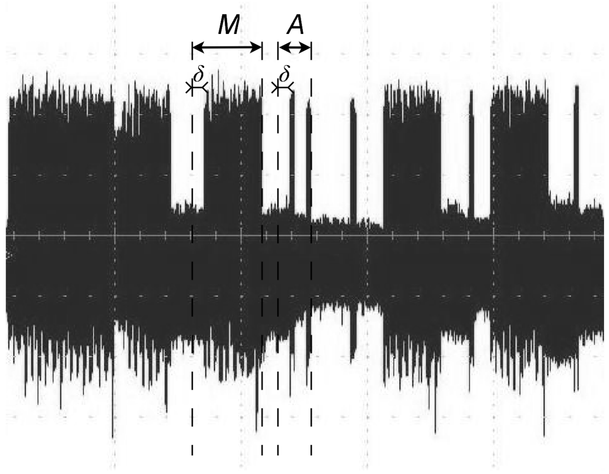

Fig. 1. Comparison between modular multiplication (M) and modular addition (A) timings.

From experiments on different smart cards provided with an arithmetic coprocessor, we estimated the average cost of modular additions/subtractions compared to modular multiplications. Our results are presented in Table 1 where A and M denote the cost of a field addition/subtraction and the cost of a field multiplication respectively. We observe that the average value of A/M for considered bit lengths is about 0.2.

Bit length	160	192	224	256	320	384	512	521
A/M	0.36	0.30	0.25	0.22	0.16	0.13	0.09	0.09

Table 1. Measured A/M ratio on smart cards with crypto-coprocessor for NIST and Brainpool ECC bit lengths.

Another useful field operation is negation in $\mathbb{F}_p$, i.e. the map $x \to -x$, which can be carried out by one non-modular subtraction $p - x$. The cost N of this operation is therefore half the cost of modular addition/subtraction and thus we fix $N/M = 0.5\,A/M$.

In the following sections we also consider the cost S of field squaring. The cost of a squaring compared to a multiplication depends on the functionalities of the corresponding crypto-coprocessor. When a dedicated squaring is available a commonly accepted value for S/M is 0.8 [8, 18] which is also corroborated by our experiments. Otherwise squarings must be carried out as multiplications and the ratio S/M is thus 1.

2 Elliptic Curves

In this section we recall some generalities about elliptic curves, and useful point representations. Then we present two efficient scalar multiplication algorithms.

Cryptology makes use of elliptic curves over binary fields $\mathbb{F}_{2^n}$ and large characteristic prime fields $\mathbb{F}_p$. In this study we focus on the latter case and hence assume $p > 3$.

2.1 Group law over $\mathbb{F}_p$

An elliptic curve $\mathcal{E}$ over $\mathbb{F}_p$, $p > 3$ can be defined as an algebraic curve of affine Weierstraß equation:

$$\mathcal{E} : y^2 = x^3 + ax + b \tag{1}$$

where $a, b \in \mathbb{F}_p$ and $4a^3 + 27b^2 \not\equiv 0 \pmod{p}$.

The set of points of $\mathcal{E}$ – i.e. the pairs $(x, y) \in {\mathbb{F}_p}^2$ satisfying (1) –, plus an extra point $\mathcal{O}$ called *point at infinity* form an abelian group where $\mathcal{O}$ is the neutral element. In the following, we present the corresponding law depending on the selected point representation.

Affine Coordinates. Under the group law a point $P = (x_1, y_1)$ lying on the elliptic curve $\mathcal{E}$ admits an opposite $-P = (x_1, -y_1)$.

The sum of $P = (x_1, y_1)$ and $Q = (x_2, y_2)$, with $P, Q \neq \mathcal{O}$ and $P \neq \pm Q$, is the point $P + Q = (x_3, y_3)$ such that:

$$\begin{cases} x_3 = ((y_2 - y_1)/(x_2 - x_1))^2 - x_1 - x_2 \\ y_3 = (x_1 - x_3)(y_2 - y_1)/(x_2 - x_1) - y_1 \end{cases} \tag{2}$$

The double of the point $P = (x_1, y_1)$, with $P \neq \mathcal{O}$ and $y_1 \neq 0$, is the point $2P = (x_2, y_2)$ as defined below, or $\mathcal{O}$ if $y_1 = 0$.

$$\begin{cases} x_2 = ((3{x_1}^2 + a)/(2y_1))^2 - 2x_1 \\ y_2 = (x_1 - x_2)(3{x_1}^2 + a)/(2y_1) - y_1 \end{cases} \tag{3}$$

Each point addition or point doubling requires an inversion in $\mathbb{F}_p$. This operation can be very time consuming and leads developers on embedded devices to use other kinds of representations with which point operations involve no field inversion. In the following part of this section, we detail two of them.

Jacobian Projective Coordinates. By denoting $x = X/Z^2$ and $y = Y/Z^3$, $Z \neq 0$, we obtain the Jacobian projective Weierstraß equation of the elliptic curve $\mathcal{E}$:

$$Y^2 = X^3 + aXZ^4 + bZ^6 \ , \tag{4}$$

where $a, b \in \mathbb{F}_p$ and $4a^3 + 27b^2 \neq 0$. Each point $P = (x, y)$ can be represented by its Jacobian projective coordinates $(q^2x : q^3y : q)$ with $q \in \mathbb{F}_p$. Conversely, every point $P = (X : Y : Z)$ different from $\mathcal{O}$ can be represented in affine coordinates by $(x, y) = (X/Z^2, Y/Z^3)$.

The opposite of a point $(X : Y : Z)$ is $(X : -Y : Z)$ and the point at infinity $\mathcal{O}$ is denoted by the unique point with $Z = 0$, $\mathcal{O} = (1 : 1 : 0)$.

The sum of $P = (X_1 : Y_1 : Z_1)$ and $Q = (X_2 : Y_2 : Z_2)$, with $P, Q \neq \mathcal{O}$ and $P \neq \pm Q$, is the point $P + Q = (X_3 : Y_3 : Z_3)$ such that:

$$\begin{cases} X_3 = F^2 - E^3 - 2AE^2 \\ Y_3 = F\left(AE^2 - X_3\right) - CE^3 \\ Z_3 = Z_1 Z_2 E \end{cases} \quad \text{with} \quad \begin{array}{l} A = X_1{Z_2}^2 \\ B = X_2{Z_1}^2 \\ C = Y_1{Z_2}^3 \\ D = Y_2{Z_1}^3 \\ E = B - A \\ F = D - C \end{array} \tag{5}$$

If P is given in affine coordinates – i.e. $Z_1 = 1$ – it is possible to save up one field squaring and four multiplications in (5). Such a case is referred to as *mixed affine-Jacobian addition.* On the other hand if P has to be added several times, storing ${Z_1}^2$ and ${Z_1}^3$ saves one squaring and one multiplication in all following additions involving P. This latter case is referred to as *readdition.*

The double of the point $P = (X_1 : Y_1 : Z_1)$ is the point $2P = (X_2 : Y_2 : Z_2)$ such that:

$$\begin{cases} X_2 = C^2 - 2B \\ Y_2 = C\left(B - X_2\right) - 2A^2 \\ Z_2 = 2Y_1 Z_1 \end{cases} \quad \text{with} \quad \begin{array}{l} A = 2{Y_1}^2 \\ B = 2AX_1 \\ C = 3{X_1}^2 + a{Z_1}^4 \end{array} \tag{6}$$

When curve parameter a is -3, doubling can be carried out taking $C = 3\left(X_1 + {Z_1}^2\right)\left(X_1 - {Z_1}^2\right)$ which saves two squarings in (6). We denote this operation by *fast doubling*.

Adding up field operations yields $12M + 4S + 7A$ for general addition, $11M + 3S + 7A$ for readdition, $8M + 3S + 7A$ for mixed addition, $4M + 6S + 11A$ for general doubling formula and $4M + 4S + 12A$ for fast doubling.

Modified Jacobian Projective Coordinates. This representation, introduced in [11], is derived from the Jacobian projective representation to which a fourth coordinate is added for computation convenience. In this representation, a point on the curve $\mathcal{E}$ is thus represented by $(X : Y : Z : aZ^4)$, where $(X : Y : Z)$ stands for the Jacobian representation.

Modified Jacobian projective coordinates provide a particularly efficient doubling formula. Indeed, the double of a point $P = (X_1 : Y_1 : Z_1 : W_1)$ is given by $2P = (X_2 : Y_2 : Z_2 : W_2)$ such that:

$$\begin{cases} X_2 = A^2 - 2C \\ Y_2 = A\left(C - X_2\right) - D \\ Z_2 = 2Y_1 Z_1 \\ W_2 = 2DW_1 \end{cases} \quad \text{with} \quad \begin{array}{l} A = 3{X_1}^2 + W_1 \\ B = 2{Y_1}^2 \\ C = 2BX_1 \\ D = 2B^2 \end{array} \tag{7}$$

Doubling hence requires only $4M + 4S + 12A$ for all a values. On the other hand, addition is less efficient compared to Jacobian projective representation: by applying formula (5), we need to compute the fourth coordinate which is required in point doubling, adding an overhead of $1M + 2S$ [21].

On S–M trade-offs. Addition and doubling formulas presented above are voluntarily not state-of-the-art, see [5]. Indeed, recent advances have provided Jacobian formulas where some field multiplications have been traded for faster field squarings [27, Sec. 4.1]. These advances have been achieved by using the so-called *S–M trade-off* principle which is based on the fact that computing ab when a^2 and b^2 are known can be done as $2ab = (a+b)^2 - a^2 - b^2$. This allows a squaring to replace a multiplication since the additional factor 2 can be handled by considering the representative of the Jacobian coordinates equivalence class $(X : Y : Z) = (2^2X : 2^3Y : 2Z)$.

Nevertheless such trade-offs not only replace field multiplications by field squarings but also add field additions. In the previous example at least 3 extra additions have to be performed, thus taking $S/M = 0.8$ implies that the trade-off is profitable only if $A/M < 0.067$ which is never the case with devices considered using standardized curves as seen in Section 1.3. These new formulas are thus not relevant in the context of embedded devices.

2.2 Scalar Multiplication

Generalities. The operation consisting in calculating the multiple of a point $k \cdot P = P + P + \cdots + P$ (k times) is called *scalar multiplication* and the integer k is thus referred to as the *scalar*.

Scalar multiplication is used in ECDSA signature [1] and ECDH key agreement [2] protocols. Implementing such protocols on embedded devices requires particular care from both the efficiency and the security points of view. Indeed scalar multiplication turns out to be the most time consuming part of the aforementioned protocols, and since it uses secret values as scalars, side-channel analysis endangers the security of those protocols.

Most of the scalar multiplication algorithms published so far are derived from the traditional *double and add* algorithm. This algorithm can scan the binary representation of the scalar in both directions which leads to the *left-to-right* and *right-to-left* variants. The former is generally preferred over the latter since it saves one point in memory.

Moreover since computing the opposite of a point P on an elliptic curve is virtually free, the most efficient methods for scalar multiplication use signed digit representations such as the Non-Adjacent Form (NAF) [3]. Under the NAF representation, an n-bit scalar has an average Hamming weight of $n/3$ which implies that one point doubling is performed every bit of scalar and one point addition is performed every three bits.

In the two next subsections, we present a left-to-right and a right-to-left NAF scalar multiplication algorithms.

Left To Right Binary NAF Scalar Multiplication. Alg. 1 presents the classical NAF scalar multiplication algorithm.

Algorithm 1 Left-to-right binary NAF scalar multiplication [19]

```
INPUTS : $P = (X_1 : Y_1 : Z_1) \in \mathcal{E}(\mathbb{F}_p)$, $k = (k_{l-1} \ldots k_1 k_0)_{\mathrm{NAF}}$
OUTPUT : $k \cdot P$

1. $(X_2 : Y_2 : Z_2) \leftarrow (X_1 : Y_1 : Z_1)$
2. $i \leftarrow l - 2$
3. while $i \geq 0$ do
     $(X_2 : Y_2 : Z_2) \leftarrow 2 \cdot (X_2 : Y_2 : Z_2)$
     if $k_i = 1$ then
         $(X_2 : Y_2 : Z_2) \leftarrow (X_2 : Y_2 : Z_2) + (X_1 : Y_1 : Z_1)$
     if $k_i = -1$ then
         $(X_2 : Y_2 : Z_2) \leftarrow (X_2 : Y_2 : Z_2) - (X_1 : Y_1 : Z_1)$
     $i \leftarrow i - 1$
4. return $(X_2 : Y_2 : Z_2)$
```

Point doubling can be done in Alg. 1 using general Jacobian doubling formula or fast doubling formula. Since NIST curves fulfill $a = -3$ and each Brainpool curve is provided with an isomorphism to a curve with $a = -3$, we thus assume that fast doubling is always possible. Point addition can be performed using mixed addition formula if input points are given in affine coordinates or by using readdition formula otherwise.

It is possible to reduce the number of point additions by using window techniques[5] which need the precomputation of some first odd multiples of the point P. Table 2 recalls the number of point additions per bit of scalar when having from 0 (simple NAF) to 4 precomputed points. More than 4 points allows even better results but seems not practical in the context of constrained memory.

Nb. of precomp. points	0	1	2	3	4
Precomputed points	–	$3P$	$3P, 5P$	$3P, 5P, 7P$	$3P, \ldots, 9P$
Point additions / bit	$1/3 \approx 0.33$	$1/4 = 0.25$	$2/9 \approx 0.22$	$1/5 = 0.20$	$4/21 \approx 0.19$

Table 2. Average number of point additions per bit of scalar using window NAF algorithms.

Right To Left Binary NAF Mixed Coordinates Multiplication. We recall here a very efficient algorithm performing right-to-left NAF scalar multiplication. Indeed this algorithm uses the fast modified Jacobian doubling formula which works for all curves – i.e. for all a – without needing the slow modified Jacobian addition.

This is achieved by reusing the idea of *mixed coordinates* scalar multiplication (i.e. two coordinate systems are used simultaneously) introduced by Cohen, Ono and Miyaji in [11]. The aim of this approach is to make the best use of two coordinates systems by processing some operations with one system and others with the second. Joye proposed in [21] to perform additions by using Jacobian coordinates, doublings – referred to as $*$ – by using modified Jacobian coordinates, and to compute the NAF representation of the scalar on-the-fly, cf. Alg. 2[6].

In the same way as their left-to-right counterpart benefits from precomputed points, right-to-left algorithms can be enhanced using window techniques if extra memory is available [22, 35]. In this case precomputations are replaced by postcomputations the cost of which is negligible for the considered window sizes and bit lengths.

In [21] the author suggests protecting Alg. 2 against SSCA by using the so-called atomicity principle. We recall in the next section the principle of this SSCA countermeasure.

[5] By *window techniques* we mean the sliding window NAF and the Window NAF$_w$ algorithms, see [19] for more details.

[6] In Alg. 2, Jacobian addition is assumed to handle the special cases $P = \pm Q$, $P = \mathcal{O}$, $Q = \mathcal{O}$ as discussed in [21].

Algorithm 2 Right-to-left binary NAF mixed coordinates multiplication [21]

INPUTS : $P = (X_1 : Y_1 : Z_1) \in \mathcal{E}(\mathbb{F}_p)$, k
OUTPUT : $k \cdot P$

```
1. $(X_2 : Y_2 : Z_2) \leftarrow (1 : 1 : 0)$
2. $(R_1 : R_2 : R_3 : R_4) \leftarrow (X_1 : Y_1 : Z_1 : aZ_1{}^4)$
3. while $k > 1$ do
     if $k \equiv 1 \mod 2$ then
         $u \leftarrow 2 - (k \mod 4)$
         $k \leftarrow k - u$
         if $u = 1$ then
             $(X_2 : Y_2 : Z_2) \leftarrow (X_2 : Y_2 : Z_2) + (R_1 : R_2 : R_3)$
         else
             $(X_2 : Y_2 : Z_2) \leftarrow (X_2 : Y_2 : Z_2) + (R_1 : -R_2 : R_3)$
     $k \leftarrow k/2$
     $(R_1 : R_2 : R_3 : R_4) \leftarrow 2 * (R_1 : R_2 : R_3 : R_4)$
4. $(X_2 : Y_2 : Z_2) \leftarrow (X_2 : Y_2 : Z_2) + (R_1 : R_2 : R_3)$
5. return $(X_2 : Y_2 : Z_2)$
```

3 Atomicity

In this section we recall the principle of atomicity and its application to scalar multiplication. Other countermeasures exist in order to thwart SSCA such as regular algorithms [12,22,24] and unified formulas [7,16]. However regular algorithms require costly extra curve operations, and unified formulas for Weierstrass curves over $\mathbb{F}_p$ – only known in the affine and homogeneous coordinate systems, see [7] – are also very costly. Therefore atomicity turns out to be more efficient in the context of embedded devices. It is thus natural to compare the efficiency of the two scalar multiplication methods presented in Section 2.2 protected by atomicity.

We recall in the following how atomicity is generally implemented on elliptic curves cryptography, for a complete atomicity principle description see [9].

3.1 State-of-the-art

The atomicity principle has been introduced in [10]. This countermeasure consists in rewriting all the operations carried out through an algorithm into a sequence of identical *atomic patterns*. The purpose of this method is to defeat SSCA since an attacker has nothing to learn from an uniform succession of identical patterns.

In the case of scalar multiplications, a succession of point doublings and point additions is performed. Each of these operations being composed of field operations, the execution of a scalar multiplication can be seen as a succession of field operations. The atomicity consists here in rewriting the succession of field operations into a sequence of identical atomic patterns. The atomic pattern (1) proposed in [9] is composed of the following field operations: a multiplication, two additions and a negation. R_i's denote the crypto-coprocessor registers.

$$(1)\left[\begin{array}{l} R_1 \leftarrow R_2 \cdot R_3 \\ R_4 \leftarrow R_5 + R_6 \\ R_7 \leftarrow -R_8 \\ R_9 \leftarrow R_{10} + R_{11} \end{array}\right.$$

This choice relies on the observation that during the execution of point additions and point doublings, no more than two additions and one negation are required between two multiplications. Atomicity consists then of writing point addition and point doubling as sequences of this pattern – as many as there are field multiplications (including squarings).

Therefore this countermeasure induces two kinds of costs:

- Field squarings have to be performed as field multiplications. Then this approach is costly on embedded devices with dedicated hardware offering modular squaring operation, i.e. when $S/M < 1$.
- Dummy additions and negations are added. Their cost is generally negligible from a theoretical point of view but, as shown in Section 1.3, the cost of such operations must be taken into account in the context of embedded devices.

To reduce these costs, Longa proposed in his PhD thesis [27, Chap. 5] the two following atomic patterns in the context of Jacobian coordinates:

$$(2)\left[\begin{array}{ll} R_1 & \leftarrow R_2 \cdot R_3 \\ R_4 & \leftarrow -R_5 \\ R_6 & \leftarrow R_7 + R_8 \\ R_9 & \leftarrow R_{10} \cdot R_{11} \\ R_{12} & \leftarrow -R_{13} \\ R_{14} & \leftarrow R_{15} + R_{16} \\ R_{17} & \leftarrow R_{18} + R_{19} \end{array}\right. \qquad (3)\left[\begin{array}{ll} R_1 & \leftarrow {R_2}^2 \\ R_3 & \leftarrow -R_4 \\ R_5 & \leftarrow R_6 + R_7 \\ R_8 & \leftarrow R_9 \cdot R_{10} \\ R_{11} & \leftarrow -R_{12} \\ R_{13} & \leftarrow R_{14} + R_{15} \\ R_{16} & \leftarrow R_{17} + R_{18} \end{array}\right.$$

Compared with atomic pattern (1), these two patterns slightly reduce the number of field additions (gain of one addition every two multiplications). Moreover, atomic pattern (3) takes advantage of the squaring operation by replacing one multiplication out of two by a squaring.

In [27, Appendices] Longa expresses mixed affine-Jacobian addition formula as 6 atomic patterns (2) or (3) and fast doubling formula as 4 atomic patterns (2) or (3). It allows to perform an efficient left-to-right scalar multiplication using fast doubling and mixed affine-Jacobian addition protected with atomic patterns (2) or (3).

3.2 Atomic left-to-right scalar multiplication

We detail in the following why the Longa's left-to-right scalar multiplication using fast doubling and mixed affine-Jacobian addition is not compatible with our security constraints.

Defeating DSCA[7] requires the randomization of input point coordinates. This can be achieved by two means: projective coordinates randomization [12] and random curve isomorphism [23]. The first one allows to use the fast point doubling formula but prevents the use of mixed additions since input points $P, 3P, \ldots$ have their Z coordinate randomized. On the other hand the random curve isomorphism keeps input points in affine coordinates but randomizes a which thus imposes the use of the general doubling formula instead of the fast one.

Since Longa didn't investigate general doubling nor readdition, we present in Appendix A.1 the formulas to perform the former by using 5 atomic patterns (2) or (3) and in Appendix A.2 the formulas to perform the latter by using 7 atomic patterns (2). It seems very unlikely that one can express readdition using atomic pattern (3): since state-of-the-art readdition formula using the S–M trade-off requires 10 multiplications and 4 squarings, 3 other multiplications would have to be traded for squarings.

Therefore secure left-to-right scalar multiplication can be achieved either by using atomic pattern (2) and projective coordinates randomization which would involve fast doublings and readditions or by using atomic pattern (3) and random curve isomorphism which would involve general doublings and mixed additions.

3.3 Atomic right-to-left mixed scalar multiplication

As suggested in [21] we protected Alg. 2 with atomicity. Since Longa's atomic patterns have not been designed for modified Jacobian doubling, we applied atomic pattern (1) to protect Alg. 2.

The decomposition of general Jacobian addition formula in 16 atomic patterns (1) is given in [9]. Since we haven't found it in the literature, we present in Appendix A.3 a decomposition of modified Jacobian doubling formula in 8 atomic patterns (1).

Projective coordinates randomization and random curve isomorphism countermeasures can both be applied to this solution.

3.4 Atomic Scalar Multiplication Algorithms Comparison

We compare in Table 3 the three previously proposed atomically protected algorithms. As discussed in Section 1.3 we fix $A/M = 0.2$ and $N/M = 0.1$. Costs are given as the average number of field multiplications per bit of scalar. Each cost is estimated for devices providing dedicated modular squaring – i.e. $S/M = 0.8$ – or not – i.e. $S/M = 1$. If extra memory is available, precomputations or postcomputations are respectively used to speed up left-to-right and right-to-left scalar multiplications. The pre/postcomputation cost is here not taken into account but is constant for every row of the chart.

[7] We include in DSCA the Template Attack on ECDSA from [29].

Nb. of extra points	S/M	Left-to-right with (2)	Left-to-right with (3)	Right-to-left with (1)
0	0.8	17.7	18.2	20.0
	1	17.7	19.6	20.0
1	0.8	16.1	16.9	18.0
	1	16.1	18.2	18.0
2	0.8	15.6	16.5	17.3
	1	15.6	17.7	17.3
3	0.8	15.1	16.1	16.8
	1	15.1	17.4	16.8
4	0.8	14.9	16.0	16.6
	1	14.9	17.2	16.6

Table 3. Cost estimation in field multiplications per bit of the 3 atomically protected scalar multiplication algorithms with $A/M = 0.2$.

It appears that in our context atomic left-to-right scalar multiplication using atomic pattern (2) with fast doubling and readditions is the fastest solution and is, on average for the 10 rows of Table 3, 10.5 % faster than atomic right-to-left mixed scalar multiplication using atomic pattern (1).

In the next section we present our contribution that aims at minimizing the atomicity cost by optimizing the atomic pattern. Then we apply it on the right-to-left mixed scalar multiplication algorithm since efficient patterns are already known for the two left-to-right variants.

4 Atomic Pattern Improvement

We propose here a twofold atomicity improvement method: firstly, we take advantage of the fact that a squaring can be faster than a multiplication. Secondly, we reduce the number of additions and negations used in atomic patterns in order to increase the efficiency of scalar multiplication.

4.1 First Part: Atomic Pattern Extension

As explained previously, our first idea is to reduce the efficiency loss due to field squarings turned into multiplications.

Method Presentation. Let O_1 and O_2 be two atomically written operations (point addition and doubling in our case) such that they require m and n atomic patterns respectively. Let us assume that a sub-operation o_1 from the atomic pattern (field multiplication in our case) could sometimes be replaced by another preferred sub-operation o_2 (such as field squaring). Let us eventually assume that O_1 requires at least m' sub-operations o_1 (along with $m - m'$ sub-operations o_2) and O_2 requires at least n' sub-operations o_1 (along with $n - n'$ sub-operations o_2).

Then, if $d = \gcd(m, n) > 1$, let e represents the greatest positive integer satisfying:

$$e \cdot \frac{m}{d} \leq m - m' \quad \text{and} \quad e \cdot \frac{n}{d} \leq n - n' \ . \tag{8}$$

Since 0 is obviously a solution, it is certain that e is defined. If $e > 0$ we can now apply the following method. Let a new pattern be defined with $d - e$ original atomic patterns followed by e atomic patterns with o_2 replacing o_1 – the order can be modified at convenience.

It is now possible to express operations O_1 and O_2 with m/d and n/d new patterns respectively. Using the new pattern in O_1 (resp. O_2) instead of the old one allows replacing $e \cdot m/d$ (resp. $e \cdot n/d$) sub-operations o_1 by o_2.

Application to Mixed Coordinates Scalar Multiplication. Applying this method to Alg. 2 yields the following result: O_1 being the Jacobian projective addition, O_2 the modified Jacobian projective doubling, o_1 the field multiplication and o_2 the field squaring, then m=16, m'=11, $n = 8$, $n' = 3$, $d = 8$ and $e = 2$. Therefore we define a new temporary atomic pattern composed of 8 patterns (1) where 2 multiplications are replaced by squarings. We thus have one fourth of the field multiplications carried out as field squarings. This extended pattern would have to be repeated twice for an addition and once for a doubling.

We applied this new approach in Fig. 2 where atomic general Jacobian addition and modified Jacobian doubling are rewritten in order to take advantage of the squarings. We denote by $\star$ the dummy field additions and negations that must be added to complete atomic patterns.

4.2 Second Part: Atomic Pattern Cleaning-Up

In a second step we aim at reducing the number of dummy field operations. In Fig. 2, we identified by $\star$ the operations that are never used in Add.1, Add.2 and Dbl. These field operations may then be removed saving up 5 field additions and 3 field negations per pattern occurrence.

However, we found out that field operations could be rearranged in order to maximize the number of rows over the three columns composed of dummy operations only. We then merge negations and additions into subtractions when possible. This improvement is depicted in Fig. 3.

$$\text{Add. 1}\left[\begin{array}{l}
R_1 \leftarrow {Z_2}^2 \\
\star \\
{\color{gray}\star} \\
{\color{gray}\star} \\
R_2 \leftarrow X_1 \cdot R_1 \\
\star \\
{\color{gray}\star} \\
{\color{gray}\star} \\
R_1 \leftarrow R_1 \cdot Z_2 \\
\star \\
{\color{gray}\star} \\
{\color{gray}\star} \\
R_3 \leftarrow Y_1 \cdot R_1 \\
\star \\
\star \\
\star \\
R_1 \leftarrow {Z_1}^2 \\
{\color{gray}\star} \\
\star \\
{\color{gray}\star} \\
R_4 \leftarrow R_1 \cdot X_2 \\
\star \\
R_4 \leftarrow -R_4 \\
R_4 \leftarrow R_2 + R_4 \\
R_1 \leftarrow Z_1 \cdot R_1 \\
\star \\
\star \\
\star \\
R_1 \leftarrow R_1 \cdot Y_2 \\
\star \\
R_1 \leftarrow -R_1 \\
R_1 \leftarrow R_3 + R_1
\end{array}\right.
\quad
\text{Add. 2}\left[\begin{array}{l}
R_6 \leftarrow {R_4}^2 \\
\star \\
{\color{gray}\star} \\
{\color{gray}\star} \\
R_5 \leftarrow Z_1 \cdot Z_2 \\
\star \\
{\color{gray}\star} \\
{\color{gray}\star} \\
Z_3 \leftarrow R_5 \cdot R_4 \\
\star \\
{\color{gray}\star} \\
{\color{gray}\star} \\
R_2 \leftarrow R_2 \cdot R_6 \\
\star \\
R_1 \leftarrow -R_1 \\
\star \\
R_5 \leftarrow {R_1}^2 \\
{\color{gray}\star} \\
R_3 \leftarrow -R_3 \\
{\color{gray}\star} \\
R_4 \leftarrow R_4 \cdot R_6 \\
R_6 \leftarrow R_5 + R_4 \\
R_2 \leftarrow -R_2 \\
R_6 \leftarrow R_6 + R_2 \\
R_3 \leftarrow R_3 \cdot R_4 \\
X_3 \leftarrow R_2 + R_6 \\
\star \\
R_2 \leftarrow X_3 + R_2 \\
R_1 \leftarrow R_1 \cdot R_2 \\
Y_3 \leftarrow R_3 + R_1 \\
\star \\
\star
\end{array}\right.
\quad
\text{Dbl.}\left[\begin{array}{l}
R_1 \leftarrow {X_1}^2 \\
R_2 \leftarrow Y_1 + Y_1 \\
{\color{gray}\star} \\
{\color{gray}\star} \\
Z_2 \leftarrow R_2 \cdot Z_1 \\
R_4 \leftarrow R_1 + R_1 \\
{\color{gray}\star} \\
{\color{gray}\star} \\
R_3 \leftarrow R_2 \cdot Y_1 \\
R_6 \leftarrow R_3 + R_3 \\
{\color{gray}\star} \\
{\color{gray}\star} \\
R_2 \leftarrow R_6 \cdot R_3 \\
R_1 \leftarrow R_4 + R_1 \\
\star \\
R_1 \leftarrow R_1 + W_1 \\
R_3 \leftarrow {R_1}^2 \\
{\color{gray}\star} \\
\star \\
{\color{gray}\star} \\
R_4 \leftarrow R_6 \cdot X_1 \\
R_5 \leftarrow W_1 + W_1 \\
R_4 \leftarrow -R_4 \\
R_3 \leftarrow R_3 + R_4 \\
W_2 \leftarrow R_2 \cdot R_5 \\
X_2 \leftarrow R_3 + R_4 \\
R_2 \leftarrow -R_2 \\
R_6 \leftarrow R_4 + X_2 \\
R_4 \leftarrow R_6 \cdot R_1 \\
\star \\
R_4 \leftarrow -R_4 \\
Y_2 \leftarrow R_4 + R_2
\end{array}\right.$$

Fig. 2. Extended atomic pattern applied to Jacobian projective addition and modified Jacobian projective doubling.

$$
\text{Add. 1}\left[\begin{array}{l}
R_1 \leftarrow {Z_2}^2\\
\star\\
{\color{gray}\star}\\
{\color{gray}\star}\\
R_2 \leftarrow Y_1 \cdot Z_2\\
\star\\
{\color{gray}\star}\\
{\color{gray}\star}\\
R_5 \leftarrow Y_2 \cdot Z_1\\
\star\\
{\color{gray}\star}\\
{\color{gray}\star}\\
R_3 \leftarrow R_1 \cdot R_2\\
\star\\
{\color{gray}\star}\\
\star\\
R_4 \leftarrow {Z_1}^2\\
{\color{gray}\star}\\
{\color{gray}\star}\\
{\color{gray}\star}\\
R_2 \leftarrow R_5 \cdot R_4\\
\star\\
{\color{gray}\star}\\
R_2 \leftarrow R_2 - R_3\\
R_5 \leftarrow R_1 \cdot X_1\\
\star\\
{\color{gray}\star}\\
\star\\
R_6 \leftarrow X_2 \cdot R_4\\
{\color{gray}\star}\\
{\color{gray}\star}\\
R_6 \leftarrow R_6 - R_5
\end{array}\right.
\qquad
\text{Add. 2}\left[\begin{array}{l}
R_1 \leftarrow {R_6}^2\\
\star\\
{\color{gray}\star}\\
{\color{gray}\star}\\
R_4 \leftarrow R_5 \cdot R_1\\
\star\\
{\color{gray}\star}\\
{\color{gray}\star}\\
R_5 \leftarrow R_1 \cdot R_6\\
\star\\
{\color{gray}\star}\\
{\color{gray}\star}\\
R_1 \leftarrow Z_1 \cdot R_6\\
\star\\
{\color{gray}\star}\\
\star\\
R_6 \leftarrow {R_2}^2\\
{\color{gray}\star}\\
{\color{gray}\star}\\
{\color{gray}\star}\\
Z_3 \leftarrow R_1 \cdot Z_2\\
R_1 \leftarrow R_4 + R_4\\
{\color{gray}\star}\\
R_6 \leftarrow R_6 - R_1\\
R_1 \leftarrow R_5 \cdot R_3\\
X_3 \leftarrow R_6 - R_5\\
{\color{gray}\star}\\
R_4 \leftarrow R_4 - X_3\\
R_3 \leftarrow R_4 \cdot R_2\\
{\color{gray}\star}\\
{\color{gray}\star}\\
Y_3 \leftarrow R_3 - R_1
\end{array}\right.
\qquad
\text{Dbl.}\left[\begin{array}{l}
R_1 \leftarrow {X_1}^2\\
R_2 \leftarrow Y_1 + Y_1\\
{\color{gray}\star}\\
{\color{gray}\star}\\
Z_2 \leftarrow R_2 \cdot Z_1\\
R_4 \leftarrow R_1 + R_1\\
{\color{gray}\star}\\
{\color{gray}\star}\\
R_3 \leftarrow R_2 \cdot Y_1\\
R_6 \leftarrow R_3 + R_3\\
{\color{gray}\star}\\
{\color{gray}\star}\\
R_2 \leftarrow R_6 \cdot R_3\\
R_1 \leftarrow R_4 + R_1\\
{\color{gray}\star}\\
R_1 \leftarrow R_1 + W_1\\
R_3 \leftarrow {R_1}^2\\
{\color{gray}\star}\\
{\color{gray}\star}\\
{\color{gray}\star}\\
R_4 \leftarrow R_6 \cdot X_1\\
R_5 \leftarrow W_1 + W_1\\
{\color{gray}\star}\\
R_3 \leftarrow R_3 - R_4\\
W_2 \leftarrow R_2 \cdot R_5\\
X_2 \leftarrow R_3 - R_4\\
{\color{gray}\star}\\
R_6 \leftarrow R_4 - X_2\\
R_4 \leftarrow R_6 \cdot R_1\\
{\color{gray}\star}\\
{\color{gray}\star}\\
Y_2 \leftarrow R_4 - R_2
\end{array}\right.
$$

Fig. 3. Improved arrangement of field operations in extended atomic pattern from Fig. 2.

This final optimization now allows us to save up 6 field additions and to remove the 8 field negations per pattern occurrence. One may note that no more dummy operation remains in modified Jacobian doubling. We thus believe that our resulting atomic pattern (4) is optimal for this operation:

$$(4)\ \left[\begin{array}{l} R_1 \leftarrow {R_2}^2 \\ R_3 \leftarrow R_4 + R_5 \\ R_6 \leftarrow R_7 \cdot R_8 \\ R_9 \leftarrow R_{10} + R_{11} \\ R_{12} \leftarrow R_{13} \cdot R_{14} \\ R_{15} \leftarrow R_{16} + R_{17} \\ R_{18} \leftarrow R_{19} \cdot R_{20} \\ R_{21} \leftarrow R_{22} + R_{23} \\ R_{24} \leftarrow R_{25} + R_{26} \\ R_{27} \leftarrow {R_{28}}^2 \\ R_{29} \leftarrow R_{30} \cdot R_{31} \\ R_{32} \leftarrow R_{33} + R_{34} \\ R_{35} \leftarrow R_{36} - R_{37} \\ R_{38} \leftarrow R_{39} \cdot R_{40} \\ R_{41} \leftarrow R_{42} - R_{43} \\ R_{44} \leftarrow R_{45} - R_{46} \\ R_{47} \leftarrow R_{48} \cdot R_{49} \\ R_{50} \leftarrow R_{51} - R_{52} \end{array}\right.$$

4.3 Theoretical gain

In Table 4 we present the cost of right-to-left mixed scalar multiplication protected with atomic pattern (4). We also draw up in this chart the gains obtained over left-to-right and right-to-left algorithms protected with atomic patterns (2) and (1) respectively.

Nb. of extra points	S/M	Right-to-left with (4)	Gain over l.-to-r. with (2)	Gain over r.-to-l. with (1)
0	0.8	16.0 M	9.6 %	20.0 %
	1	16.7 M	5.6 %	16.5 %
1	0.8	14.4 M	10.6 %	20.0 %
	1	15.0 M	6.8 %	16.7 %
2	0.8	13.9 M	10.9 %	19.7 %
	1	14.4 M	7.7 %	16.8 %
3	0.8	13.4 M	11.3 %	20.2 %
	1	14.0 M	7.3 %	16.7 %
4	0.8	13.3 M	10.7 %	19.9 %
	1	13.8 M	7.4 %	16.9 %

Table 4. Costs estimation in field multiplications per bit of Alg. 2 protected with improved pattern (4) and comparison with two others methods presented in Table 3 assuming $A/M = 0.2$.

Due to our new atomic pattern (4), right-to-left mixed scalar multiplication turns out to be the fastest among these solutions in every cases. The average speed-up over pattern (1) is 18.3 % and the average gain over left-to-right scalar multiplication protected with atomic pattern (2) is 10.6 % if dedicated squaring is available or 7.0 % otherwise.

4.4 Experimental Results

We have implemented Alg. 2 – without any window method – protected with the atomic pattern (1) on one hand and with our improved atomic pattern (4) on the other hand. We used a chip equipped with an 8-bit CPU running at 30 MHz and with a 32-bit crypto-coprocessor running at 50 MHz. In particular, this crypto-coprocessor provides a dedicated modular squaring. The characteristics of the corresponding implementation are given in Table 5. On the NIST P-192 curve [17] we obtained a practical speed-up of about 14.5 % to be compared to the predicted 20 %. This difference can be explained by the extra software processing required in the scalar multiplication loop management, especially the on-the-fly NAF decomposition of the scalar in an SSCA-resistant way.

Timing	RAM size	Code size
29.6 ms	412 B	3.5 KB

Table 5. Characteristics of our implementation of the atomically protected 192-bit scalar multiplication on an 8-bit chip with a 32-bit crypto-coprocessor.

When observing the side-channel leakage of our implementation we obtained the signal presented in Fig. 4. Atomic patterns comprising 8 modular multiplications and several additions/subtractions can easily be identified.

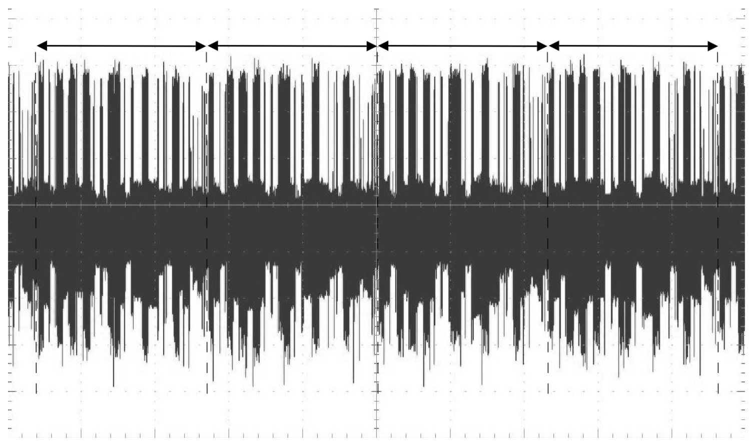

Fig. 4. Side-channel leakage observed during the execution of our scalar multiplication implementation showing a sequence of atomic patterns.

5 Conclusion

In this paper, we propose a new atomic pattern for scalar multiplication on elliptic curves over $\mathbb{F}_p$ and detail our method for atomic pattern improvement. To achieve this goal, two ways are explored. Firstly we maximize the use of squarings to replace multiplications since the latter are slower. Secondly we minimize the use of field additions and negations since they induce a non-negligible penalty. In particular, we point out that the classical hypothesis taken by scalar multiplication designers to neglect the cost of additions/subtractions in $\mathbb{F}_p$ is not valid when focusing on embedded devices such as smart cards.

In this context our method provides an average 18.3 % improvement for the right-to-left mixed scalar multiplication from [21] protected with the atomic pattern from [9]. It also provides an average 10.6 % gain over the fastest algorithm identified before our contribution if dedicated squaring is available. Furthermore, though the topic of this paper is right-to-left scalar multiplication, our atomic pattern improvement method can be generically used to speed up atomically protected algorithms.

In conclusion we recommend that algorithm designers, addressing the scope of embedded devices, take into account additions and subtractions cost when these operations are heavily used in an algorithm. Moreover the issue of designing efficient atomic patterns should be considered when proposing non regular sensitive algorithms.

Acknowledgments

The authors are very grateful to Yannick Sierra for pointing them out the improvement using the subtractions leading to the efficient patterns of Fig. 3. The authors would also like to thank Christophe Clavier, Sean Commercial, Emmanuelle Dottax, Emmanuel Prouff, Matthieu Rivain and the anonymous referees of Cardis 2010 for their helpful comments on the preliminary versions of this paper.

References

1. ANSI X9.62–2005. *Public Key Cryptography for The Financial Service Industry : The Elliptic Curve Digital Signature Algorithm (ECDSA).* American National Standards Institute, Nov. 16, 2005.
2. ANSI X9.63–2001. *Public Key Cryptography for The Financial Service Industry : Key Agreement and Key Transport Using Elliptic Curve Cryptography.* American National Standards Institute, Nov. 20, 2001.
3. S. Arno and F. Wheeler. Signed digit representations of minimal Hamming weight. *IEEE Transactions on Computers*, 42(8):1007–1009, 1993.
4. D. J. Bernstein, P. Birkner, M. Joye, T. Lange, and C. Peters. Twisted edwards curves. In S. Vaudenay, editor, *AFRICACRYPT*, volume 5023 of *Lecture Notes in Computer Science*, pages 389–405. Springer, 2008.

5. D. J. Bernstein and T. Lange. Explicit-formulas database. `http://www.hyperelliptic.org/EFD`.
6. D. J. Bernstein and T. Lange. Faster addition and doubling on elliptic curves. Cryptology ePrint Archive, Report 2007/286, 2007. `http://eprint.iacr.org/`.
7. E. Brier and M. Joye. Weierstraß Elliptic Curves and Side-Channel Attacks. In D. Naccache and P. Paillier, editors, *Public Key Cryptography – PKC 2002*, volume 2274 of *Lecture Notes in Computer Science*, pages 335–345. Springer, 2002.
8. M. Brown, D. Hankerson, J. López, and A. Menezes. Software Implementation of the NIST Elliptic Curves Over Prime Fields. In D. Naccache, editor, *Topics in Cryptology – CT-RSA 2001*, volume 2020 of *Lecture Notes in Computer Science*, pages 250–265. Springer, 2001.
9. B. Chevallier-Mames, M. Ciet, and M. Joye. Low-cost Solutions for Preventing Simple Side-Channel Analysis: Side-Channel Atomicity. *IEEE Transactions on Computers*, 53(6):760–768, 2004.
10. B. Chevallier-Mames and M. Joye. Procédé cryptographique protégé contre les attaques de type à canal caché. French patent, Apr. 2002. FR 28 38 210.
11. H. Cohen, T. Ono, and A. Miyaji. Efficient Elliptic Curve Exponentiation Using Mixed Coordinate. In K. Ohta and P. Dinggyi, editors, *Advances in Cryptology – ASIACRYPT '98*, volume 1514 of *Lecture Notes in Computer Science*, pages 51–65. Springer, 1998.
12. J.-S. Coron. Resistance against Differential Power Analysis for Elliptic Curve Cryptosystems. In Ç. Koç and C. Paar, editors, *Cryptographic Hardware and Embedded Systems – CHES '99*, volume 1717 of *Lecture Notes in Computer Science*, pages 292–302. Springer, 1999.
13. V. Dimitrov, L. Imbert, and P. Mishra. Efficient and Secure Elliptic Curve Point Multiplication using Double-Base Chains. In B. Roy, editor, *Advances in Cryptology – ASIACRYPT 2005*, volume 3788 of *Lecture Notes in Computer Science*, pages 59–78. Springer, 2005.
14. ECC Brainpool. *ECC Brainpool Standard Curves and Curve Generation.* BSI, 2005. v. 1.0 `http://www.ecc-brainpool.org`.
15. ECC Brainpool. *ECC Brainpool Standard Curves and Curve Generation.* BSI, 2009. Internet Draft v. 3 `http://tools.ietf.org/html/draft-lochter-pkix-brainpool-ecc-03`.
16. H. M. Edwards. A normal form for elliptic curves. *Bulletin of the American Mathematical Society*, 44:393–422, 2007.
17. FIPS PUB 186-3. *Digital Signature Standard.* National Institute of Standards and Technology, Mar. 13, 2006. Draft.
18. J. Großschädl, R. M. Avanzi, E. Savas, and S. Tillich. Energy-Efficient Software Implementation of Long Integer Modular Arithmetic. In J. Rao and B. Sunar, editors, *Cryptographic Hardware and Embedded Systems – CHES 2005*, volume 3659 of *Lecture Notes in Computer Science*, pages 75–90. Springer, 2005.
19. D. Hankerson, A. Menezes, and S. Vanstone. *Guide to Elliptic Curve Cryptography.* Springer Professional Computing Series, Jan. 2003.
20. O. Hesse. Uber die Elimination der Variabeln aus drei algebraischen Gleichungen vom zweiten Grade mit zwei Variabeln. *Journal für die reine und angewandte Mathematik*, 10:68–96, 1844.
21. M. Joye. Fast Point Multiplication on Elliptic Curves Without Precomputation. In J. von zur Gathen, J. L. Imaña, and Ç. K. Koç, editors, *Arithmetic of Finite Fields – WAIFI 2008*, volume 5130 of *Lecture Notes in Computer Science*, pages 36–46. Springer, 2008.

22. M. Joye. Highly regular m-ary powering ladders. In M. J. Jacobson, V. Rijmen, and R. Safavi-Naini, editors, *Selected Areas in Cryptography – SAC 2009*, Lecture Notes in Computer Science, pages 135–147. Springer, 2009.
23. M. Joye and C. Tymen. Protections against Differential Analysis for Elliptic Curve Cryptography. In Ç. Koç, D. Naccache, and C. Paar, editors, *Cryptographic Hardware and Embedded Systems – CHES 2001*, volume 2162 of *Lecture Notes in Computer Science*, pages 386–400. Springer, 2001.
24. M. Joye and S.-M. Yen. The Montgomery Powering Ladder. In B. Kaliski Jr., Ç. Koç, and C. Paar, editors, *Cryptographic Hardware and Embedded Systems – CHES 2002*, volume 2523 of *Lecture Notes in Computer Science*, pages 291–302. Springer, 2002.
25. P. Kocher. Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. In N. Koblitz, editor, *Advances in Cryptology – CRYPTO '96*, volume 1109 of *Lecture Notes in Computer Science*, pages 104–113. Springer, 1996.
26. P. Kocher, J. Jaffe, and B. Jun. Differential Power Analysis. In M. Wiener, editor, *Advances in Cryptology – CRYPTO '99*, volume 1666 of *Lecture Notes in Computer Science*, pages 388–397. Springer, 1999.
27. P. Longa. *Accelerating the Scalar Multiplication on Elliptic Curve Cryptosystems over Prime Fields.* PhD thesis, School of Information Technology and Engineering, University of Ottawa, 2007.
28. P. Longa and A. Miri. New Multibase Non-Adjacent Form Scalar Multiplication and its Application to Elliptic Curve Cryptosystems (extended version). Cryptology ePrint Archive, Report 2008/052, 2008. `http://eprint.iacr.org/`.
29. M. Medwed and E. Oswald. Template attacks on ECDSA. Cryptology ePrint Archive, Report 2008/081, 2008. `http://eprint.iacr.org/`.
30. N. Meloni. New point addition formulae for ECC applications. In C. Carlet and B. Sunar, editors, *Arithmetic of Finite Fields – WAIFI 2007*, volume 4547 of *Lecture Notes in Computer Science*, pages 189–201. Springer, 2007.
31. N. Meloni and M. A. Hasan. Elliptic Curve Scalar Multiplication Combining Yao's Algorithm and Double Bases. In C. Clavier and K. Gaj, editors, *Cryptographic Hardware and Embedded Systems – CHES 2009*, volume 5747 of *Lecture Notes in Computer Science*, pages 304–316. Springer, 2009.
32. P. Montgomery. Speeding the Pollard and Elliptic Curve Methods of Factorization. *Mathematics of Computation*, 48:243–264, 1987.
33. SP 800-78-1. *Cryptographic Algorithms and Key Sizes for Personal Identity Verification.* National Institute of Standards and Technology, Aug. 2007.
34. TR-03111. *Elliptic Curve Cryptography Based on ISO 15946.* Federal Office for Information Security (BSI), Feb. 14, 2007.
35. A. C.-C. Yao. On the Evaluation of Powers. *SIAM Journal on Computing*, 5(1):100–103, 1976.

A Atomic formulas

A.1 Atomic General Doubling Using Pattern (2) or (3)

The decomposition of a general – i.e. for all a – doubling in Jacobian coordinates using atomic pattern (3) is depicted hereafter. The corresponding decomposition using atomic pattern (2) can straightforwardly be obtained by replacing every squaring by a multiplication using the same operand twice.

The input point is given as (X_1, Y_1, Z_1) and the result is written into the point (X_2, Y_2, Z_2). Four intermediate registers, R_1 to R_4, are used.

$$1\left[\begin{array}{l} R_1 \leftarrow {X_1}^2 \\ \star \\ R_3 \leftarrow R_1 + R_1 \\ R_2 \leftarrow Z_1 \cdot Z_1 \\ \star \\ R_1 \leftarrow R_1 + R_3 \\ R_4 \leftarrow X_1 + X_1 \end{array}\right.$$

$$2\left[\begin{array}{l} R_2 \leftarrow {R_2}^2 \\ \star \\ \star \\ R_2 \leftarrow a \cdot R_2 \\ \star \\ R_1 \leftarrow R_1 + R_2 \\ R_2 \leftarrow Y_1 + Y_1 \end{array}\right.$$

$$3\left[\begin{array}{l} R_3 \leftarrow {Y_1}^2 \\ \star \\ \star \\ Z_2 \leftarrow Z_1 \cdot R_2 \\ \star \\ R_3 \leftarrow R_3 + R_3 \\ \star \end{array}\right.$$

$$4\left[\begin{array}{l} R_2 \leftarrow {R_1}^2 \\ \star \\ \star \\ R_4 \leftarrow R_4 \cdot R_3 \\ R_4 \leftarrow -R_4 \\ R_2 \leftarrow R_2 + R_4 \\ X_2 \leftarrow R_2 + R_4 \end{array}\right.$$

$$5\left[\begin{array}{l} R_3 \leftarrow {R_3}^2 \\ R_1 \leftarrow -R_1 \\ R_4 \leftarrow X_2 + R_4 \\ R_1 \leftarrow R_1 \cdot R_4 \\ R_3 \leftarrow -R_3 \\ R_3 \leftarrow R_3 + R_3 \\ Y_2 \leftarrow R_3 + R_1 \end{array}\right.$$

A.2 Atomic Readdition Using Pattern (2)

The decomposition of a readdition in Jacobian coordinates using atomic pattern (2) is depicted hereafter.

The input points are given as (X_1, Y_1, Z_1), (X_2, Y_2, Z_2) and the result is written into the point (X_3, Y_3, Z_3). Seven intermediate registers, R_1 to R_7, are used.

$$1\left[\begin{array}{l} R_1 \leftarrow Y_2 \cdot {Z_1}^3 \\ \star \\ \star \\ R_2 \leftarrow Y_1 \cdot Z_2 \\ \star \\ \star \\ \star \end{array}\right.$$

$$2\left[\begin{array}{l} R_3 \leftarrow Z_2 \cdot Z_2 \\ \star \\ \star \\ R_4 \leftarrow R_2 \cdot R_3 \\ R_5 \leftarrow -R_1 \\ R_4 \leftarrow R_4 + R_5 \\ \star \end{array}\right.$$

$$3\left[\begin{array}{l} R_2 \leftarrow X_2 \cdot {Z_1}^2 \\ \star \\ \star \\ R_3 \leftarrow X_1 \cdot R_3 \\ R_5 \leftarrow -R_2 \\ R_3 \leftarrow R_3 + R_5 \\ \star \end{array}\right.$$

$$4\left[\begin{array}{l} R_5 \leftarrow R_3 \cdot R_3 \\ \star \\ \star \\ R_2 \leftarrow R_2 \cdot R_5 \\ R_6 \leftarrow -R_2 \\ R_6 \leftarrow R_6 + R_6 \\ \star \end{array}\right.$$

$$5\left[\begin{array}{l} R_5 \leftarrow R_5 \cdot R_3 \\ \star \\ \star \\ R_3 \leftarrow Z_2 \cdot R_3 \\ \star \\ \star \\ \star \end{array}\right.$$

$$6\left[\begin{array}{l} R_7 \leftarrow R_4 \cdot R_4 \\ \star \\ \star \\ Z_3 \leftarrow R_3 \cdot Z_1 \\ R_5 \leftarrow -R_5 \\ R_7 \leftarrow R_7 + R_6 \\ X_3 \leftarrow R_7 + R_5 \end{array}\right.$$

$$7\left[\begin{array}{l} R_3 \leftarrow R_1 \cdot R_5 \\ R_1 \leftarrow -X_3 \\ R_2 \leftarrow R_2 + R_1 \\ R_1 \leftarrow R_2 \cdot R_4 \\ \star \\ Y_3 \leftarrow R_1 + R_3 \\ \star \end{array}\right.$$

A.3 Atomic Modified Jacobian Coordinates Doubling Using Pattern (1)

The decomposition of a doubling in modified Jacobian coordinates using atomic pattern (1) is depicted hereafter.

The input point is given as (X_1, Y_1, Z_1) and the result is written into the point (X_2, Y_2, Z_2). Six intermediate registers, R_1 to R_6, are used.

$$\begin{array}{ll}
1\left[\begin{array}{l} R_1 \leftarrow X_1 \cdot X_1 \\ R_2 \leftarrow Y_1 + Y_1 \\ \star \\ \star \end{array}\right. & 5\left[\begin{array}{l} R_3 \leftarrow R_1 \cdot R_1 \\ \star \\ \star \\ \star \end{array}\right. \\
2\left[\begin{array}{l} Z_2 \leftarrow R_2 \cdot Z_1 \\ R_4 \leftarrow R_1 + R_1 \\ \star \\ \star \end{array}\right. & 6\left[\begin{array}{l} R_4 \leftarrow R_6 \cdot X_1 \\ R_5 \leftarrow W_1 + W_1 \\ R_4 \leftarrow -R_4 \\ R_3 \leftarrow R_3 + R_4 \end{array}\right. \\
3\left[\begin{array}{l} R_3 \leftarrow R_2 \cdot Y_1 \\ R_6 \leftarrow R_3 + R_3 \\ \star \\ \star \end{array}\right. & 7\left[\begin{array}{l} W_2 \leftarrow R_2 \cdot R_5 \\ X_2 \leftarrow R_3 + R_4 \\ R_2 \leftarrow -R_2 \\ R_6 \leftarrow R_4 + X_2 \end{array}\right. \\
4\left[\begin{array}{l} R_2 \leftarrow R_6 \cdot R_3 \\ R_1 \leftarrow R_4 + R_1 \\ \star \\ R_1 \leftarrow R_1 + W_1 \end{array}\right. & 8\left[\begin{array}{l} R_4 \leftarrow R_6 \cdot R_1 \\ \star \\ R_4 \leftarrow -R_4 \\ Y_2 \leftarrow R_4 + R_2 \end{array}\right.
\end{array}$$