Published in P. Paillier and I. Verbauwhede, Eds.,*Cryptographic Hardware and Embedded Systems – CHES 2007*, vol. 4727 of *Lecture Notes in Computer Science*, pp. 135–147, Springer, 2007.

Highly Regular Right-to-Left Algorithms for Scalar Multiplication

Marc Joye

Thomson R&D France
Technology Group, Corporate Research, Security Laboratory
1 avenue de Belle Fontaine, 35576 Cesson-Sévigné Cedex, France
marc.joye@thomson.net

Abstract. This papers introduces several binary scalar multiplication algorithms with applications to cryptography. Remarkably, the proposed algorithms regularly repeat the same pattern when evaluating a scalar multiplication in an (additively written) abelian group. Furthermore, they are generic and feature the following properties:
- no dummy operation is involved;
- no precomputation nor prior recoding is needed;
- a small number of temporary registers / code memory is required;
- the scalar is processed right-to-left.

As a result, they are particularly well fitted for implementing cryptosystems in constrained devices, in an efficient yet secure way.

Keywords: Scalar multiplication, exponentiation, implementation attacks, constrained devices, cryptography.

1 Introduction

Fast exponentiation techniques (e.g., [19]) are central in the implementation of public-key cryptography. RSA cryptosystem uses exponentiation in $\mathbb{Z}_N^*$; Diffie-Hellman, DSA or ElGamal cryptosystem use exponentiation in $\mathbb{F}_p^*$ or on elliptic curves [34]. The efficiency of all these systems depend heavily on the underlying exponentiation algorithm.

This paper proposes new exponentiation methods in an arbitrary abelian group $\mathbb{G}$. As we choose to use additive notation (i.e., the group operation in $\mathbb{G}$ is denoted by $+$ and the identity element by $\boldsymbol{O}$), exponentiation in $\mathbb{G}$ is usually referred to as *scalar multiplication.* Given a scalar k and an element $\boldsymbol{P} \in \mathbb{G}$, we want to compute $k\boldsymbol{P}$, that is, $\boldsymbol{P} + \boldsymbol{P} + \ldots + \boldsymbol{P}$ (k times).

In constrained devices, this is usually achieved through the *double-and-add algorithm* (or square-and-multiply algorithm in multiplicative notation). Letting $k = \sum_{j=0}^{t-1} k_j\, 2^i$ with $k_j \in \{0, 1\}$ denote the binary expansion of k, the double-and-add algorithm runs as follows.

Input: $\boldsymbol{P} \in \mathbb{G}$ and $k = (k_{t-1}, \ldots, k_0)_2 \in \mathbb{N}$
Output: $\boldsymbol{Q} = k\boldsymbol{P}$
$\boldsymbol{R_0} \leftarrow \boldsymbol{O}$
for $j = t-1$ downto 0 **do**
  $\boldsymbol{R_0} \leftarrow 2\boldsymbol{R_0}$
  if $(k_j = 1)$ **then** $\boldsymbol{R_0} \leftarrow \boldsymbol{R_0} + \boldsymbol{P}$
end for
return $\boldsymbol{R_0}$

In addition to efficiency, another concern when implementing cryptosystems is of course security. The methods we propose resist certain side-channel attacks and fault attacks.

Simple power analysis (SPA) exploits distinct patterns from a single power trace [28]. If scalar k represents a secret value, it can be so recovered because, as presented, the double-and-add algorithm behaves irregularly. Making it regular is however not difficult by replacing the *if-then* statement with an indirect addressing and by adding a dummy group addition when bit k_j is equal to 0 [11]. More explicitly, the line '**if** $(k_j = 0)$ **then** $\boldsymbol{R_0} \leftarrow \boldsymbol{R_0} + \boldsymbol{P}$' can be replaced with

$$\boldsymbol{R_{1-k_j}} \leftarrow \boldsymbol{R_{1-k_j}} + \boldsymbol{P} \ .$$

Although protecting against SPA-type attacks, the so-obtained algorithm is now vulnerable to *safe-error attacks* [48]. By timely inducing a fault during the evaluation of '$\boldsymbol{R_{1-k_j}} \leftarrow \boldsymbol{R_{1-k_j}} + \boldsymbol{P}$', an attacker may deduce whether bit $k_j = 0$ by checking if the returned result, $\boldsymbol{Q} = k\boldsymbol{P}$, is correct —for example, if $\boldsymbol{Q}$ is a digital signature (or a part thereof), the attacker may check its validity using the corresponding public verification key. Indeed, when bit $k_j = 0$, the faulty evaluation $\boldsymbol{R_1} + \boldsymbol{P}$ is written back in $\boldsymbol{R_1}$ and as $\boldsymbol{R_1}$ is not needed, the output of the algorithm won't be affected.

The so-called Montgomery ladder [36] protects against SPA-type attacks and safe-error attacks, at the same time [26]: it is highly regular and does not involve dummy operation. The algorithm is given below.

Input: $\boldsymbol{P} \in \mathbb{G}$ and $k = (k_{t-1}, \ldots, k_0)_2 \in \mathbb{N}$
Output: $\boldsymbol{Q} = k\boldsymbol{P}$
$\boldsymbol{R_0} \leftarrow \boldsymbol{O}$; $\boldsymbol{R_1} \leftarrow \boldsymbol{P}$
for $j = t-1$ downto 0 **do**
  $b \leftarrow 1 - k_j$; $\boldsymbol{R_b} \leftarrow \boldsymbol{R_b} + \boldsymbol{R_{k_j}}$
  $\boldsymbol{R_{k_j}} \leftarrow 2\boldsymbol{R_{k_j}}$
end for
return $\boldsymbol{R_0}$

In this paper, we present algorithms that enjoy the security features of Montgomery ladder (i.e., resistance against SPA-type attacks and safe-error attacks[1]) but with various performance Moreover, they can be combined with previous techniques (e.g., [5, Chapter V] or [10, Chapter 29]) to offer resistance against DPA-type attacks [28, 23]. Finally, they process the multiplier bits from the right to the left, which offers a better resistance against certain attacks (e.g., [16, 47]).

The rest of this paper is organized as follows. In the next section, we present our new algorithms. In Section 3, we give illustrations to the secure implementation of cryptographic algorithms in constrained devices. Finally, we conclude in Section 4.

2 New Algorithms

Throughout this section, we let $\mathbb{G}$ denote an additively written abelian group with identity element $\boldsymbol{O}$. We also let $\mathrm{ord}_{\mathbb{G}}(\boldsymbol{P})$ denote the order of $\boldsymbol{P}$ as an element in $\mathbb{G}$.

On input an element $\boldsymbol{P} \in \mathbb{G}$ and a t-bit integer k, the goal is to compute $\boldsymbol{Q} = k\boldsymbol{P} \in \mathbb{G}$. We present below several new right-to-left binary methods for evaluating $k\boldsymbol{P}$.

2.1 Double-add algorithm

Let $\sum_{j=0}^{t-1} k_j\, 2^j$ with $k_j \in \{0, 1\}$ be the binary expansion of k. We have:

$$\boldsymbol{Q} = \sum_{j=0}^{t-1} (k_j\, 2^j)\boldsymbol{P} = \sum_{j=0}^{t-1} k_j\, \boldsymbol{B_j} \;\text{ with }\; \boldsymbol{B_j} = 2^j \boldsymbol{P} \enspace .$$

Now, for $j \geqslant 0$, we define

$$\boldsymbol{S_j} = \sum_{i=0}^{j} k_i\, \boldsymbol{B_i} \quad \text{and} \quad \boldsymbol{T_j} = \boldsymbol{B_{j+1}} - \boldsymbol{S_j} \enspace .$$

Hence we get

$$\begin{aligned} \boldsymbol{S_j} &= \sum_{i=0}^{j} k_i\, \boldsymbol{B_i} = k_j\, \boldsymbol{B_j} + \boldsymbol{S_{j-1}} = k_j(\boldsymbol{S_{j-1}} + \boldsymbol{T_{j-1}}) + \boldsymbol{S_{j-1}} \\ &= (1 + k_j)\boldsymbol{S_{j-1}} + k_j\, \boldsymbol{T_{j-1}} \,, \end{aligned}$$

and

$$\begin{aligned} \boldsymbol{T_j} &= \boldsymbol{B_{j+1}} - \boldsymbol{S_j} = 2\boldsymbol{B_j} - (k_j\, \boldsymbol{B_j} + \boldsymbol{S_{j-1}}) = (2 - k_j)\boldsymbol{B_j} - \boldsymbol{S_{j-1}} \\ &= (2 - k_j)\boldsymbol{T_{j-1}} + (1 - k_j)\boldsymbol{S_{j-1}} \enspace . \end{aligned}$$

[1] We do not consider fault attacks on the control logic [17]. Furthermore, we note that randomizing scalar k mitigates such attacks.

Equivalently, we have shown:

Proposition 1. *For any $j \geqslant 0$,*

$$\boldsymbol{S_j} = \begin{cases} \boldsymbol{S_{j-1}} & \textit{if } k_j = 0 \\ 2\boldsymbol{S_{j-1}} + \boldsymbol{T_{j-1}} & \textit{if } k_j = 1 \end{cases} \tag{1}$$

and

$$\boldsymbol{T_j} = \begin{cases} \boldsymbol{S_{j-1}} + 2\boldsymbol{T_{j-1}} & \textit{if } k_j = 0 \\ \boldsymbol{T_{j-1}} & \textit{if } k_j = 1 \end{cases} . \tag{2}$$

□

Noting that $\boldsymbol{Q} = k\boldsymbol{P} = \boldsymbol{S_{t-1}}$, this yields the following right-to-left scalar multiplication algorithm. At the end of iteration j, temporary registers $\boldsymbol{R_0}$ and $\boldsymbol{R_1}$ respectively contain the values of $\boldsymbol{S_j}$ and of $\boldsymbol{T_j}$.

Algorithm 1 Double-add scalar multiplication algorithm.

Input: $\boldsymbol{P} \in \mathbb{G}$ and $k = (k_{t-1}, \ldots, k_0)_2 \in \mathbb{N}$
Output: $\boldsymbol{Q} = k\boldsymbol{P}$

```
1: R_0 ← O; R_1 ← P
2: for j = 0 to t − 1 do
3:     b ← 1 − k_j; R_b ← 2R_b + R_{k_j}
4: end for
5: return R_0
```

Remark 1. In certain groups $\mathbb{G}$, adding identity element $\boldsymbol{O}$ needs a special treatment, which, in turn, may reveal the number of least significant zero-bits of k. This can be circumvented by adding to k an appropriate multiple of $\mathrm{ord}_{\mathbb{G}}(\boldsymbol{P})$ to ensure that the parity of the resulting scalar is always the same. This can also be circumvented by computing $k\boldsymbol{P}$ with the least significant bit of k forced to 1, $k \leftarrow (k - k_0) + 1$, and then by subtracting $\boldsymbol{P}$ to the so-obtained element if bit k_0 is zero. This second approach is useful for scalars $k \ll \mathrm{ord}_{\mathbb{G}} \boldsymbol{P}$. A similar trick is used in Algorithm 2 (see next section).

Algorithm 1′ Double-add scalar multiplication algorithm (for small scalars).

Input: $\boldsymbol{P} \in \mathbb{G}$ and $k = (k_{t-1}, \ldots, k_0)_2 \in \mathbb{N}$
Output: $\boldsymbol{Q} = k\boldsymbol{P}$

```
1: R_0 ← P; R_1 ← P
2: for j = 0 to t − 1 do
3:     b ← 1 − k_j; R_b ← 2R_b + R_{k_j}
4: end for
5: b ← k_0; R_b ← R_b − P
6: return R_0
```

2.2 Add-only algorithm

Algorithm 1 can be modified into an algorithm only involving additions in $\mathbb{G}$, that is, *without involving doublings*. This may be advantageous as *only* the general addition operation in $\mathbb{G}$ has to be implemented, resulting in some code/memory savings.

An extra temporary register, $\boldsymbol{R_2}$, is used to store the value of $\boldsymbol{B_{j+1}} = \boldsymbol{S_j} + \boldsymbol{T_j}$. Remember that temporary registers $\boldsymbol{R_0}$ and $\boldsymbol{R_1}$ are used to respectively store the values of $\boldsymbol{S_j}$ and of $\boldsymbol{T_j}$.

Algorithm 2 Add-only scalar multiplication algorithm.

Input: $\boldsymbol{P} \in \mathbb{G}$ and $k = (k_{t-1}, \ldots, k_0)_2 \in [0, \mathrm{ord}_{\mathbb{G}}(\boldsymbol{P})[$
Output: $\boldsymbol{Q} = k\boldsymbol{P}$
1: $\boldsymbol{R_0} \leftarrow \boldsymbol{P}$; $\boldsymbol{R_1} \leftarrow \boldsymbol{P}$; $\boldsymbol{R_2} \leftarrow 2\boldsymbol{P}$
2: **for** $j = 1$ to $t - 1$ **do**
3: $\quad b \leftarrow 1 - k_j$; $\boldsymbol{R_b} \leftarrow \boldsymbol{R_b} + \boldsymbol{R_2}$
4: $\quad \boldsymbol{R_2} \leftarrow \boldsymbol{R_0} + \boldsymbol{R_1}$
5: **end for**
6: $b \leftarrow k_0$; $\boldsymbol{R_b} \leftarrow \boldsymbol{R_b} - \boldsymbol{P}$
7: **return** $\boldsymbol{R_0}$

We assume that k is odd (i.e., $\boldsymbol{R_0}$ is initialized to $\boldsymbol{P}$ and loop counter starts at $j = 1$) in the computation of $\boldsymbol{Q} = k\boldsymbol{P}$ and subtract $\boldsymbol{P}$ at the end of the computation if it is not the case (cf. Line 6). Let $g = \mathrm{ord}_{\mathbb{G}}(\boldsymbol{P})$. We also assume that scalar k is smaller than g. Finally, we let $k_{\ell-1}$ denote the most significant bit of k different from 0, i.e., $k_{\ell-1} = 1$ and $k_j = 0$ for $\ell \leqslant j \leqslant t - 1$.

Because $k < g$ is assumed to be odd, temporary register $\boldsymbol{R_0}$ always contains an odd multiple of $\boldsymbol{P}$;[2] the same is true for temporary register $\boldsymbol{R_1}$ for $j < \ell - 1$ since $\boldsymbol{T_j} = 2^{j+1}\boldsymbol{P} - \boldsymbol{S_j}$. Similarly, temporary register $\boldsymbol{R_2}$ always contains a power of 2 multiple of $\boldsymbol{P}$ for $j < \ell - 1$. Consequently, it follows that, for $1 \leqslant j \leqslant \ell - 2$,

- $\boldsymbol{R_b} \neq \boldsymbol{R_2}$ in the computation $\boldsymbol{R_b} \leftarrow \boldsymbol{R_b} + \boldsymbol{R_2}$ (cf. Line 3), and
- $\boldsymbol{R_0} \neq \boldsymbol{R_1}$ in the computation $\boldsymbol{R_2} \leftarrow \boldsymbol{R_0} + \boldsymbol{R_1}$ (cf. Line 4) — note that $\boldsymbol{R_2}$ containing $2^{j+1}\boldsymbol{P}$ can only be the sum of two equal odd multiples of $\boldsymbol{P}$ when $\boldsymbol{R_0}, \boldsymbol{R_1} \leftarrow \boldsymbol{P}$, which never occurs when $j \geqslant 1$,

and when $j = \ell - 1$,

- temporary register $\boldsymbol{R_0}$ gets the value of $k\boldsymbol{P}$ (and is no longer modified for $\ell \leqslant j \leqslant t - 1$).

We further note that the evaluation of $2\boldsymbol{P}$ (cf. Line 1) does not necessarily require a doubling operation as its value can be evaluated as $2\boldsymbol{P} = (\boldsymbol{P} + \boldsymbol{A}) + (\boldsymbol{P} - \boldsymbol{A})$ for an arbitrary element $\boldsymbol{A} \in \mathbb{G} \setminus \{\boldsymbol{O}\}$ and $\mathrm{ord}_{\mathbb{G}}(\boldsymbol{A}) \neq 2$.

[2] More precisely, we mean that if $\boldsymbol{R_0} \leftarrow r\boldsymbol{P}$ then $r \pmod{g}$ is odd.

Remark 2. Certain randomization techniques for preventing differential side-channel analysis lead to a scalar larger than $\mathrm{ord}_{\mathbb{G}}(\boldsymbol{P})$. For example, in [11], Coron suggests to evaluate $\boldsymbol{Q} = k\boldsymbol{P}$ as $\boldsymbol{Q} = k^*\boldsymbol{P}$ where $k^* = k + r\ \mathrm{ord}_{\mathbb{G}}(\boldsymbol{P})$ for a (small) random integer r. Letting $g = \mathrm{ord}_{\mathbb{G}}(\boldsymbol{P})$, $b = \lfloor \log_2(g) \rfloor$, and $\beta = 2^b$, we write $k^* = \sum_{i=0}^{l-1} k_i^*\, \beta^i$ with $k_i^* \in \{0, \ldots, \beta - 1\}$. The computation of $k^*\boldsymbol{P}$ can then be carried out with Algorithm 2 as

$$\boldsymbol{Q} = \sum_{i=0}^{l-1} k_i^*\, (\beta^i \boldsymbol{P}) \quad \text{with } k_i^* < g\ ,$$

by observing that the value of $\beta^{(i+1)}\boldsymbol{P}$ is available in temporary register $\boldsymbol{R_2}$ at the end of the computation of $k_i^*\, (\beta^i \boldsymbol{P})$.

2.3 Add-always algorithms

There is another possible modification. By construction, we have $\boldsymbol{B_{j+1}} = \boldsymbol{S_j} + \boldsymbol{T_j} = 2\boldsymbol{B_j}$. The updating step of $\boldsymbol{R_2}$ in Algorithm 2 (Line 4) can thus be replaced with $\boldsymbol{R_2} \leftarrow 2\boldsymbol{R_2}$. Doing so, however, the operation $\boldsymbol{R_b} \leftarrow \boldsymbol{R_b} + \boldsymbol{R_2}$ (Line 3) becomes dummy when $b = 1$ — and so the resulting algorithm may be subject to safe-error attacks.

The right way is to first perform a doubling and next an addition. This yields a right-to-left binary algorithm *only using two temporary registers* (and not three) and where *all group operations are effective*, as depicted in Algorithm 3. The correctness of the so-obtained algorithm easily follows from Algorithm 1. Note also that it can be adapted when $\boldsymbol{O}$ behaves differently (see Remark 1).

Algorithm 3 Add-always scalar multiplication algorithm (General case).

Input: $\boldsymbol{P} \in \mathbb{G}$ and $k = (k_{t-1}, \ldots, k_0)_2 \in \mathbb{N}$
Output: $\boldsymbol{Q} = k\boldsymbol{P}$
$\boldsymbol{R_0} \leftarrow \boldsymbol{O}$; $\boldsymbol{R_1} \leftarrow \boldsymbol{P}$
for $j = 0$ to $t - 1$ **do**
  $b \leftarrow 1 - k_j$; $\boldsymbol{R_b} \leftarrow 2\boldsymbol{R_b}$
  $\boldsymbol{R_b} \leftarrow \boldsymbol{R_b} + \boldsymbol{R_{k_j}}$
end for
return $\boldsymbol{R_0}$

Rewriting Line 4 of Algorithm 2 as $\boldsymbol{R_2} \leftarrow 2\boldsymbol{R_2}$ can nevertheless be useful to get an algorithm for the evaluation of *Lucas recurrences* — namely, elements $\boldsymbol{x_n} \in \mathscr{L}$ satisfying

$$\begin{cases} \boldsymbol{x_{t+i}} = f(\boldsymbol{x_i}, \boldsymbol{x_t}, \boldsymbol{x_{t-i}}) & \text{for } i \neq t \\ \boldsymbol{x_{2t}} = g(\boldsymbol{x_t}) \end{cases}$$

by observing that the relation $\boldsymbol{R_2} - \boldsymbol{R_b} = \boldsymbol{R_{k_j}}$ is always satisfied throughout the algorithm: hence both $\boldsymbol{R_b}$ and $\boldsymbol{R_{k_j}}$ are needed for the "add" operation. See Section 3.2 for an illustration.

Algorithm 4 Add-always algorithm for Lucas recurrences.

Input: $\boldsymbol{x_0}, \boldsymbol{x_1} \in \mathscr{L}$ and $k = (k_{t-1}, \ldots, k_0)_2 \in \mathbb{N}$
Output: $\boldsymbol{v} = \boldsymbol{x_k}$

$\boldsymbol{R_0} \leftarrow \boldsymbol{x_0}$; $\boldsymbol{R_1} \leftarrow \boldsymbol{x_1}$; $\boldsymbol{R_2} \leftarrow \boldsymbol{x_1}$
for $j = 0$ to $t - 1$ **do**
  $b \leftarrow 1 - k_j$; $\boldsymbol{R_b} \leftarrow \mathsf{Ladd}(\boldsymbol{R_b}, \boldsymbol{R_2}, \boldsymbol{R_{k_j}})$
  $\boldsymbol{R_2} \leftarrow \mathsf{Ldouble}(\boldsymbol{R_2})$
end for
return $\boldsymbol{R_0}$

3 Applications

In this section, we present several applications of our algorithms to the secure implementation of elliptic curve cryptography. The security of elliptic curve cryptosystems relies on the difficulty of the discrete logarithm problem in the group $\mathbb{G}$ of points of an elliptic curve over a finite field. Elliptic curve based applications require the computation of $\boldsymbol{Q} = k\boldsymbol{P}$ in $\mathbb{G}$ where $k \in [1, \mathrm{ord}_{\mathbb{G}}(\boldsymbol{P})[$ represents an ephemeral or a long-term secret (see, e.g., [4, 10, 22]).

Consider the elliptic curve

$$E/\mathbb{F}_{p^m} : y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6$$

defined over the finite field $\mathbb{F}_{p^m}$, with $a_i \in \mathbb{F}_{p^m}$. In cryptography applications, elliptic curves are usually defined either over large prime fields (i.e., $m = 1$ and $|p|_2 \geqslant 160$) or binary fields (i.e., $p = 2$ and $m \geqslant 160$). The curve equations then simplify to

$$E/\mathbb{F}_p : y^2 = x^3 + a_4x + a_6$$

and

$$E/\mathbb{F}_{2^m} : y^2 + xy = x^3 + a_2x^2 + a_6$$

assuming non-supersingular binary curves. The set of points (x, y) satisfying the above equations together with the 'point at infinity' $\boldsymbol{O}$ form an abelian group $\mathbb{G}$ under the chord-and-tangent rule, where $\boldsymbol{O}$ is the neutral element. We summarize in the Table 1 the cost of certain point operations.

As aforementioned, attention should be paid when implementing scalar multiplication algorithms for cryptographic purposes for the evaluation of $k\boldsymbol{P}$ as they may be prone to implementation attacks. The algorithms of the previous section were devised as a protection against SPA-type attacks and safe-error attacks. We recall that they can be used in conjunction with other techniques to offer resistance against other classes of attacks. See, e.g., [5, Chapter V] or [10, Chapter 29] for a collection of useful techniques.

3.1 Optimal extension fields

The ratio I/M is of crucial interest when comparing the performance of different algorithms. In particular, this ratio depends on the type and representation of the

Table 1. Number of inversions (I), squarings (S) and multiplications (M) to add, double and double-add points on elliptic curves.

Operation	$\mathbb{F}_p$ cost ($a_1 = a_2 = a_3 = 0$)	$\mathbb{F}_{2^m}$ cost ($a_1 = 1,\ a_3 = a_4 = 0$)
$\boldsymbol{P}+\boldsymbol{Q}$	$1I + 1S + 2M$	$1I + 1S + 2M$
$2\boldsymbol{P}$	$1I + 2S + 2M$	$1I + 1S + 2M$
$2\boldsymbol{P}+\boldsymbol{Q}$ ([14])	$2I + 2S + 3M$	$2I + 2S + 3M$
$2\boldsymbol{P}+\boldsymbol{Q}$ ([8])	$1I + 2S + 9M$	$1I + 2S + 9M$

underlying finite field [7, 21]. If the field inversion is too expensive then projective coordinates are preferred [12]. In [43], Smart gives a comprehensive comparison for different fields representations. He concludes that optimal extension fields (OEFs) offer the best performance, even when used with affine coordinates.

OEFs were introduced by Bailey and Paar [2] for optimized field arithmetic (see also [27]). An OEF is a field $\mathbb{F}_{p^m} \simeq \mathbb{F}_p[t]/(t^m - \omega)$ where p is a pseudo-Mersenne prime of the form $2^n \pm c$ that fits in a processor word. In an OEF, a typical value for the ratio I/M is 4.

From Table 1, we see that the evaluation of point $\boldsymbol{Q} = k\boldsymbol{P}$ on an elliptic curve costs 2 inversions (resp. 1 inversion), 2 squarings and 3 multiplications (resp. 9 multiplications) per bit of scalar k, using our double-add algorithm (Algorithm 1). This represents an improvement of

$$\beta_\delta = \frac{\delta\, S + \max\{M, I - 5M\}}{2I + (2+\delta)S + 4M}$$

compared to the classical SPA-resistant binary algorithms ([11]), where $\delta = 1$ over $\mathbb{F}_p$ and $\delta = 0$ over $\mathbb{F}_{2^m}$. For example, if we estimate $S/M = 1$ and $I/M = 4$, we get a **13.3%** improvement for elliptic curves over an OEF.

3.2 Right-to-left López-Dahab ladder

Carrying out the evaluation of $\boldsymbol{Q} = k\boldsymbol{P}$ with the x-coordinate *only* is a promising venue to speed up the computations: since the y-coordinate does not need to be evaluated, one may expect to save some modular multiplications. This technique was successfully applied to the elliptic curve method of factoring by Montgomery on special-form elliptic curves [36]. We remark that not all elliptic curves admit a Montgomery form [38]; addition formulas for general elliptic curves over (large) prime fields are given in [6, 25] (see also [45, Chapter 5]).

López and Dahab adapted Montgomery's method to non-supersingular elliptic curves over binary fields [32]. They obtained a beautiful algorithm for evaluating the x-coordinate of $k\boldsymbol{P}$ in projective coordinates, and allowing the recovery of the y-coordinate.

If we let $\boldsymbol{x_n}$ denote the x-coordinate of $n\boldsymbol{P}$ on an elliptic curve over $\mathbb{F}_{2^m}$, it can be shown that $\{\boldsymbol{x_n}\}$ satisfies a Lucas recurrence [1, 32]. Consequently, Algorithm 4 can be used to evaluate the x-coordinate of $k\boldsymbol{P}$, namely $\boldsymbol{v} := \boldsymbol{x_k}$. We use projective coordinates, the x-coordinate of $n\boldsymbol{P}$, $\boldsymbol{x_n} = X_n/Z_n$, is represented by the pair (X_n, Z_n). For improved efficiency, we use the following curve equation[3]

$$E/\mathbb{F}_{2^m} : y^2 + a_1xy = x^3 + a_2x^2 + a_1^{-2}$$

introduced by Stam [44]. The Ladd and $\mathsf{Ldouble}$ routines are then given by

$$\begin{aligned}(X_{t+i}, Z_{t+i}) &= \mathsf{Ladd}\big((X_i, Z_i), (X_t, Z_t), (X_{t-i}, Z_{t-i})\big)\\ &:= \big(Z_{t-i}(X_iX_t + Z_iZ_t)^2, X_{t-i}(X_iZ_t + X_tZ_i)^2\big)\end{aligned}$$

and

$$(X_{2t}, Z_{2t}) = \mathsf{Ldouble}\big((X_t, Z_t)\big) := \big((X_t + Z_t)^4, (a_1X_tZ_t)^2\big)\ .$$

If the cross-product $(X_iZ_t + X_tZ_i)$ is computed as $(X_i + Z_i)(X_t + Z_t) + X_iX_t + Z_iZ_t$ then Ladd takes 5 multiplications and 2 squarings in $\mathbb{F}_{2^m}$. The evaluation of $\mathsf{Ldouble}$ takes 1 general multiplication, 1 multiplication by constant a_1 and 3 squarings in $\mathbb{F}_{2^m}$. Consequently, neglecting the cost of squarings, we can construct a secure *right-to-left* version of López-Dahab ladder, which requires 6 multiplications plus 1 multiplication by constant a_1 per bit of scalar.

3.3 Further applications

Algorithm 2 only requires the *general* addition formula for adding points: no doublings are involved. Hence, there is no need to implement a routine for doubling points. This results in *code savings* (or area savings for hardware implementations).[4]

This may also give rise to better performance. For example, in [9, Exerc. 4 in § 10.6], Cohen reports formulas for adding points with 9 multiplications and for doubling a point with 10 multiplications on an elliptic curve given by a Fermat parameterization in projective coordinates. In this case, the faster addition formula leads to a **5%** speed-up improvement.

Another application of our algorithms resides in pairing-based cryptography (e.g., [5, Chapter X] or [10, Chapter 24]). For example, in contrast to the version of Miller's algorithm as described in [3] for computing pairings, Algorithm 1 leads to the evaluation of a parabola at a point rather than the evaluation of lines and of their product at a point. As detailed in [14], this needs less effort. Moreover, this results in an algorithm protected against certain implementation attacks [40].

---

[3] As shown in [44], any non-supersingular elliptic curve over $\mathbb{F}_{2^m}$ is isomorphic to this form.

[4] To a lesser extent, this also holds for Algorithm 1 provided that the code for $2\boldsymbol{P}+\boldsymbol{Q}$ is more compact than the total code for $\boldsymbol{P}+\boldsymbol{Q}$ and $2\boldsymbol{P}$.

4 Conclusion

This paper presented highly regular right-to-left binary scalar multiplication algorithms (or exponentiation algorithms for multiplicatively written groups). The proposed algorithms are very simple to implement (both in hardware and in software) and require little memory. Moreover, they are protected against SPA-type attacks and safe-error attacks, and can be combined with previous techniques to offer resistance against other attacks. Consequently, we believe that our algorithms are useful for cryptographic applications in constrained devices as they offer a number of advantages compared to the classical binary algorithms.

References

1. G.B. Agnew, R.C. Mullin, and S.A. Vanstone, *An implementation of elliptic curve cryptosystems over* $\mathbb{F}_{2^{155}}$, IEEE Journal on Selected Areas in Communications **11** (1993), no. 5, 804–813.
2. D. Bailey and C. Paar, *Optimal extension fields for fast arithmetic in public-key algorithms*, Advances in Cryptology − CRYPTO '98 (H. Krawczyk, ed.), Lecture Notes in Computer Science, vol. 1492, Springer-Verlag, 1999, pp. 472–485.
3. P.S.L.M. Barreto, H.Y. Kim, B. Lynn, and M. Scott, *Efficient algorithms for pairing-based cryptosystems*, Advances in Cryptology − CRYPTO 2002 (M. Yung, ed.), Lecture Notes in Computer Science, vol. 2442, Springer-Verlag, 2002, pp. 354–368.
4. I.F. Blake, G. Seroussi, and N.P. Smart, *Elliptic curves in cryptography*, London Mathematical Society Lecture Note Series, no. 265, Cambridge University Press, 1999.
5. ______, *Advances in elliptic curve cryptography*, London Mathematical Society Lecture Note Series, no. 317, Cambridge University Press, 2005.
6. E. Brier and M. Joye, *Weierstraß elliptic curves and side-channel attacks*, Public Key Cryptography (PKC 2002) (P. Paillier and D. Naccache, eds.), Lecture Notes in Computer Science, vol. 2274, Springer-Verlag, 2002, pp. 335–345.
7. M. Brown, D. Hankerson, J. López, and A. Menezes, *Software implementation of the NIST elliptic curves over prime fields*, Topics in Cryptology − CT-RSA 2001 (D. Naccache, ed.), Lecture Notes in Computer Science, vol. 2020, Springer-Verlag, 2001, pp. 250–265.
8. M. Ciet, M. Joye, K. Lauter, and P.L. Montgomery, *Trading inversions for multiplications in elliptic curve cryptography*, Designs, Codes and Cryptography **39** (2006), no. 2, 189–206.
9. H. Cohen, *A course in computational algebraic number theory*, Graduate Texts in Mathematics, vol. 138, Springer-Verlag, 1993.
10. H. Cohen and G. Frey (eds.), *Handbook of elliptic and hyperelliptic curve cryptography*, Chapman & Hall/CRC, 2006.
11. J.-S. Coron, *Resistance against differential power analysis for elliptic curve cryptosystems*, Cryptographic Hardware and Embedded Systems (CHES '99) (Ç.K. Koç and C. Paar, eds.), Lecture Notes in Computer Science, vol. 1717, Springer-Verlag, 1999, pp. 292–302.
12. E. De Win, Serge Mister, B. Preneel, and M.J. Wiener, *On the performance of signature schemes based on elliptic curves*, Algorithmic Number Theory (ANTS-III) (J. Buhler, ed.), Lecture Notes in Computer Science, vol. 1423, Springer-Verlag, 1998, pp. 252–266.

13. A. Durand, *Efficient ways to implement elliptic curve exponentiation on a smart card*, Smart card research and applications (J.-J. Quisquater and B. Schneier, eds.), Lecture Notes in Computer Science, vol. 1820, Springer-Verlag, 2000, pp. 357–365.
14. K. Eisenträger, K. Lauter, and P.L. Montgomery, *Fast elliptic curve arithmetic and improved Weil pairing evaluation*, Topics in Cryptology − CT-RSA 2003 (M. Joye, ed.), Lecture Notes in Computer Science, vol. 2612, Springer-Verlag, 2003, pp. 343–354.
15. K. Fong, D. Hankerson, J. López, and A. Menezes, *Field inversion and point halving revisited*, Tech. Report CORR 2003-18, CACR, University of Waterloo, 2003.
16. P.-A. Fouque and F. Valette, *The doubling attack − why upwards is better than downwards*, Cryptographic Hardware and Embedded Systems − CHES 2003 (C.D. Walter, Ç.K. Koç, and C. Paar, eds.), Lecture Notes in Computer Science, vol. 2779, Springer-Verlag, 2003, pp. 269–280.
17. G. Gaubatz, E. Savaş, and B. Sunar, *Sequential circuit design for embedded cryptographic applications resilient to adversarial faults*, IEEE Transactions on Computers, to appear.
18. Christophe Giraud, *Fault resistant RSA implementation*, Second Workshop on Fault Detection and Tolerance in Cryptography (Edinburgh, UK), September 2, 2005, pp. 142–151.
19. D.M. Gordon, *A survey of fast exponentiation methods*, Journal of Algorithms **27** (1998), no. 1, 129–146.
20. J. Guajardo and C. Paar, *Efficient algorithms for elliptic curve cryptosystems*, Advances in Cryptology − CRYPTO '97 (B. Kaliski Jr., ed.), Lecture Notes in Computer Science, vol. 1294, Springer-Verlag, 1997, pp. 342–356.
21. D. Hankerson, J. López, and A. Menezes, *Software implementation of elliptic curve cryptography over binary fields*, Cryptographic Hardware and Embedded Systems − CHES 2000 (Ç.K. Koç and C. Paar, eds.), Lecture Notes in Computer Science, vol. 1965, Springer-Verlag, 2000, pp. 1–24.
22. D. Hankerson, A. Menezes, and S. Vanstone, *Guide to elliptic curve cryptography*, Springer, 2004.
23. K. Itoh, T. Izu, and M. Takenaka, *Address-bit differential power analysis of cryptographic schemes OK-ECDH and OK-ECDSA*, Cryptographic Hardware and Embedded Systems − CHES 2002 (B.S. Kaliski Jr., Ç.K. Koç, and C. Paar, eds.), Lecture Notes in Computer Science, vol. 2523, Springer-Verlag, 2002, pp. 129–143.
24. ———, *Efficient countermeasures against power analysis for elliptic curve cryptosystems*, Smart Card Research and Advanced Applications VI (J.-J. Quisquater et al., eds.), Kluwer Academic Publishers, 2004, pp. 99–113.
25. T. Izu and T. Takagi, *A fast parallel elliptic curve multiplication resistant against side channel attacks*, Public Key Cryptography (PKC 2002) (P. Paillier and D. Naccache, eds.), Lecture Notes in Computer Science, vol. 2274, Springer-Verlag, 2002, pp. 280–296.
26. M. Joye and S.-M. Yen, *The Montgomery powering ladder*, Cryptographic Hardware and Embedded Systems − CHES 2002 (B.S. Kaliski Jr., Ç.K. Koç, and C. Paar, eds.), Lecture Notes in Computer Science, vol. 2523, Springer-Verlag, 2002, pp. 291–302.
27. T. Kobayashi, H. Morita, K. Kobayashi, and F. Hoshino, *Fast elliptic curve algorithm combining Frobenius map and table reference to adapt to higher characteristic*, Advances in Cryptology − EUROCRYPT '99 (J. Stern, ed.), Lecture Notes in Computer Science, vol. 1592, Springer-Verlag, 1999, pp. 176–189.

28. P. Kocher, J. Jaffe, and B. Jun, *Differential power analysis*, Advances in Cryptology − CRYPTO '99 (M. Wiener, ed.), Lecture Notes in Computer Science, vol. 1666, Springer-Verlag, 1999, pp. 388–397.
29. C.H. Lim, *A new method for securing elliptic scalar multiplication against side-channel attacks*, Information Security and Privacy (ACISP 2004), (H. Wang, J. Pieprzyk, and V. Varadharajan, eds.), Lecture Notes in Computer Science, vol. 3108, Springer-Verlag, 2004, pp. 289–300.
30. C.H. Lim and H.S. Hwang, *Fast implementation of elliptic curve arithmetic in $GF(p^n)$*, Public Key Cryptography (PKC 2000) (H. Imai and Y. Zheng, eds.), Lecture Notes in Computer Science, vol. 1751, Springer-Verlag, 2000, pp. 405–421.
31. J. López and R. Dahab, *Improved algorithms for elliptic curve arithmetic in $GF(2^n)$*, Selected Areas in Cryptography (SAC '98) (S. Tavares and H. Meijer, eds.), Lecture Notes in Computer Science, vol. 1556, Springer-Verlag, 1998, pp. 201–212.
32. ———, *Fast multiplication on elliptic curves over $GF(2^m)$ without precomputation*, Cryptographic Hardware and Embedded Systems (CHES '99) (Ç.K. Koç and C. Paar, eds.), Lecture Notes in Computer Science, vol. 1717, Springer-Verlag, 1999, pp. 316–327.
33. H. Mamiya, A. Miyaji, and H. Morimoto, *Efficient countermeasures against RPA, DPA, and SPA*, Cryptographic Hardware and Embedded Systems − CHES 2004 (M. Joye and J.-J. Quisquater, eds.), Lecture Notes in Computer Science, vol. 3156, Springer-Verlag, 2004, pp. 343–356.
34. A.J. Menezes, P.C. van Oorschot, and S.A. Vanstone, *Handbook of applied cryptography*, CRC Press, 1997.
35. B. Möller, *Securing elliptic curve point multiplication against side-channel attacks*, Information Security − ISC 2001 (G.I. Davida and Y. Frankel, eds.), Lecture Notes in Computer Science, vol. 2200, Springer-Verlag, 2001, pp. 324–334.
36. P.L. Montgomery, *Speeding the Pollard and elliptic curve methods of factorization*, Mathematics of Computation **48** (1987), no. 177, 243–264.
37. K. Nguyen, *Curve based cryptography: The state of the art in smart card environments*, 6th Workshop on Elliptic Curve Cryptography (ECC 2002), Essen, Germany, September 23-25, 2002.
38. K. Okeya, H. Kurumatani, and K. Sakurai, *Elliptic curves with Montgomery form and their cryptographic applications*, Public Key Cryptography (PKC 2000) (H. Imai and Y. Zheng, eds.), Lecture Notes in Computer Science, vol. 1751, Springer-Verlag, 2000, pp. 238–257.
39. K. Okeya and T. Takagi, *The width-w NAF method provides small memory and fast elliptic scalar multiplications secure against side channel attacks*, Topics in Cryptology − CT-RSA 2003 (M. Joye, ed.), Lecture Notes in Computer Science, vol. 2612, Springer-Verlag, 2003, pp. 328–334.
40. D. Page and F. Vercauteren, *Fault and side-channel attacks on pairing based cryptography*, Cryptology ePrint Archive, Report 2004/283, 2004, `http://eprint.iacr.org/2004/283/`.
41. E. Savaş and Ç.K. Koç, *Architectures for unified field inversion with applications in elliptic curve cryptography*, 9th International Conference on Electronics, Circuits and Systems − ICECS 2002, vol. 3, IEEE Press, 2002, pp. 1155–1158.
42. R. Schroeppel, H. Orman, S. O'Malley, and O. Spatschek, *Fast key exchange with elliptic curve systems*, Advances in Cryptology − CRYPTO '95 (D. Coppersmith, ed.), Lecture Notes in Computer Science, vol. 963, Springer-Verlag, 1995, pp. 43–56.

43. N. Smart, *A comparison of different finite fields for elliptic curve cryptosystems*, Computers and Mathematics with Applications **42** (2001), 91–100.
44. M. Stam, *On Montgomery-like representations for elliptic curves over* $GF(2^k)$, Public Key Cryptography − PKC 2003 (Y. Desmedt, ed.), Lecture Notes in Computer Science, vol. 2567, Springer-Verlag, 2003, pp. 240–253.
45. ———, *Speeding up subgroup cryptosystems*, Ph.D. thesis, Technische Universiteit Eindhoven, Eindhoven, 2003.
46. N. Thiérault, *SPA resistant left-to-right integer recodings*, Selected Areas in Cryptography − SAC 2005 (B. Preneel and S.E. Tavares, eds.), Lecture Notes in Computer Science, vol. 3897, Springer-Verlag, 2006, pp. 345–358.
47. C.D. Walter, *Sliding windows succumbs to big Mac attack*, Cryptographic Hardware and Embedded Systems − CHES 2001 (Ç.K. Koç, D. Naccache, and C. Paar, eds.), Lecture Notes in Computer Science, vol. 2162, Springer-Verlag, 2001, pp. 286–299.
48. S.-M. Yen and M. Joye, *Checking before output may not be enough against fault-based cryptanalysis*, IEEE Transactions on Computers **49** (2000), no. 9, 967–970.

A Multiplicative Notation

When group $\mathbb{G}$ is written multiplicatively with 1 as identity element, the double-add algorithm (Algorithm 1) becomes the square-multiply algorithm (see below). Algorithms 2 and 3 are adapted analogously.

Algorithm 1″ Square-multiply exponentiation algorithm.

Input: $g \in \mathbb{G}$ and $k = (k_{t-1}, \ldots, k_0)_2 \in \mathbb{N}$
Output: $y = g^k$
$R_0 \leftarrow 1$; $R_1 \leftarrow g$
for $j = 0$ to $t - 1$ **do**
    $b \leftarrow 1 - k_j$; $R_b \leftarrow {R_b}^2\, R_{k_j}$
end for
return R_0