Revisiting Atomic Patterns for Scalar Multiplications on Elliptic Curves

Franck Rondepierre

Oberthur Technologies, Crypto Group
420, rue Estienne d'Orves, 92 700 Colombes, France
f.rondepierre@oberthur.com

Abstract. This paper deals with the protection of elliptic curve scalar multiplications against side-channel analysis by using the atomicity principle. Unlike other atomic patterns, we investigate new formulæ with same cost for both doubling and addition. This choice is particularly well suited to evaluate double scalar multiplications with the Straus-Shamir trick. Thus, in situations where this trick is used to evaluate single scalar multiplications our pattern allows an average improvement of 40% when compared with the most efficient atomic scalar multiplication published so far. Surprisingly, in other cases our choice remains very efficient. Besides, we also point out a security threat when the curve parameter a is null and propose an even more efficient pattern in this case.

Keywords: Elliptic Curves, Scalar Multiplication, Straus-Shamir Trick, Side-Channel Analysis, Atomicity

1 Introduction

The first algorithms performing public-key cryptography, such as the Rivest-Shamir-Adleman (RSA) algorithm [27], have been published in the seventies and remain widely used nowadays. However, current key lengths required with these protocols are limiting their efficiency. Elliptic Curve Cryptography (ECC) provides equivalent cryptographic primitives, but with significant improvements in terms of speed and memory, and is now recommended by governmental organizations such as the National Institute of Standards and Technology (NIST). The main resource-consuming operation in ECC is the computation of a *scalar multiplication* $[k]\,P$ for a secret scalar k and a public point P on an elliptic curve.

We hence consider the implementation on smart cards of scalar multiplications on standardized elliptic curves over $\mathbb{F}_p$. In this context, side-channel resistance, memory and power consumptions have to be taken into account before designing a fast implementation.

Side-Channel Analysis (SCA) is one of the main attack used to disclose secret data hidden in low-resource devices. SCA exploits the fact that a device leaks information about the processed operations and data, that can be physically measured: timing, power consumption, electromagnetic emanations, etc. Among

all kinds of SCA, the *Simple Side-Channel Analysis* (SSCA) [21] focuses on detecting on a single execution trace differences of behavior depending on a secret value. Many proposals have been made to thwart SSCA and the *atomicity principle* introduced by Chevallier-Mames et al. [3] is one of the most efficient propositions. This countermeasure has been widely studied and improved first by P. Longa [22] and then by C. Giraud and V. Verneuil [12].

In this paper, we revisit EC formulæ in a novel way and propose corresponding patterns to optimally benefit from the Straus-Shamir trick, a twice as fast method to evaluate *double scalar multiplications* $[u]\,P + [v]\,Q$. Few ECC protocols explicitly require double scalar multiplications. However as shown in [28] it can be adapted to process single scalar multiplications $[k]\,P$ which gives us an advantage since the best implementation known so far [12] cannot take advantage of this trick. Besides, when this method is not used (e.g. because of memory constraints) our formulæ still allow the most interesting ratio between performances and memory cost. In all cases, our implementation requires less memory than other methods. Besides, we also point out a security flaw concerning many implementations computing over Weierstrass curves with $a = 0$. Therefore, our method outperforms existing solutions in terms of security, memory and speed and is suited for low-resource devices.

The rest of the paper is organized as follows. In the next section we introduce some background on elliptic curves and detail known techniques to perform efficient scalar multiplications. Then Section 3 deals with the security of scalar multiplications against SSCA. In Section 4 we present our formulæ which allow the use of the most efficient scalar multiplication algorithms. Eventually we conclude in Section 5.

2 Elliptic Curve Background

2.1 Definitions

An elliptic curve $\mathcal{E}$ over $\mathbb{F}_p$, for a prime $p > 3$ is defined with the short Weierstrass equation [15]:

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

where $x, y, a, b \in \mathbb{F}_p$ and $4a^3 + 27b^2 \neq 0$.

With the so-called *chord-and-tangent* law, the set of all points on the elliptic curve together with the point at infinity (denoted by $\mathcal{O}$) form an abelian group $\mathcal{E}(\mathbb{F}_p)$. Excepting trivial cases, the group law requires the computation of one inverse in $\mathbb{F}_p$ which is significantly more expensive than a multiplication. Therefore we use Jacobian coordinates to represent points in order to limit the number of inversions performed in a scalar multiplication. These coordinates use the following equivalence class, for non all-zero triples:

$$(X : Y : Z) = \{(\lambda^2 X, \lambda^3 Y, \lambda Z) : \lambda \in \mathbb{F}_p^*\}.$$

In this case the short Weierstrass equation (1) becomes:

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

and $\mathcal{O} = (1 : 1 : 0)$. The opposite of $(X : Y : Z)$ is the point $(X : -Y : Z)$.

The sum of two points $P = (X_p : Y_p : Z_p)$ and $Q = (X_q : Y_q : Z_q)$ is the point $P + Q = (X_{p+q} : Y_{p+q} : Z_{p+q})$ such that:

$$\begin{cases} X_{p+q} = F^2 - E^3 - 2AE^2 \\ Y_{p+q} = F(AE^2 - X_{p+q}) - CE^3 \\ Z_{p+q} = Z_p Z_q E \end{cases} \text{ with } \begin{array}{l} A = X_p Z_q^2 \\ B = X_q Z_p^2 \\ C = Y_p Z_q^3 \\ D = Y_q Z_p^3 \\ E = B - A \\ F = D - C \end{array} \tag{3}$$

However, the addition formula is only valid under the following assumptions: $P \neq Q$, $P \neq \mathcal{O}$ and $Q \neq \mathcal{O}$. Several operations in $\mathbb{F}_p$ are required to evaluate this formula: squarings (denoted by S), multiplications (denoted by M), additions and subtractions (both denoted by A). This formula requires $4S + 12M + 7A$.

The double of a point $P = (X : Y : Z)$ is the point $[2]P = (X_2 : Y_2 : Z_2)$ such that:

$$\begin{cases} X_2 = A^2 - 2C \\ Y_2 = A(C - X_2) - D \\ Z_2 = 2YZ \end{cases} \text{ with } \begin{array}{l} A = 3X^2 + aZ^4 \\ B = 2Y \cdot Y \\ C = 2BX \\ D = 2B \cdot B \end{array} \tag{4}$$

Using this formula, the evaluation of a doubling requires $4S + 6M + 9A$.

The result of $P + P$ is naturally denoted by the point $[2]P$ and such an operation is called a *doubling* whereas the addition rather refers to the computation of $P + Q$ with $P \neq \pm Q$. More generally, the operation $P + \ldots + P$ where the point P is added k times is called a *scalar multiplication* and is denoted by $[k]P$.

2.2 Efficient Scalar Multiplication Implementation

This section presents the different known tools to optimize the implementation of an elliptic curve multiplication.

Double and Add. The scalar multiplication is efficiently computed with the so-called Double and Add algorithm [20]. Using the EC group law, this algorithm evaluates:

$$[k]P = \sum_{i=0}^{\ell-1} [k_i]\left[2^i\right]P = [k_0]P + [2]\left[\sum_{i=1}^{\ell-1} k_i 2^{i-1}\right]P$$

where digits $k_i \in \mathcal{S}$ and $\mathcal{S}$ is some set of integers containing 0 and 1, as presented in [23]. The ℓ digits of k can be evaluated in two ways, i.e. starting from least significant ones (right to left) or from most significant ones (left to right), see Algorithm 1 for the later case.

Double and Add trade-off. The scalar multiplication consists in a succession of doublings and additions. Depending on the ratio between the number of evaluated additions and the number of doublings, one formula may be favored at the expense of the other in order to reduce the overall cost.

Algorithm 1 Left to Right Double and Add

Input: $k = \sum_{i=0}^{\ell-1} k_i 2^i$, $k_{\ell-1} \neq 0$, $k_i \in \mathcal{S}$, $P \in \mathcal{E}$
Output: $R = [k]\,P$
  Precompute $[m]\,P$, $\forall m \in \mathcal{S}\backslash\{0\}$
  Initialize $R = [k_{\ell-1}]\,P$
  for $i = \ell - 2$ **downto** 0 **do**
    $R = [2]\,R$
    if $k_i \neq 0$ **then**
      $R = R + [k_i]\,P$
    end if
  end for

- First, the cost of the addition formula (3) can be reduced with some assumptions. Indeed, in Algorithm 1, the points $[m]\,P$ are constant. Hence Chudnovsky [4] proposed to compute $Z^2_{[m]P}$ and $Z^3_{[m]P}$ once for all, which saves $1S + 1M$ per point addition. Furthermore, at the cost of one inversion and few multiplications[1] during the precomputation phase, one can choose the representative of $[m]\,P$ with $Z_{[m]P} = 1$, which instead saves $1S + 4M$ in (3). Besides, since there is always a doubling of R before the point addition $R+[k_i]\,P$, we propose to move the computation of Z_r^2 and Z_r^3 in the doubling formula which also saves $1S + 1M$.
- The doubling formula (4) can be speeded up [5] with the help of one extra value W initialized with the value aZ^4. In this case, A is evaluated as $3X^2 + W$ and W_2 is equal to $2DW$, which gives a global cost of $2S + 6M + 10A$. However, in Algorithm 1, if the point R is represented as $(X_r : Y_r : Z_r : W_r)$ then the value W_r has to remain consistent when the addition $R + [k_i]\,P$ is performed. Therefore Formula (3) has to update this value W: $W_{r+q} = W_r(E^2 Z^2_{[k_i]P})^2$ which costs either $1S + 2M$ if $Z_{[k_i]P} \neq 1$ or $1S + 1M$.

We will now see two techniques used to change the addition/doubling ratio.

Straus-Shamir Trick. The Straus-Shamir trick [30] [8] is a simple but very efficient way of evaluating double scalar multiplications $[u]\,P + [v]\,Q$. The naive way to evaluate the double multiplication performs two multiplications and adds the results. Using two calls to Algorithm 1, this approach costs on average 2ℓ doublings and ℓ additions (for random scalars u and v). However, neither $[u]\,P$ nor $[v]\,Q$ are needed values. The trick consists then in building one sequence of intermediate results directly converging to the value $[u]\,P + [v]\,Q$ in one execution of a Double and Add algorithm. Algorithm 2 implements such a trick which only requires on average ℓ doublings and 0.75ℓ additions.

This trick can also be used to improve the evaluation of $[k]\,P$ if the multiplication is decomposed as:

$$[k]\,P = [k_0]\,P + [k_1]\,([\lambda]\,P).$$

[1] A trick from Montgomery [24] enables to evaluate several inverses at the cost of only one inversion and few multiplications: $\frac{1}{a} = \frac{1}{ab} \cdot b$, $\frac{1}{b} = \frac{1}{ab} \cdot a$.

Of course, the trick has an interest when k_0 and k_1 have a size $\ell/2$ and when k_0, k_1 and $Q = [\lambda] P$ are available with reasonable cost. As recalled in [28], one can take $\lambda = 2^{\ell/2}$ and precompute Q when the point P is reused for many scalar multiplications. One can remark that this trick – working only with left-to-right scan of digits – is generalizable for multi-multiplication.

Algorithm 2 Double Scalar Multiplication using Straus-Shamir Trick

Input: $u = \sum_{i=0}^{\ell-1} u_i 2^i$, $v = \sum_{i=0}^{\ell-1} v_i 2^i$, $(u_{\ell-1}, v_{\ell-1}) \neq (0,0)$, $(u_i, v_i) \in \mathcal{S}^2$, $(P, Q) \in \mathcal{E}^2$, $P \neq \pm Q$
Output: $R = [u] P + [v] Q$
  Precompute $W_{i,j} = [i] P + [j] Q$, $\forall (i,j) \in \mathcal{S}^2 \backslash \{(0,0)\}$
  Initialize $R = W_{u_{\ell-1}, v_{\ell-1}}$
  for $i = \ell - 2$ **downto** 0 **do**
    $R = [2] R$
    if $(u_i, v_i) \neq (0,0)$ **then**
      $R = R + W_{u_i, v_i}$
    end if
  end for

Scalar Recoding. The previous trick allows to cut to half the number of doublings. Independently, the number of additions can also be reduced by using windowing and signed digit representations in order to maximize the number of null digits of the scalar. The best signed digit representations to reduce the number of non-zero digits are the Non-Adjacent Form (NAF) [1] for one scalar, and the Joint Sparse Form (JSF) [28] for a couple of scalars. NAF and JSF have been generalized to larger digit set than $\{0, \pm 1\}$ [29] [26] but only NAF_{w+1} (which indicates a NAF representation with window size w) fits smart card memory constraints. Tables 1 and 2 indicate, for several cases, the definition of the set $\mathcal{S}$, the number of involved points which corresponds to a certain RAM cost, and the average number of point addition performed per bit which also corresponds to the ratio between additions and doublings.

Table 1: Several scalar recoding techniques and their average number of point additions per bit of k in the context of single scalar multiplication $[k] P$

Recoding technique	None	NAF	$\text{NAF}_{w=3}$	$\text{NAF}_{w=4}$
Involved points	P	$\pm P$	$\pm P$, $\pm 3P$	$\pm P$, $\pm 3P$, $\pm 5P$, $\pm 7P$
$\mathcal{S}$	$\{0, 1\}$	$\{-1, 0, 1\}$	$\{-3, -1, 0, 1, 3\}$	$\{-7, -5, -3, -1, 0, 1, 3, 5, 7\}$
Point additions/bit	1/2	1/3	1/4	1/5

Table 2: Several scalar recoding techniques and their average number of point additions per bit of k in the context of double scalar multiplication $[u]\,P + [v]\,Q$

Recoding technique	None	JSF
Involved points	P, Q, $P+Q$	$\pm P$, $\pm Q$, $\pm(P+Q)$, $\pm(P-Q)$
$\mathcal{S}$	$\{0,1\}$	$\{-1,0,1\}$
Point additions/bit	3/4	1/2

3 Secure Scalar Multiplication Implementation

As shown in Formulæ (3) and (4) the evaluation of a doubling is different from the evaluation of an addition: the number of each type of operation and the number of operations in total are not the same. These differences are easily detected via SSCA and give information on the handled scalar k. Indeed in Algorithm 1 the addition is performed only if k_i is not null, thus knowing that 2 successive doublings have been performed means that the corresponding k_i was null. If the binary representation has been used, i.e. $k_i \in \mathcal{S} = \{0,1\}$, then the knowledge of all indexes where k_i is null is enough to retrieve the whole value k. Using other scalar representation reduces the amount of information leaked but sophisticated attacks [9] only need some bits to be successful. This section deals with the proposed countermeasures to thwart such a leakage.

3.1 State-of-the-Art

Secure Implementation. Many algorithms have been published so far to resist this SSCA with different strategies. A unified formula has been proposed [2] to evaluate the sum of two points P, Q (different from $\mathcal{O}$) without the restriction $P \neq Q$. The Double and Add Always [6] performs the addition whatever the value of the digit k_i. Other regular algorithms such as the Montgomery Ladder [11] [18] [2] perform a doubling and an addition for each bit but without dummy operations which is more interesting from a security point of view. Recent works [16] [14] have investigated efficient Montgomery Ladder formulæ with good results, but this approach cannot benefit from scalar recoding and hence remains costly if compared with best non-secure implementations. The principle of atomicity focuses on reaching the same security level at a closer cost of non-secure implementations.

Atomicity Principle. The atomicity principle [3] can be seen as a refined unified formula. Instead of having one formula valid in both situations ($P = Q$ and $P \neq \pm Q$) this countermeasure focuses on evaluating two different formulæ using the same operations or the same flow of operations. On can notice that such an evaluation is always possible but may induce additional dummy operations. The tour-de-force of Chevallier-Mames et al. [3] was to propose such an evaluation at an almost negligible cost, which is without any dummy modular multiplication.

They have split the two formulæ in a sequence of identical *atomic patterns* such that ten calls of this pattern make a doubling, while sixteen calls give an addition. Their pattern is depicted below, where the R_i's denote some intermediate values.

$$\begin{cases} R_1 \leftarrow R_2 \cdot R_3 \\ R_4 \leftarrow R_5 + R_6 \\ R_7 \leftarrow -R_8 \\ R_9 \leftarrow R_{10} + R_{11} \end{cases} \tag{5}$$

However, as shown in Table 3 such a pattern implies a lot of dummy additions (A) and negations (N) when evaluating Formulæ (3) and (4). If the cost of one addition or a negation is small compared to a modular multiplication it cannot be neglected yet. Another drawback of the formula lies in the loss of squares traded for multiplications, which implies a loss of efficiency when a dedicated function is available to evaluate squares faster than multiplications. Therefore, this pattern (5) has been first improved by Longa [22] and more reworked by Giraud and Verneuil (GV) [12]. In the rest of the paper, as explained in [12], we assume $a = -3$ and Chudnovsky optimization for Longa's implementation of Algorithm 1. The GV implementation optimizes doublings with the extra value W but uses a right-to-left scan of digits to limit the cost of point additions.

Table 3: Comparing operation cost between main atomic implementations and non-secure implementations, with $2N = A$

Algorithm	Pattern Cost	Addition Cost	Doubling Cost
Chevallier-Mames *et al.* [3]	$M + 2A + N$	$16M + 40A$	$10M + 25A$
Longa L2R [22]	$2M + 3A + 2N$	$14M + 28A$	$8M + 16A$
GV R2L [12]	$2S + 6M + 10A$	$4S + 12M + 20A$	$2S + 6M + 10A$
non-secure R2L	-	$4S + 12M + 7A$	$2S + 6M + 10A$
non-secure L2R	-	$4S + 9M + 7A$	$2S + 6M + 10A$

3.2 Scalar Evaluation

The elliptic curve formulæ are not the only part to secure when considering implementations protected with the atomicity principle. The security of the scalar treatment is also crucial to thwart SSCA similar to the following example. For instance, let us assume that the evaluation at round i of a doubling $\mathcal{D}_i$ cannot be distinguished from the evaluation of a point addition $\mathcal{A}_i$ (i.e. atomic patterns have been used), let E_i denote the scalar treatment at round i, then observing the following sequence (6) gives information on the scalar k. Indeed, patterns corresponding to a doubling and an addition are performed between E_i and E_{i-1} which means that $k_i \neq 0$ while $k_{i-1} = 0$ since only a doubling is performed

between E_{i-1} and E_{i-2}.

$$\mathcal{D}_i, E_i, \mathcal{A}_i, \mathcal{D}_{i-1}, E_{i-1}, \mathcal{D}_{i-2}, E_{i-2}, \dots \tag{6}$$

Hence, if the atomic pattern must be called x (resp. y) times to perform a point addition (resp. point doubling), then a call to the scalar treatment has to be done every $\gcd(x, y)$ pattern. Besides, this scalar treatment has to implement the atomicity principle to have the same behavior whatever the value of the digit k_i. The induced overhead required to securely evaluate the secret scalar is generally under-estimated. Actually, this consists in the main limitation of the atomic countermeasure when the scalar multiplication has to be implemented on components that cannot process the scalar treatment in parallel of the pattern evaluation. Taking $x = y$ allows to reduce the cost of this treatment.

4 New Atomic Pattern

4.1 New Fomulæ

Up to now, improvements on atomic implementations have mainly focused on optimizing the doubling formula. This strategy comes from the observation that a doubling has to be computed for each bit of the scalar, while the addition is only performed for non-zero digit. In the context of processing one scalar (e.g. Algorithm 1), there is a low addition/doubling ratio which validates this strategy (see Table 1). However this is not the case in the double scalar multiplication.

Hence, we propose hereafter to focus on improving point addition to better match double multiplication case. The new formulæ are detailed hereafter to ease the verification of the proposed atomic patterns. The sum of $P = (X : Y : Z : Z^2 : Z^3)$ and $Q = (X_q : Y_q : 1)$ is the point $P + Q = (X_3 : Y_3 : Z_3 : Z_3^2)$ such that:

$$\begin{cases} X_3 = F^2 - E^3 - 2AE^2 \\ Y_3 = F(AE^2 - X_3) - CE^3 \\ Z_3 = ZE \\ Z_3^2 = (Z_3)^2 \end{cases} \quad \text{with} \quad \begin{aligned} A &= X \\ B &= X_q Z^2 \\ C &= Y \\ D &= Y_q Z^3 \\ E &= B - A \\ F &= D - C \end{aligned} \tag{7}$$

The subtraction $P - Q$ is obtained by replacing F by $\bar{F} = D + 2C - C$ and then $Y_3 = \bar{F}(X_3 - AE^2) - CE^3$. The double of a point $P = (X : Y : Z : Z^2)$ is the point $[2]\,P = (X_2 : Y_2 : Z_2 : Z_2^2 : Z_2^3)$ such that:

$$\begin{cases} X_2 = A^2 - 2C \\ Y_2 = A(C - X_2) - D \\ Z_2 = 2YZ \\ Z_2^2 = (Z_2)^2 \\ Z_2^3 = (Z_2)^3 \end{cases} \quad \text{with} \quad \begin{aligned} A &= 3(X - IZ^2)(X + IZ^2) \\ B &= 2Y \cdot Y \\ C &= 2BX \\ D &= 2B \cdot B \end{aligned} \tag{8}$$

We introduce a new constant value $I = \sqrt{-a3^{-1}}$ in order to reduce the cost of doublings. For a random value a (i.e. a random curve) this constant I exists with probability 0.5. However, for most standard curves (NIST [10], Brainpool [7], ANSSI [19]) the constant I exists and may have some particular value (e.g. $I = 1$ or $I = 0$). The case $I = 0$ has a special treatment in Section 4.3. A pattern working for all curves is given in Appendix A. This pattern is less efficient, trading an addition for a square, but still of interest.

Eventually we have $3S + 7M + 7A$ to evaluate the addition and $2S + 8M + 10A$ for the doubling, which means that a square has to be performed as a modular multiplication and three dummy modular additions are required in the addition formula in order to get a unified cost. Besides, in order to save some memory when using NAF or JSF recodings, we benefit from the dummy modular additions to also present a subtraction formula which avoids the storage of opposite points.

From $P = (X, Y, Z, Z^2, Z^3)$ and $Q(X_q, Y_q, 1)$, one can compute $P \leftarrow P + Q$, $P \leftarrow P - Q$ and $P \leftarrow [2]\,P$ with the following patterns, where each pattern requires $2S + 8M + 10A$:

Addition	Subtraction	Doubling
$R_1 \leftarrow X_q \cdot Z^2$	$R_1 \leftarrow X_q \cdot Z^2$	$R_0 \leftarrow I \cdot Z^2$
$R_1 \leftarrow R_1 - X$	$R_1 \leftarrow R_1 - X$	$R_1 \leftarrow X - R_0$
$\star \leftarrow \star + \star$	$Z^2 \leftarrow Y + Y$	$R_2 \leftarrow Y + Y$
$R_2 \leftarrow R_1 \cdot R_1$	$R_2 \leftarrow R_1 \cdot R_1$	$Z_2^2 \leftarrow Y \cdot R_2$
$\star \leftarrow \star + \star$	$\star \leftarrow \star + \star$	$Y_2 \leftarrow Z_2^2 + Z_2^2$
$R_3 \leftarrow X \cdot R_2$	$R_3 \leftarrow X \cdot R_2$	$R_3 \leftarrow R_2 \cdot Z$
$R_0 \leftarrow Y_q \cdot Z^3$	$R_0 \leftarrow Y_q \cdot Z^3$	$R_2 \leftarrow Y_2 \cdot X$
$\star \leftarrow \star + \star$	$R_0 \leftarrow Z^2 + R_0$	$X_2 \leftarrow X + R_0$
$Z^3 \leftarrow R_1 \cdot R_2$	$Z^3 \leftarrow R_1 \cdot R_2$	$R_0 \leftarrow R_1 \cdot X_2$
$R_2 \leftarrow Z \cdot R_1$	$R_2 \leftarrow Z \cdot R_1$	$R_1 \leftarrow Z_2^2 \cdot Y_2$
$X_3 \leftarrow R_3 + R_3$	$X_3 \leftarrow R_3 + R_3$	$X_2 \leftarrow R_0 + R_0$
$X_3 \leftarrow Z^3 + X_3$	$X_3 \leftarrow Z^3 + X_3$	$R_0 \leftarrow R_0 + X_2$
$Z_3^2 \leftarrow (R_2)^2$	$Z_3^2 \leftarrow (R_2)^2$	$X_2 \leftarrow (R_0)^2$
$R_0 \leftarrow R_0 - Y$	$R_0 \leftarrow R_0 - Y$	$X_2 \leftarrow X_2 - R_2$
$R_1 \leftarrow (R_0)^2$	$R_1 \leftarrow (R_0)^2$	$Z_2^2 \leftarrow (R_3)^2$
$X_3 \leftarrow R_1 - X_3$	$X_3 \leftarrow R_1 - X_3$	$X_2 \leftarrow X_2 - R_2$
$R_1 \leftarrow R_3 - X_3$	$R_1 \leftarrow X_3 - R_3$	$R_2 \leftarrow R_2 - X_2$
$R_3 \leftarrow R_1 \cdot R_0$	$R_3 \leftarrow R_1 \cdot R_0$	$Z_2^3 \leftarrow Z_2^2 \cdot R_3$
$R_0 \leftarrow Y \cdot Z^3$	$R_0 \leftarrow Y \cdot Z^3$	$Y_2 \leftarrow R_0 \cdot R_2$
$Y_3 \leftarrow R_3 - R_0$	$Y_3 \leftarrow R_3 - R_0$	$Y_2 \leftarrow Y_2 - R_1$
$Z_3 \leftarrow R_2$	$Z_3 \leftarrow R_2$	$Z_2 \leftarrow R_3$

In order to reduce the number of intermediate buffers R_i, the output buffers (e.g. X_3, X_2) have been used as intermediate buffers which allows to use 4 intermediate buffers R_i only. Besides particular attention has been paid to allow in-place EC operations (e.g. an overlap of buffers X and X_2). However, the patterns do not contain in-place modular multiplications ($R_1 \leftarrow R_1 \cdot R_2$) as discussed in [16] to limit the memory consumption.

4.2 Performances

In this section, the performances of the new pattern are compared with best known atomic patterns for both single scalar and double scalar multiplication cases. In order to evaluate the cost of an implementation we use the average per bit cost. We put aside the cost of pre-computations, or post-computations since it gives only a small advantage to the right-to-left variant. The per bit cost is obtained with the following formula:

$$\mathcal{D} + E + H \cdot (\mathcal{A} + E)$$

where $\mathcal{D}$ (resp. $\mathcal{A}$) stands for the cost of a doubling (resp. addition), E is the cost to treat the scalar and H is the average number of point additions per bit of the scalar.

Theoretical Study. We combine here the cost of point addition and doubling (see Table 3 and Section 4.1) with the addition/doubling ratios given in tables 1 and 2. *Remark:* As indicated in Table 3 we assume $2N = A$. Indeed the negation of a value v already reduced modulo p can be performed as the subtraction $p - v$ without the need of modular reduction.

Double Scalar Multiplication. The RAM available in the smart card context limits the ratio H to be greater than 0.5 when using Algorithm 2. As shown in Table 4, our pattern always offers the best performances when compared with Longa's pattern.

Table 4: Double Scalar Multiplication Cost Per Bit

Algorithm	$H = 3/4$	$H = 1/2$
Longa L2R	$18.5M + 37A + 9.25E$	$15M + 30A + 7.5E$
This paper	$3.5S + 14M + 17.5A + 1.75E$	$3S + 12M + 15A + 1.5E$

Single Scalar Multiplication. Due to smart card memory constraints, lower values for H (i.e. less than 0.5) correspond to single scalar multiplication cases. We present in Table 5 the limit of our method encountered with our practical values (Table 9). The new solution still requires less modular additions than other

propositions but it requires more products. The ratios A/M and S/M will help to select the solution with the best per bit cost. The cost of exponent recoding E shall also be carefully taken into account in the case of on-the-fly scalar recoding. Indeed, protecting this heavy scalar treatment against SSCA may be costly.

Table 5: Single Scalar Multiplication Cost Per Bit

Algorithm	$H = 1/4$	$H = 1/5$
Longa L2R	$11.5M + 23A + 5.75E$	$10.8M + 21.6A + 5.4E$
GV R2L	$3S + 9M + 15A + 1.5E$	$2.8S + 8.4M + 14A + 1.4E$
This paper	$2.5S + 10M + 12.5A + 1.25E$	$2.4S + 9.6M + 12A + 1.2E$

Memory Cost. The smaller the ratio point additions per bit (i.e. H), the more buffers required in memory. Depending on the supported ECC bit lengths these buffers correspond to more or less RAM. Therefore the developer selects a ratio with regard to the desired memory cost. The new pattern has been optimized in order to only use four intermediate registers (see Section 4.1) and the same results can be obtained for the pattern of Giraud-Verneuil – and should be possible for Longa's pattern. If X denotes the number of extra points (see Tables 1 and 2), an implementation using Longa's pattern requires $12 + 5X$ buffers (assuming a subtraction pattern), $11 + 3X$ buffers are required using GV pattern and only $11 + 2X$ buffers are required with the new pattern.

Table 6: Number of buffers required for the scalar multiplication

Operation	Recoding	H	GV R2L	Longa L2R	This paper
$[u]P + [v]Q$	none	3/4	-	22	15
	JSF	1/2	-	27	17
$[k]P$	NAF	1/3	11	12	11
	$\text{NAF}_{w=3}$	1/4	14	17	13
	$\text{NAF}_{w=4}$	1/5	20	27	17

Practical Values. One usually considers the chip characteristics before selecting the best matching algorithm. In Table 7 we give measured costs of elementary operations performed in the field $\mathbb{F}_p$ on our chip. We also give the relative cost of a secured on-the-fly NAF scalar recoding. Therefore, the following table 8 contains a line with $E/M = 0$ since we did not implement on-the-fly JSF recoding. This choice is greatly in favor of Longa's pattern (see Table 4) and in practice E/M may not be negligible. If we look at the 224-bit size (considered as secured in the midterm [13]) we see an improvement of at least 14.6% for

the double scalar multiplication case. In the single scalar multiplication case (Table 9), the new proposition has a performance close to the Giraud-Verneuil solution. However, a fair comparison must take into account the memory cost of each algorithm which emphasize the interest of the new pattern. Besides, the small gain/loss of the new pattern in the single scalar multiplication case may be seen as a fair cost if we consider that only one implementation is now required for both double scalar multiplication and single scalar multiplication situations. Eventually, about components with $S < M$, our propositions benefit from fast squares but to a lesser extent than GV since it relatively contains less squares (see Tables 4 and 5).

In conclusion, though the new pattern is not optimal in case of low addition/doubling ratio – i.e. when large amount of memory is available – and asymptotically not optimal – i.e. when $A/M \approx 0$ – it has revealed to be efficient in our context.

Table 7: Characteristics of our implementation on a smart card

Bit length	160	192	224	256	320	384	512	521
A/M	0.23	0.21	0.21	0.19	0.17	0.16	0.14	0.14
E/M	0.95	0.65	0.65	0.47	0.36	0.28	0.19	0.19

Table 8: Number of equivalent modular multiplications for the double scalar multiplication with $S/M = 1$

H	ECC bit size	Longa L2R	This paper	Gain
1/2 ($E/M = 0$)	160	21.9	18.4	15.7%
	224	21.2	18.1	14.6%
	256	20.6	17.8	13.7%
	320	20.2	17.6	12.8%
	384	19.8	17.4	12.0%
	521	19.1	17.1	10.8%

4.3 The special case $a = 0$

If the curve parameter a is null then a multiplication with zero is performed in the doubling formula which may result in a security flaw. Indeed, such a product may have a leakage distinguishable from other products. Since the addition formula does not use this parameter a, it can hence be distinguished from a doubling. This case $a = 0$ allows to only save one multiplication in efficient doubling formulas (using an extra coordinate W or $a = -3$). Therefore, using Longa's pattern

Table 9: Number of equivalent modular multiplications for the single scalar multiplication with $S/M = 1$

H	ECC size	Longa L2R	GV R2L	This paper	Gain over GV
1/4	160	22.2	16.9	16.5	1.8%
	224	20.0	16.0	15.9	1.1%
	256	18.5	15.5	15.4	0.6%
	320	17.5	15.1	15.1	0.1%
	384	16.8	14.8	14.8	-0.2%
	521	15.7	14.3	14.4	-0.7%
1/5	160	20.9	15.7	15.9	-0.9%
	224	18.7	15.0	15.3	-1.7%
	256	17.4	14.5	14.8	-2.3%
	320	16.5	14.1	14.5	-2.7%
	384	15.8	13.8	14.2	-3.1%
	521	14.8	13.4	13.9	-3.6%

or Giraud-Verneuil pattern, one cannot benefit from this case to improve the scalar multiplication since their atomic patterns contain several multiplications. An improvement is possible with the pattern of Chevallier-Mames et al. but this pattern remains costly due to the high number of dummy additions. Hence, a new formula is proposed here to thwart this attack and to benefit from the possible speed improvement.

With $a = 0$, there are no more available modular subtractions at the beginning of the doubling formula therefore the atomic pattern has to be rebuilt to match with the addition formula. The trick used here consists in saving in memory the opposite values of the coordinates of Q: $-X_q, -Y_q$. The point equivalent class is also used with $\lambda = -1$ to represent $P + Q = (X_3 : -Y_3 : -Z_3)$. However, no trick has been found to propose a subtraction formula.

The sum of $P = (X : Y : Z : Z^2 : Z^3)$ and $Q = (X_q : Y_q : 1)$ is the point $P + Q = (X_3 : Y_3 : Z_3)$. This sum is depicted in Formula (9). The double of a point $P = (X : Y : Z)$ is the point $[2]\,P = (X_2 : Y_2 : Z_2 : Z_2^2 : Z_2^3)$. The evaluation is performed using Formula (8) with $A = 3X^2$. The resulting pattern only contains 9 multiplications ($2S + 7M$) and 8 additions which represents an improvement of more than 10% if compared with the case $a \neq 0$. It is depicted in Table 11 in Appendix A.

$$\begin{cases} X_3 = F^2 - (2A\bar{E}^2 - \bar{E}^3) \\ Y_3 = \bar{F}(A\bar{E}^2 - X_3) - C\bar{E}^3 \\ Z_3 = Z\bar{E} \end{cases} \quad \text{with} \quad \begin{array}{l} A = X \\ \bar{B} = (-X_q)Z^2 \\ C = Y \\ \bar{D} = (-Y_q)Z^3 \\ \bar{E} = \bar{B} + A \\ \bar{F} = \bar{D} + C \end{array} \tag{9}$$

5 Conclusion

In this paper a new atomic pattern has been proposed that outperforms the implementations of most scalar multiplications used in elliptic curve cryptography standards. Our pattern enables to securely perform double scalar multiplications on curves over $\mathbb{F}_p$ with the Straus-Shamir trick which has not been done before. The new pattern also turns out to be efficient for single scalar multiplication. We give hereafter in Table 10 the results obtained on our chip when using main standard algorithms compared with the best known implementations. The first row corresponds to the process of $[k]\,G$ evaluated as $[k_0]\,G + [k_1]\,2^{\ell/2}G$. The double scalar multiplication can also be used in the PACE [17] protocol to evaluate a point $\tilde{G}$. A Key Agreement requires the computation of $[k]\,P$. Longa's atomic pattern using Straus-Shamir trick is given for information only as its implementation requires too much memory to be used with our component. Hence the GV implementation was the fastest algorithm available to perform signatures and key generations and this proposal improves it by 38.6%.

Table 10: Example costs to perform a secure scalar multiplication on a 256-bit curve

Used in	Multiplication / bit	Memory cost	Algorithm
Keygen, Sign	8.9	17	Algo. 2, this paper
	10.3	27	Algo. 2, Longa L2R
Key Agreement	14.5	20	GV R2L
	14.8	17	Algo. 1, this paper
PACE	17.8	17	Algo. 2, this paper
	20.6	27	Algo. 2, Longa L2R

Acknowledgements

The author is grateful to Christophe Giraud and Emmanuelle Dottax for their valuable comments on preliminary versions of this article. Many thanks also go to anonymous reviewers of Cardis 2013 for their advices.

References

1. Arno, S., Wheeler, F.: Signed digit representations of minimal Hamming weight. IEEE Transactions on Computers 42(8), 1007–1009 (1993)
2. Brier, E., Joye, M.: Weierstraß Elliptic Curves and Side-Channel Attacks. In: Naccache and Paillier [25], pp. 335–345
3. Chevallier-Mames, B., Ciet, M., Joye, M.: Low-Cost Solutions for Preventing Simple Side-Channel Analysis: Side-Channel Atomicity. Cryptology ePrint Archive, Report 2003/237 (2003), http://eprint.iacr.org/

4. Chudnovsky, D.V., Chudnovsky, G.V.: Sequences of numbers genereated by addition in formal groups and new primality and factorization tests. Advances in Applied Mathematics 7, 385–434 (1986)
5. Cohen, H., Ono, T., Miyaji, A.: Efficient Elliptic Curve Exponentiation Using Mixed Coordinate. In: Ohta, K., Dinggyi, P. (eds.) Advances in Cryptology – ASIACRYPT '98. LNCS, vol. 1514, pp. 51–65. Springer (1998)
6. Coron, J.S.: Resistance against Differential Power Analysis for Elliptic Curve Cryptosystems. In: Koç, Ç., Paar, C. (eds.) Cryptographic Hardware and Embedded Systems – CHES '99. LNCS, vol. 1717, pp. 292–302. Springer (1999)
7. 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
8. ElGamal, T.: A Public-Key Cryptosystems and a Signature Scheme based on Discret Logarithms. IEEE Transaction on Information Theory 31(4), 469–172 (1985)
9. Faugère, J.C., Goyet, C., Renault, G.: Attacking (EC)DSA Given Only an Implicit Hint. In: Knudsen, L.R., Wu, H. (eds.) Selected Areas in Cryptography – SAC 2012. LNCS, vol. 7707, pp. 252–274. Springer (2013)
10. FIPS PUB 186-4: Digital Signature Standard. National Institute of Standards and Technology (Jul 2013)
11. Fischer, W., Giraud, C., Knudsen, E.W., Seifert, J.P.: Parallel scalar multiplication on general elliptic curves over $\mathbb{F}_p$ hedged against Non-Differential Side-Channel Attacks. Cryptology ePrint Archive, Report 2002/007 (Jan 2002), http://eprint.iacr.org/
12. Giraud, C., Verneuil, V.: Atomicity Improvement for Elliptic Curve Scalar Multiplication. In: Gollmann, D., Lanet, J.L. (eds.) Smart Card Research and Advanced Applications, 9th International Conference – CARDIS 2010. LNCS, vol. 6035, pp. 80–101. Springer (2010)
13. Giry, D., Bulens, P.: Keylength.com – Cryptographic Key Length Recommandation (Aug 2007), http://www.keylength.com
14. Goundar, R.R., Joye, M., Miyaji, A., Rivain, M., Venelli, A.: Scalar multiplication on weierstraß elliptic curves from co-z arithmetic. Journal of Cryptology 1(2), 161–176 (2011)
15. Hankerson, D., Menezes, A., Vanstone, S.: Guide to Elliptic Curve Cryptography. Springer Professional Computing Series (Jan 2003)
16. Hutter, M., Joye, M., Sierra, Y.: Memory-Constrained Implementations of Elliptic Curve Cryptography in Co-Z Coordinate Representation. In: Nitaj, A., Pointcheval, D. (eds.) AFRICACRYPT 2011. LNCS, vol. 6737, pp. 170–187. Springer (2011)
17. ISO/IEC JTC1 SC17 WG3/TF5: Supplemental Access Control for Machine Readable Travel Documents. International Civial Aviation Organization (Nov 2010)
18. Izu, T., Takagi, T.: A Fast Parallel Elliptic Curve Multiplication Resistant against Side Channel Attacks. In: Naccache and Paillier [25], pp. 280–296
19. JORF n: Avis relatif aux paramètres de courbes elliptiques définis par l'État français (Oct 2011)
20. Knuth, D.: The Art of Computer Programming, vol. 2. Addison Wesley, third edn. (1988)
21. Kocher, P.: Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. In: Koblitz, N. (ed.) Advances in Cryptology – CRYPTO '96. LNCS, vol. 1109, pp. 104–113. Springer (1996)

22. Longa, P.: Accelerating the Scalar Multiplication on Elliptic Curve Cryptosystems over Prime Fields. Master's thesis, School of Information Technology and Engineering, University of Ottawa, Canada (2007)
23. Möller, B.: Improved Techniques for Fast Exponentiation. In: Lee, P., Lim, C. (eds.) Information Security and Cryptology – ICISC 2002. LNCS, vol. 2587, pp. 298–312. Springer (2002)
24. Montgomery, P.: Modular multiplication without trial division. Math. Comp. 44(170), 519–521 (Apr 1985)
25. Naccache, D., Paillier, P. (eds.): Public Key Cryptography – PKC 2002, LNCS, vol. 2274. Springer (2002)
26. Okeya, K., Katou, H., Ogami, Y.: Width-3 joint sparse form. In: Kwak, J., Deng, R.H., Won, Y., Wang, G. (eds.) ISPEC. LNCS, vol. 6047, pp. 67–84. Springer (2010)
27. Rivest, R., Shamir, A., Adleman, L.: A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. Communications of the ACM 21(2), 120–126 (1978)
28. Solinas, J.: Low-Weight Binary Representations for Pairs of Integers. Tech. rep. (2001), http://cacr.uwaterloo.ca/techreports/2001/corr2001-41.ps
29. Solinas, J.A.: Efficient arithmetic on koblitz curves. Design, Codes and Cryptography 19(2/3), 195–249 (2000)
30. Straus, E.G.: Addition chains of vectors (problem 5125). American Mathematical Monthly 70, 806–808 (1964)

A Atomic Patterns

The patterns for any value a allow to perform an addition or doubling at a cost of $3S + 8M + 9A$. These patterns implement Formulæ (9) and (4):

Table 11: Atomic patterns for the case $a = 0$ (left-hand side) and for any value a (right-hand side)

Addition	Doubling
$\star \leftarrow \star + \star$	$R_0 \leftarrow Y + Y$
$R_0 \leftarrow (-X_q) \cdot Z^2$	$R_1 \leftarrow R_0 \cdot Y$
$R_1 \leftarrow X + R_0$	$Y_2 \leftarrow R_1 + R_1$
$R_2 \leftarrow (R_1)^2$	$R_2 \leftarrow (X)^2$
$R_0 \leftarrow X \cdot R_2$	$R_3 \leftarrow R_0 \cdot Z$
$R_3 \leftarrow Z \cdot R_1$	$Z_2 \leftarrow Y_2 \cdot X$
$X_3 \leftarrow (-Y_q) \cdot Z^3$	$R_0 \leftarrow R_1 \cdot Y_2$
$Z_3 \leftarrow R_2 \cdot R_1$	$Z_2^2 \leftarrow R_3 \cdot R_3$
$R_1 \leftarrow R_0 + R_0$	$R_1 \leftarrow R_2 + R_2$
$R_2 \leftarrow Y + X_3$	$R_1 \leftarrow R_1 + R_2$
$X_3 \leftarrow (R_2)^2$	$X_2 \leftarrow (R_1)^2$
$R_1 \leftarrow R_1 - Z_3$	$X_2 \leftarrow X_2 - Z_2$
$X_3 \leftarrow X_3 - R_1$	$X_2 \leftarrow X_2 - Z_2$
$R_0 \leftarrow R_0 - X_3$	$Z_2 \leftarrow Z_2 - X_2$
$R_1 \leftarrow Y \cdot Z_3$	$Z_2^3 \leftarrow Z_2^2 \cdot R_3$
$Y_3 \leftarrow R_0 \cdot R_2$	$Y_2 \leftarrow R_1 \cdot Z_2$
$Y_3 \leftarrow Y_3 - R_1$	$Y_2 \leftarrow Y_2 - R_0$
$Z_3 \leftarrow R_3$	$Z_2 \leftarrow R_3$

Addition	Doubling
$R_0 \leftarrow (Z)^2$	$R_0 \leftarrow (Z)^2$
$\star \leftarrow \star + \star$	$R_1 \leftarrow Y + Y$
$R_1 \leftarrow (-X_q) \cdot R_0$	$R_2 \leftarrow R_1 \cdot Y$
$R_1 \leftarrow X + R_1$	$Y_2 \leftarrow R_2 + R_2$
$R_3 \leftarrow (-Y_q) \cdot R_0$	$R_3 \leftarrow R_1 \cdot Z$
$R_2 \leftarrow (R_1)^2$	$Z_2 \leftarrow (X)^2$
$R_0 \leftarrow Z \cdot R_3$	$R_1 \leftarrow Y_2 \cdot R_2$
$R_3 \leftarrow Z \cdot R_1$	$R_2 \leftarrow Y_2 \cdot X$
$Z_3 \leftarrow R_1 \cdot R_2$	$X_2 \leftarrow a \cdot R_0$
$R_1 \leftarrow X \cdot R_2$	$Y_2 \leftarrow X_2 \cdot R_0$
$R_2 \leftarrow R_1 + R_1$	$R_0 \leftarrow Z_2 + Z_2$
$R_0 \leftarrow R_0 + Y$	$X_2 \leftarrow R_0 + Z_2$
$\star \leftarrow \star + \star$	$R_0 \leftarrow X_2 + Y_2$
$X_3 \leftarrow (R_0)^2$	$X_2 \leftarrow (R_0)^2$
$R_2 \leftarrow R_2 - Z_3$	$X_2 \leftarrow X_2 - R_2$
$X_3 \leftarrow X_3 - R_2$	$X_2 \leftarrow X_2 - R_2$
$R_1 \leftarrow R_1 - X_3$	$Z_2 \leftarrow R_2 - X_2$
$R_2 \leftarrow Y \cdot Z_3$	$\star \leftarrow \star \cdot \star$
$Y_3 \leftarrow R_0 \cdot R_1$	$Y_2 \leftarrow R_0 \cdot Z_2$
$Y_3 \leftarrow Y_3 - R_2$	$Y_2 \leftarrow Y_2 - R_1$
$Z_3 \leftarrow R_3$	$Z_2 \leftarrow R_3$