Randomized Mixed-Radix Scalar Multiplication

Eleonora Guerrini, Laurent Imbert, Théo Winterhalter

▶ **To cite this version:**

Eleonora Guerrini, Laurent Imbert, Théo Winterhalter. Randomized Mixed-Radix Scalar Multiplication. IEEE Transactions on Computers, 2018, 67 (3), pp.418-431. 10.1109/TC.2017.2750677 . lirmm-01587488v2

HAL Id: lirmm-01587488
https://hal-lirmm.ccsd.cnrs.fr/lirmm-01587488v2

Submitted on 6 Oct 2017

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Randomized Mixed-Radix Scalar Multiplication

Eleonora Guerrini, Laurent Imbert, and Théo Winterhalter

Abstract—A set of congruence relations is a $\mathbb{Z}$-covering if each integer belongs to at least one congruence class from that set. In this paper, we first show that most existing scalar multiplication algorithms can be formulated in terms of covering systems of congruences. Then, using a special form of covering systems called exact n-covers, we present a novel uniformly randomized scalar multiplication algorithm with built-in protections against most passive side-channel attacks. Our algorithm randomizes the addition chain using a mixed-radix representation of the scalar. Its reduced overhead and purposeful robustness could make it a sound replacement to several conventional countermeasures. In particular, it is significantly faster than Coron's scalar blinding technique for elliptic curves when the choice of a particular finite field tailored for speed compels to double the size of the scalar, hence the cost of the scalar multiplication.

Index Terms—Scalar multiplication, side-channel attacks, randomized algorithms, covering systems of congruences, mixed-radix number system

1 Introduction

Exponentiation in multiplicative subgroups of finite fields and jacobians of low genus hyperelliptic curves is a crucial operation for many public key cryptosystems. For instance, it is used extensively in the generation/verification of electronic signatures (e.g. using DSA/ECDSA) and in the encryption/decryption phases of RSA or DL-based algorithms. In general, data manipulated during these computations should absolutely be kept secret as even a small amount of information may be maliciously exploited by an attacker, e.g. for forging one's signature or for acquiring some confidential information. This constraint appears to be much harder to satisfy than one might expect. For the past twenty years, following the pioneer work of Kocher [1] on *Side-Channel Attacks* (SCA), it has been a designer's nightmare and an extraordinary playground for researchers.

Side-channel attacks come in many different flavours and constitute nowadays a vast arsenal for the attackers. At the higher level, one usually considers distinctly *active attacks* and *passive attacks*. Those of the first kind are usually invasive and require physical access to the cryptographic device. They try to modify the behaviour of a cryptographic algorithm using various sources of perturbation such as laser beams, clock jitters or disturbance voltage. The usual countermeasures consist in checking the cryptographic protocols for faults [2]. The second kind of attacks may or may not require physical access to the device. They aim at measuring some well-chosen physical information (power consumption, electromagnetic emanations, computation time, etc.) that leak from the device during sensitive computations in the hope that these observations will reveal (part of) some secret data. In turn, this large family of passive attacks splits into: *simple attacks* which only require one or a small number of executions in order to recover the secret (e.g. *Simple Power Analysis* [1]), and *advanced attacks* which need a very large number of observations and the use of statistical tools (e.g. *Timing attacks* [1], *Differential Power Analysis* [3], *template attacks* [4]). Simple attacks are usually defeated using highly regular algorithms, whereas advanced attacks may be thwarted using various randomization techniques. Advanced statistics are also at the core of *Horizontal attacks* [5], [6] but unlike the above-mentioned advanced attacks, they only require a unique trace, making classical randomization techniques ineffective.

In parallel to the discovery of these attacks, a lot of research has been conducted towards designing clever and efficient countermeasures at various levels. As a result, most of today's publicly known SCA can be counteracted, at least when considered individually. However, in order to safeguard implementations against all known attacks, several countermeasures must be carefully stacked together, while ensuring that this combination of independent, yet good countermeasures does not weaken the overall implementation. Inevitably, each of these protection layers implies some overhead, e.g. computation time, circuit area, etc. Hence, low-cost solutions and protections which impede several attacks at once should be considered with great interest. The ultimate, all-in-one, protection is yet to be discovered!

The main contribution of this work is a novel elliptic curve scalar multiplication algorithm with built-in protections against differential and correlation attacks, attacks based on the hidden Markov model, timing attacks, SPA-type attacks and horizontal collision correlation attacks. It offers, by design, a high level of randomization thanks to the proper use of exact *covering systems of congruences*. To the best of our knowledge, this is the first algorithm which hindes so many different types of attacks at once. As a side contribution, we show that many algorithms from the literature can be expressed in the same framework. We assess the robustness of our solution by showing that all known relevant attacks remain unsuccessful.

1.1 Randomization as a countermeasure

As stated above, the vast majority of advanced attacks require multiple executions of the algorithm. They can be circumvented using various randomization techniques. In the context of elliptic curves for example, several randomization options

- *E. Guerrini and L. Imbert are with the LIRMM, CNRS, Université de Montpellier, France.*
- *T. Winterhalter is with the ENS Cachan, Université Paris-Saclay, France.*

Manuscript received October 28, 2016.

(scalar randomization, base point blinding, random projective coordinates) were proposed by Coron in [7]. These techniques are commonly used to counteract differential power/EM analysis, refined power analysis, zero-value analysis and other types of advanced attacks. For a good survey on securing ECC implementations, see [8], [9].

A less frequent alternative to Coron's countermeasures is the randomization of the scalar multiplication algorithm itself. This can be achieved by taking random decisions in the course of the algorithm. This approach is not new ; the MIST algorithm by Walter [10] or the Leak Resistant Arithmetic (LRA) concept by Bajard et al. [11] are two examples of such randomized algorithms proposed in the RSA context. In the elliptic curve setting, Oswald and Aigner proposed the use of randomized addition-subtraction chains [12]. Their solution was broken using the so-called hidden Markov Model (HMM) cryptanalysis by Karlof and Wagner [13]. Another randomization approach of the same kind was proposed by Ha and Moon [14]. Their solution based on Binary Signed Digit (BSD) recodings [15] was broken in [16]. More recently, Méloni and Hasan generalized the fractional w-NAF method by allowing random choices for the expansion digits [17].

Most of these randomization strategies were rather elementary and have logically been broken. Nevertheless, we believe that there are still good reasons to pursue investigations on robust randomized algorithms. Quite surprisingly, a first motivation concerns practical efficiency. In the most prominent randomization technique proposed by Coron, the private scalar k is masked by adding a random multiple of the group order. Thus, instead of computing $[k]P$ for a point P on the elliptic curve E, one evaluates $[k + rN]P$, where r is a random integer and N is the order of E. The bits of k are thus masked using a different random value at each execution. In [7], Coron suggested a nowadays far too small 20-bit random value. The question regarding the "good" size for r is crucial and deserves some thought. For the sake of efficiency, many elliptic curves are defined over special finite fields, for example modulo primes that are very close to a power of two. In this case, due to the Hasse bound, the order N of the curve is also close to that same power of two. Thus, in order to blind all the bits of k, the size of r should not be too small.

As an example, consider curve25519, a record-breaking elliptic curve in Montgomery form proposed by Bernstein in [18]. It is defined by the equation $y^2 = x^3 + 486662x^2 + x$, over the 255-bit prime field with $2^{255}-19$ elements. The binary expansion of the order N of curve25519 reveals a sequence of 127 consecutive zeros:

$$N = \texttt{0x1000000000000000000000000000000}\ldots$$
$$\ldots\texttt{14def9dea2f79cd65812631a5cf5d3ed}$$

Let $r = \texttt{0xa3787bb9c16b2b7d}$, be a 64-bit random integer. The binary expansion of rN also contains a long sequence of zeros:

$$rN = \texttt{0xa3787bb9c16b2b7d0000000000000000d53cbb}\ldots$$
$$\ldots\texttt{429156e3da99b82c008afa3bcec4b0646efb049b9}$$

Thus, when a secret scalar, say

$$k = \texttt{0x9db98047d18f8e8505b55abad0ea873a}\ldots$$
$$\ldots\texttt{1080c69a3ab0755b2e8dfb1e939b3fd}$$

is added to rN, the binary expansion of the result discloses both the bits of r and the most significant bits of k:

$$k + rN = \texttt{0xa3787bb9c16b2b7d|9db98047d18f8e85|}\ldots$$
$$\ldots\texttt{daf215fd62416b14aa38f29ac5aab129f33e5f8d8e9fdb6}$$

Observe that even if r was chosen twice as large, although rN would not contain a sequence of consecutive zeros, the binary expansion of r could entirely be revealed by a simple analysis, rendering the randomization ineffective.

For that reason, Bernstein recommends a 256-bit random value, for curve25519. The same guidance holds for many other elliptic curves, in particular most of the curves meeting all SafeCurves requirements [19]: M-221, E-222, curve1174, E-382, M-383, curve383187, curve41417, M-511, E-521 . It also applies to the elliptic curves in the Weierstrass and Edwards models recently introduced by Bos, Costello, Longa and Naehrig in [20]. Both those defined modulo pseudo-Mersenne primes, called `w-xxx-mers` and `ed-xxx-mers` for `xxx` $\in \{255, 256, 383, 384, 511, 512\}$ and those defined modulo primes of the form $2^{\alpha}(2^{\beta} - \gamma) - 1$ denoted `w-xxx-mont` and `ed-xxx-mont` for `xxx` $\in \{254, 256, 382, 384, 510, 512\}$ are concerned.

For all those curves, choosing a random value r of size the size of p yields a 100% overhead for the scalar multiplication! With this in mind, sturdy solutions based on randomizing the addition chain may provide efficient, acceptable alternatives.

A second good reason to investigate novel alternative solutions is dictated by the history of cryptography. How many algorithms have eventually been broken after being advocated as perfectly secure? In the recent years, quite a few very powerful attacks were introduced which broke implementations that were thought unscathed thanks to widely acknowledged countermeasures. In [6] for example, Bauer et al. introduced the so-called *horizontal collision correlation analysis* and defeated ECC implementations that were considered secure until then. In particular, their attack is effective against various atomic schemes [21], [22], [23] and unified formulae [24], [25], two well established countermeasures against SPA. What is worth noticing about this attack, is that it refutes one of the most established claim that both atomicity and unified formula do prevent from differentiating a point addition from a point doubling. Even more recently, Nascimento et al. [26] presented a powerful attack against a constant-time Montgomery ladder implementation with projective point randomization. Their attack directly targets the cmov (conditional move) operation and only requires a single trace. Once again, a well-known assumption has been turned down.

2 A new randomized scalar multiplication

In this section, we present a novel randomized mixed-radix scalar multiplication algorithm. It is based on some nice properties of so-called *covering systems of congruences*.

2.1 Covering systems of congruences

A *covering system of congruences* (CSC) is a finite set

$$\mathcal{S} = \{r_1 \pmod{m_1}, \ldots, r_t \pmod{m_t}\}$$

such that every integer satisfies at least one congruence relations from $\mathcal{S}$. In general, such covering sets are not difficult

to construct. A simple, non-trivial example of a *distinct* (all moduli are different) covering system is given by the set:

$$\{0 \pmod 2;\ 0 \pmod 3;\ 1 \pmod 4;\ 1 \pmod 6;\ 11 \pmod{12}\}.$$

In the following, we shall conveniently use the compact expressions $r\,(m)$ or (r, m) to denote the congruence relation $r \pmod m$. And when a CSC contains several congruence relations modulo the same modulus m, we shall use $r_0, \ldots, r_w\,(m)$ in place of $r_0 \pmod m, \ldots, r_w \pmod m$.

A covering system is called an *n-cover* if each integer is covered at least n times ; it is called an *exact n-cover* if each integer is covered *exactly* n times. For example, the set:

$$\{1\,(2);\ 2, 3\,(4);\ 0, 2, 4\,(6);\ 0, 1, 4, 5\,(8)\} \tag{1}$$

form an exact 2-cover. It is illustrated in Fig. 1.

It is easy to prove that the covering property is guaranteed as soon as all the integers modulo $\ell = \mathrm{lcm}(m_1, \ldots, m_t)$ are covered. In the example given in (1), $\ell = \mathrm{lcm}(2, 4, 6, 8) = 24$.

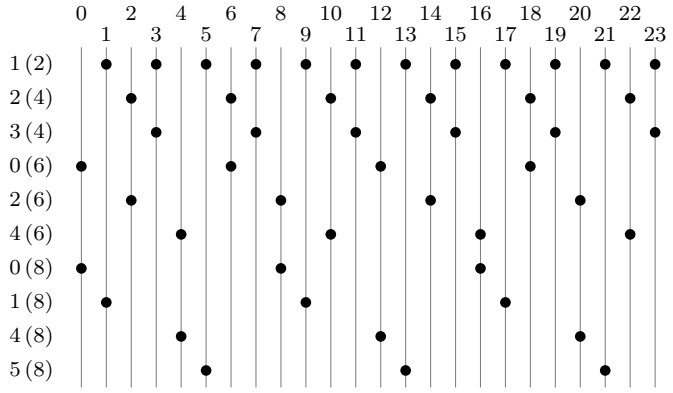


Figure 1. The set $\{1\,(2);\ 2, 3\,(4);\ 0, 2, 4\,(6);\ 0, 1, 4, 5\,(8)\}$ is an exact 2-cover. Each row corresponds to a congruence class. The dots indicate which integers are covered by each class. The number of dots in each column indicates the number of congruence classes covering the corresponding integer. One may observe that there are exactly 2 dots per column.

Problems concerning covering systems of congruences were among Erdős' favorites. One of his conjectures, the minimum modulus problem, was only solved negatively by Hough in 2013 using the Lovász local lemma [27]. The non-existence of a distinct covering system whose moduli are all odd is still an open problem.

2.2 CSC-based scalar multiplication

The link between covering systems and scalar multiplication algorithms is immediate. Let E an elliptic curve, P a point on E, and $k \in \mathbb{Z}$. Then it is clear that

$$k \equiv r \pmod m \Rightarrow [k]P = [r]P + [m]([(k - r)/m]P). \tag{2}$$

For example, the right-to-left double-and-add algorithm follows directly from the exact 1-cover given by the trivial covering set $\{0 \pmod 2;\ 1 \pmod 2\}$:

$$[k]P = \begin{cases} [k/2]([2]P) & \text{if } k \text{ is even} \\ [(k-1)/2]([2]P) + P & \text{if } k \text{ is odd} \end{cases}$$

In the world of (hyper)elliptic curves, many scalar multiplication algorithms have been proposed (m-ary, NAF, window methods, multi-base, etc.). Interestingly, many of them can be expressed using covering system of congruences. We list a few and the corresponding covering systems of congruences in Table 1.

Table 1
Scalar multiplication algorithms and their corresponding covering system of congruences

Algorithm	Covering System of Congruences
m-ary	$\{0, \ldots, m-1\,(m)\}$
NAF	$\{0\,(2);\ 1, -1\,(4)\}$
w-NAF [28]	$\{0\,(2);\ 1, -1, 3, -3, \ldots, 2^{w-1}-1, -2^{w-1}+1\,(2^w)\}$
frac. win. [29]	$\{0\,(2);\ 1, -1, 3, -3, \ldots, 2^w+m, -2^w-m\,(2^{w+1})\}$
wmbNAF [30]	$\{0\,(a_1);\ \ldots; 0\,(a_k); 1, -1, 3, -3, \ldots, (a_1-1)/2\,(a_1^w)\}$
w-HBTF [31]	$\{0\,(2);\ 0\,(3);\ 1, -1, \ldots, w/2-1, -w/2+1\,(w)\}$

2.2.1 *A generic algorithm:*

Let $\mathcal{S}$ be a covering system of congruences. For all $k \in \mathbb{Z}$, we denote by $\mathcal{S}_{(k)}$ the set of all the congruence classes covering k, i.e. the set of all the congruences relations $r \pmod m \in \mathcal{S}$ such that $k \equiv r \pmod m$. A *generic* CSC-based scalar multiplication is given in Fig. 2.

Input: $\mathcal{S}$, $k \in \mathbb{Z}, P \in E$
Output: $[k]P \in E$
1: **if** $k = 0$ **then**
2: **return** P_∞
3: Let $r \pmod m \in \mathcal{S}_{(k)}$
4: Compute $Q := [(k - r)/m]P$ recursively
5: **return** $[m]Q + [r]P$

Figure 2. Generic CSC-based scalar multiplication

The validity of the generic algorithm in Fig. 2 is based on the fact that, for any covering system $\mathcal{S}$, we have

$$[m]Q + [r]P = [m(k - r)/m]P + [r]P = [k]P. \tag{3}$$

2.2.2 *A specialized version with built-in SC protections:*

Clearly, equation (3) is true regardless of how the congruence class $(r, m) \in \mathcal{S}_{(k)}$ is selected in line 3. Note that when $|\mathcal{S}_{(k)}| > 1$, i.e. when integer k is covered by strictly more than one congruence class, several algorithms can be deduced depending on which class is chosen. Therefore, the generic algorithm presented in Fig. 2 may be specialized in various ways. The main characteristics of our randomized version are twofold:

- First, we require that the covering system $\mathcal{S}$ is an exact n-cover (with $n \geq 2$). By doing so, we ensure that there are exactly n congruence classes $r \pmod m$ in $\mathcal{S}_{(k)}$ for every k.
- Second, in line 3 of Fig. 2, we select the congruence class $r \pmod m \in \mathcal{S}_{(k)}$ uniformly at random among the n different options. In practice, these n congruence classes in $\mathcal{S}_{(k)}$ are easily determined by computing $k \bmod \ell$, where $\ell = \mathrm{lcm}(m_1, \ldots, m_{|\mathcal{S}|})$. In Section 3.1 we shall see that this leads to exponentially many ways to compute Q in line 4.

An exact covering system $\mathcal{S}$ may be represented using a convenient data structure very close to the graphical representation shown in Fig. 1. More precisely, if

$$\mathcal{S} = \{r_1 \pmod{m_1};\ r_2 \pmod{m_2};\ \ldots;\ r_t \pmod{m_t}\}$$

is an exact n-cover, it may be described as a two-dimensional array of size $\ell \times n$, where $\ell = \mathrm{lcm}(m_1, \ldots, m_t)$. For each $i \in \{1, \ldots, \ell\}$, the entry $S[i]$ is a one-dimensional array, of length exactly n, whose elements are the congruence classes (r, m) covering the subset $i + \ell\mathbb{Z}$. An example is given in Table 2.

Table 2
The array representation of the covering system $\{1 \pmod 2;\ 2, 3 \pmod 4;\ 0, 2, 4 \pmod 6;\ 0, 1, 4, 5 \pmod 8\}$.

i	$S[i][0]$	$S[i][1]$	i	$S[i][0]$	$S[i][1]$
0	$(0, 8)$	$(0, 6)$	12	$(4, 8)$	$(0, 6)$
1	$(1, 8)$	$(1, 2)$	13	$(5, 8)$	$(1, 2)$
2	$(2, 4)$	$(2, 6)$	14	$(2, 4)$	$(2, 6)$
3	$(1, 2)$	$(3, 4)$	15	$(1, 2)$	$(3, 4)$
4	$(4, 8)$	$(4, 6)$	16	$(0, 8)$	$(4, 6)$
5	$(5, 8)$	$(1, 2)$	17	$(1, 8)$	$(1, 2)$
6	$(2, 4)$	$(0, 6)$	18	$(2, 4)$	$(0, 6)$
7	$(1, 2)$	$(3, 4)$	19	$(1, 2)$	$(3, 4)$
8	$(0, 8)$	$(2, 6)$	20	$(4, 8)$	$(2, 6)$
9	$(1, 8)$	$(1, 2)$	21	$(5, 8)$	$(1, 2)$
10	$(2, 4)$	$(4, 6)$	22	$(2, 4)$	$(4, 6)$
11	$(1, 2)$	$(3, 4)$	23	$(1, 2)$	$(3, 4)$

We use this data structure in the description of the exact n-cover scalar multiplication presented in Fig. 3.

Input: S as described above, $\ell = \mathrm{lcm}(m_1, \ldots, m_{|\mathcal{S}|})$, $k \in \mathbb{N}, P \in E$
Output: $[k]P \in E$
1: **if** $k = 0$ **then**
2: **return** P_∞
3: **else if** $k = 1$ **then**
4: **return** P
5: $i := k \bmod \ell$
6: Select j uniformly at random in $\{0, \ldots, n-1\}$
7: $(r, m) := S[i][j]$
8: compute $R := [r]P$
9: compute $Q := [(k - r)/m]P$ recursively
10: **return** $[m]Q + R$

Figure 3. Exact n-cover scalar multiplication

For convenience, we presented a recursive version of our algorithm in Fig. 3. However this recursion can easily be reformulated iteratively. It can thus be implemented on small embedded devices which may not support recursion. To do so, first compute a sequence (r_i, m_i) for k using the covering system of your choice such that

$$k = r_0 + m_0(r_1 + m_1(r_2 + \cdots + (r_{s-1} + m_{s-1}r_s)\ldots)). \quad (4)$$

The above representation of k is known as a *mixed-radix representation*. Once k is converted in that form, computing $[k]P$ simply consists of traversing this sequence $(r_i, m_i)_{i \geq 0}$ in reverse order. Initialize $Q := P_\infty$, then for each pair (r_i, m_i), set $Q := [m_i]Q + [r_i]P$. Note that the elements $[r_i]P$ may be precomputed. Generating the sequence $(r_i, m_i)_{i \geq 0}$ from k only requires integer arithmetic (see Section 2.4).

2.3 Complexity analysis

In this section, we analyze the asymptotic average complexity of our randomized mixed-radix algorithm presented in Fig. 3 using a first order Markov chain. Let $\mathcal{S} = \{s_1, \ldots, s_t\}$ with $s_i := r_i \pmod{m_i}$ and let $\ell = \mathrm{lcm}(m_1, \ldots, m_t)$. We define the transition graph of the Markov chain as follows:

- the set of vertices $V = \{v_0, \ldots, v_{\ell-1}\}$ is the set of congruence classes modulo ℓ. By convention, v_i denotes the class of $i \pmod \ell$.
- the edges $(v_{r_i}, v_{r_j}) \in V^2$ are oriented and labeled with probabilities: for every $k > 0$, there are exactly[1] n congruence classes $s_i := r_i \pmod{m_i}$ in $\mathcal{S}_{(k)}$ such that $k \equiv r_i \pmod{m_i}$. In line 9 of Fig. 3 (or line 4 of Fig. 2), the algorithm is called recursively with $k' = (k - r_i)/m_i$ as input. In turn, if $k' > 0$, we select a congruence class $s_j := r_j \pmod{m_j} \in \mathcal{S}_{(k')}$. The edge (v_{r_i}, v_{r_j}) is labeled with the conditional probability $P(k' \equiv r_j \pmod{m_j} | k \equiv r_i \pmod{m_i})$.

The following lemma will come handy in the following.

Lemma 1. *The Markov chain associated to a covering set $\mathcal{S}$ is irreducible and aperiodic.*

Proof: For every $0 \leq k < \ell$, our algorithm terminates. Thus, for each $v_i \in V$, there is a path from v_i to v_0. Let T denote the transformation $T : k \mapsto (k - r)/m$ for $k \equiv r \pmod m$. Let $k \equiv 0 \pmod \ell$. After one step of the algorithm, i.e. after a division by m, we have $P(T(k) \equiv 0 (\mathrm{mod} \ell/m)) > 0$. Thus, after a finite number of steps, there exists j such that $T^{(j)}(k) \equiv 0 \pmod m$ for $m \in \mathcal{S}$. Hence, $\forall i \in \{0, \ldots, \ell - 1\}$, $P(T^{(j+1)}(k) \equiv i \pmod \ell) = 1/\ell > 0$. The Markov chain is thus irreducible. For every $k > 0$, $P(k' \equiv 0 \pmod \ell | k \equiv 0 \pmod \ell) > 0$. Therefore, the Markov chain contains at least one cycle of length 1, given by the edge (v_0, v_0). □

In the general case we need to compute the transition matrix, say A, associated to $\mathcal{S}$, and then evaluate its stationary probability, i.e. the vector π_∞ such that $\pi_\infty A = \pi_\infty$. (Note that the uniqueness of π_∞ is a consequence of Lemma 1.) When $\mathcal{S}$ is an exact n-cover however, neither A nor π_∞ need be computed. The following theorem holds.

Theorem 1. *The stationary probability obtained for an exact n-cover is uniform:*

$$\pi_\infty = (1/\ell, \ldots, 1/\ell).$$

The proof, given for completeness below, is a direct consequence of the following Lemma. Indeed, when A is doubly stochastic, $\mathbb{1}A = \mathbb{1}$. Thus, the uniform probability distribution $\pi = (1/\ell)\mathbb{1}$ satisfies $\pi A = \pi$.

Lemma 2. *The transition matrix associated to an exact n-cover is doubly stochastic, i.e. each row and column adds up to 1.*

1. Note that when $\mathcal{S}$ is a covering set but not an exact n-cover, there still exists at least one such class.

Proof: Let $\mathcal{S}$ be an exact n-cover and A its transition matrix. We want to show that $\sum_i A_{i,j} = 1$ for all $j = 0, \ldots, \ell - 1$. We will do so by showing that for all $j = 0, \ldots, \ell - 1$

$$\sum_i A_{i,j} = \sum_{(r,m)\in\mathcal{S}} \frac{1}{nm} = \frac{1}{n} \sum_{(r,m)\in\mathcal{S}} \frac{1}{m}. \tag{5}$$

Indeed, since each integer is covered exactly n times and each covering class covers exactly ℓ/m integers, we get that $\sum_{(r,m)\in\mathcal{S}} \frac{1}{m} = n$.

Now, in order to prove (5), observe that for each congruence class $(r, m) \in \mathcal{S}$, there are exactly ℓ/m integers i in $\{0, \ldots, \ell - 1\}$ that are covered by (r, m). For each of those, there are exactly m integers j in $\{0, \ldots, \ell - 1\}$ such that $j \equiv (i - r)/m \pmod{\ell/m}$. Thus, each $(r, m) \in \mathcal{S}$ contributes to exactly ℓ columns (not necessarily distinct) of A. To prove that there are no zero-column, observe that for $(r, m) \in \mathcal{S}$, letting $i = (r + m(j \bmod \ell/m)) \bmod \ell$, we get that $i \in (r, m)$, $0 \le i < \ell$ and $j \equiv (i - r)/m \pmod{\ell/m}$ for all $j \in \{0, \ldots, \ell - 1\}$. This confirms that each congruence class $(r, m) \in \mathcal{S}$ contributes to each column exactly once. Finally, since $\mathcal{S}$ is an exact n-cover, $|\mathcal{S}_{(i)}| = n$ for all $i = 0, \ldots, \ell - 1$ so that each contribution amounts to $1/nm$. □

Let us now explain how this uniform stationary probability can be turned into an average operation count per bit. We shall first compute the average number of point doublings, triplings, quintuplings, etc., as well as the average number of point additions per iteration; then per bit. We will then put together these numbers with the cost of each curve operation to get the average number of field operations per bit.

Let $\sigma = ((r_i, m_i))_{i=0\ldots s}$, be a precomputed sequence for the given scalar k. The iterative version of Algorithm 3 rewrites: set $Q = P_\infty$ and repeat $Q = [m_i]Q + [r_i]P$, for i ranging from s down to 0. For each iteration, we aim at computing the average number of point additions, and for each m in $\mathcal{S}$, the average number of scalar multiplications by m. More precisely, for the later we seek the average number of multiplications by p for each prime factor p in the prime decomposition of m.

First, observe that at each step, i.e. for each $(r_i, m_i) \in \sigma$, a point addition is performed exactly when $r_i \neq 0$. For each congruence class $(r, m) \in \mathcal{S}$, there are exactly ℓ/m integers in $\{0, \ldots, \ell-1\}$ which belong to that class. Thus, the probability to perform a point addition is

$$P_1 = 1/\ell n \left(\sum_{(r,m)\in\mathcal{S}, r\neq 0} \ell/m \right) = 1/n \left(\sum_{(r,m)\in\mathcal{S}, r\neq 0} 1/m \right).$$

For the scalar multiplication $[m_i]Q$, several options are possible. Here, we consider the prime decomposition of m_i. For example, we compute $[12]Q = [2^2 \cdot 3]Q$ using two point doublings and one point tripling. For each $(r, m) \in \mathcal{S}$, let $m = \prod_i p_i^{\alpha_i}$ be the prime decomposition of m, so that $\alpha_i = \nu_{p_i}(m)$ is the p_i-valuation of m for all i. We denote by N_{p_i} the average number of scalar multiplications by p_i. Then, using the same arguments as above we get

$$N_{p_i} = 1/n \sum_{(r,m)\in\mathcal{S}} \nu_{p_i}(m)/m.$$

In order to convert these values to an average number of point operations per bit, one needs to evaluate the average number of iterations. A slight difficulty here comes from the fact that the scalar is not divided by the same integer at every step. If $\ell = \prod_i p_i^{\alpha_i}$, the scalar is divided by the average value $\beta = \prod_i p_i^{N_{p_i}}$ so that the average number of iterations is obtained by multiplying the bitlength of k by $\rho = \log 2/\log \beta$. The above values P_1, N_{p_i} may be scaled accordingly to get an average number of point operations (addition, doubling, tripling, etc.) per bit. Finally, by plugging in the cost of each curve operation, we get an average number of field operations per bit.

As an example, let us consider the following covering system, denoted `u3c-48-24` for $n = 3$, $\ell = 48$, $|\mathcal{S}| = 24$. (We give more details on the terminology in Section 4).

$$\begin{aligned}\mathcal{S} = \{&0 \pmod 2;\\ &-1, 0 \pmod 4;\\ &-1, 1, 3 \pmod 6;\\ &-2, -1, 0, 1 \pmod 8;\\ &-3, -2, 1, 2, 5, 6 \pmod{12};\\ &-6, -5, -4, -3, 2, 3, 4, 5 \pmod{16}\}\end{aligned}$$

We have $P_1 = 17/24$, $N_2 = 13/6$, $N_3 = 1/3$, and thus $\beta = 2^{13/6} \times 3^{1/3} \approx 6.47548$. According to the explicit formula database [32], the smallest multiplication counts for a point addition, a point doubling and a point tripling on a short Weierstrass curve (assuming $a = -3, S = 0.8M$) are $10.2M$ (with $Z_2 = 1$), $7M$ and $12.6M$ respectively. The average cost is thus $(10.2P_1 + 7N_2 + 12.6N_3) \times \log_\beta(2) \approx 9.87M$ per bit. Note that `u3c-48-24` only requires two precomputed points, namely $3P$ and $5P$. If these precomputed points are not converted to affine coordinates, the point addition costs $15M$ and average cost increases to $\approx 11.13M$ per bit.

In Table 3, we give the average operation counts for short Weierstrass curves. In comparison, the Montgomery ladder on Weierstrass curves, using the differential co-Z addition-and-doubling algorithm reported in [33, Algo. 5] costs $10M + 5S \simeq 14M$ per bit; the NAF, 3-NAF and 4-NAF algorithms: $10.4M$, $9.55M$ and $9.04M$ per bit respectively. The number of precomputations corresponds to the number of different group elements $[r]P$ that may appear. In order to save some precomputations, observe that when computing $[m]Q + [r]P$, both m and r can be divided by $h = \gcd(m, r)$; the computation thus becomes $[h]([m/h]Q + [r/h]P)$. As usual, only one of any pair of opposite points is required. For `u3c-48-24`, the only points that needs to be precomputed are thus $3P$ and $5P$.

2.4 Integer arithmetic

Our theoretical analysis does not take into account the cost of integer arithmetic. Other classical algorithms like double-and-add, fixed- and sliding-window methods, the Montgomery ladder etc. process the scalar bit-by-bit or in blocks of bits. Conversely, our algorithm needs integer division with remainder where the divisor is not a power of two.

In Algorithm 3, two operations deserve some attention: the integer division with remainder $k \bmod \ell$ in line 5 and the exact division by m in line 9. Although it is possible to build a covering system of congruences such that $\ell = \mathrm{lcm}(m_1, \ldots, m_{|\mathcal{S}|})$ factors into many different primes, it seems more advantageous—at least in the context of elliptic curve for which there exists efficient explicit formula for point doubling

Table 3
Average operation counts per bit and number of precomputed points for short Weierstrass curves. The second column gives the average cost when $3P$ is computed using the best known tripling operation; the third column gives that same average cost when no tripling operation is available, i.e. $3P$ is computed as $2P + P$ with a doubling and a full addition (mixed add does not apply). The CSC are listed from faster to slower according to the 2nd column.

CSC	mult./bit w/ Tpl.	mult./bit w/o Tpl.	#P
u12c-2304-3315	8.84	10.38	355
u8c-432-600	9.33	12.22	66
cs3-48-48	9.84	11.48	8
u3c-48-24	9.87	11.03	2
cs3-48-38	9.95	11.66	7
cs6-72-103	9.95	12.91	12
u4c-48-60	9.96	11.38	7
cs3-54-47	10.61	15.54	8
cs4-24-37	10.64	12.95	4
cs3-24-23	10.93	13.4	2
u6c-120-168	11.09	12.28	19
u2c-24-10	11.23	12.33	1
cs5-60-73	12.32	14.26	8
cs2-30-19	12.7	14.64	4

and tripling—to consider moduli that only contain powers of 2 and 3 (maybe 5). In that case, the operation $k \bmod \ell$ can be greatly sped up. For example, by independently computing the remainders modulo the largest powers of 2 and 3 (possibly 5) in ℓ and by Chinese remaindering. We point the interested reader to the very fast mod3 implementation based on historic Pascal's tapes proposed in [34]. The exact division by m in line 9 can also be implemented very efficiently, for example using Jebelean's exact division [35]. (See for example GMP's exact division by 3 in `mpn_divexact_by3`.)

For security reasons, the moduli which compose the CSC should all have the same word length ; in practice they will most probably fit into a single word. Hence, the exact divisions reduce to divisions of an n-word integer by a 1-word integer and should be indistinguishable from one another.

In total, the extra cost implied by the integer arithmetic remains negligible compared to the overall scalar multiplication. As an example, converting k to a randomized mixed-radix form with our proof-of-concept implementation represents less than 2% of the total time for a 256-bit scalar.

3 Resistance to side-channel attacks

Assessing the level of resistance of an algorithmic countermeasure against the constantly growing variety of side-channel attacks is a difficult task. In the next sections, we provide solid arguments to support the robustness of our randomized algorithm against the most prominent attacks. We assume that the goal of the attacker is to recover some fixed or ephemeral secret by observing leakage during a scalar multiplication $[k]P$. Following Kerckhoffs' principle, we consider that the attacker knows precisely the instance of Algorithm 3 she is trying to break, in particular she knows everything about $\mathcal{S}$, the covering system of congruences.

As stated in Section 2, our algorithm processes the bits of k in an indirect manner. Instead, it operates on a randomized mixed-radix representation of k:

$$k = r_0 + \sum_{i=1}^{s} r_i \prod_{j=0}^{i-1} m_j$$

Therefore, unlike classical attacks which aim at recovering the bits (digits) of k assuming that the base is known (2 or a small power of 2 in general), unveiling k in our case requires to uncover both the sequence of digits $(r_0, \ldots, r_s)$ and the sequence of bases $(m_0, \ldots, m_s)$.

Note that for HMM attacks, and simple/horizontal attacks, we only consider traces that do not include triplings. We show that scrutinizing traces composed of doublings and additions only seems impracticable. Of course, the use of triplings is possible, but one has to ensure that it does not introduce weaknesses.

3.1 Differential and correlation attacks

In [7], Coron generalized Kocher et al. [3] DPA attack to elliptic curve cryptosystems. The attack is based on the fact that there exists a correlation between the bits of k and some intermediate values computed during the scalar multiplication algorithm, and the fact that this correlation may be easily found using classical DPA techniques. As an example, Coron shows that the point $4P$ is computed if and only if the second most significant bit of k is 0. Extending the attack to any addition-subtraction chain can be done in $O((\log k)^2)$. The countermeasures proposed by Coron consist in introducing random numbers during the computation of $[k]P$, so that finding correlation between (bits of) k and some intermediate quantities becomes impractical.

Instead of randomizing the input values (secret exponent or base point), our algorithm randomizes the sequence of operations used to compute $[k]P$. Hence, finding correlation between k and some intermediate quantity should remain unfeasible. Algorithms of that sort are rather scarce. In the context of elliptic curves, a first attempt was suggested by Oswald and Agnier in 2001. In [12], they rediscovered Booth's recoding techniques for integers [36], and proposed to randomize the addition-subtraction chain using an elementary randomization of the binary signed-digit (BSD) expansion of k. Their approach was broken in 2003 using the hidden Markov model [13]. (We consider this very powerful attack in Section 3.2.) In 2002, Ha and Moon proposed an almost identical randomization strategy [14] which only differs from [12] in the way the signed-digit representation of k is computed. Logically, the entropy produced by the randomization remains unchanged and the security of the algorithm remains insufficient. A different attack by Fouque et al. was presented in [16]. In that paper Fouque et al. presented a collision attack using the fact that the probabilities of the j-th BSD digit (trit) when $k_j = k_{j+1}$ can be distinguished from the probabilities of that same digit when $k_j \neq k_{j+1}$. Although a given scalar k may have many different BSD representations, the authors of [16] proved that the number of internal states remain very small. At each step of computation, at most two intermediate values can be reached. (This is illustrated in Fig. 4 for $k = 10273$.)

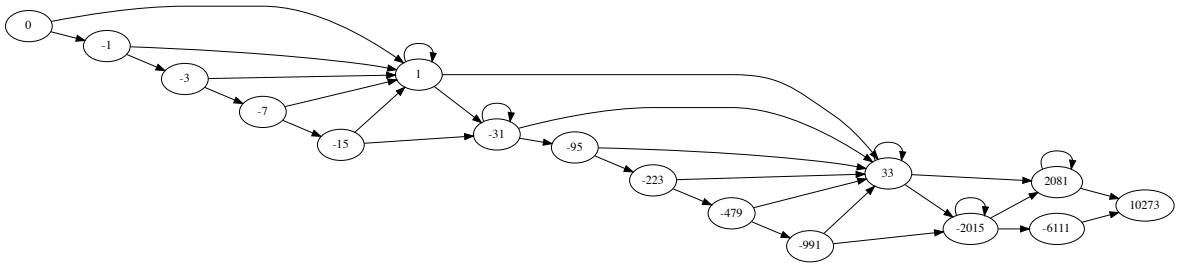


Figure 4. The graph of internal states of the BSD randomized scalar multiplication for $k = 10273$.

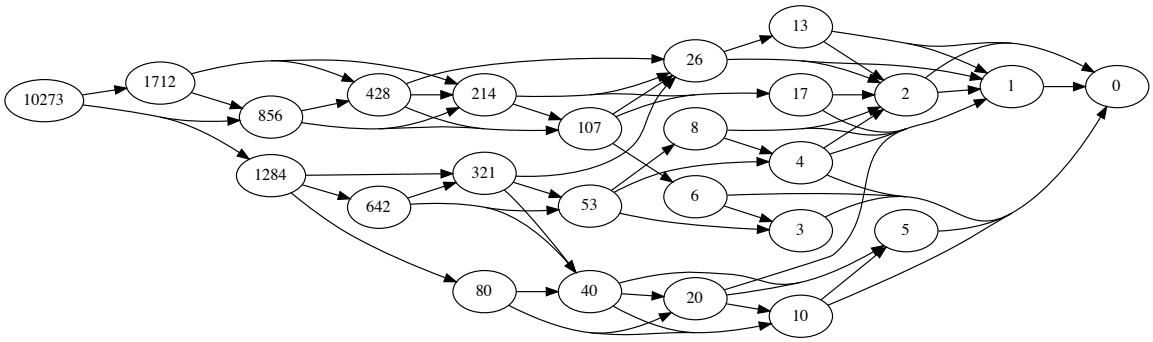


Figure 5. The graph of internal states of an exact 3-cover for $k = 10273$.

As pointed out in their conclusion: *"Any reasonable countermeasure based on randomizing the multiplication algorithm should guarantee locally a large number of possible internal states and a large number of possible transitions from each state"*.

Our algorithm does satisfy both conditions. From each internal state, there exists exactly n transitions[2] given by the n congruence classes which cover that integer. By choosing one of these n possible transitions uniformly at random, i.e. with probability $1/n$, our algorithm is locally robust. An execution trace of the algorithm corresponds to a path from k to 0 in a direct acyclic (multi)graph. In Fig. 5, we give the transition graph for $k = 10273$ obtained with an exact 3-cover. It should be compared to the graph in Fig. 4 obtained for the same scalar using the BSD randomization. The Markov analysis from Section 2.3 provides some insight into the global randomness aspects of the algorithm.

In Lemma 1, we proved the ergodicity of the Markov chain. In this case, it is known that the stationary distribution is unique and satisfies $\pi_\infty = \lim_{n\to\infty} \pi A^n$ for any probability distribution π. Notably, the stationary distribution is independent from the initial distribution. A corollary of Theorem 1 is that any random walk (of sufficiently many steps) across the transition graph of the Markov chain ends on any congruence class modulo ℓ with equal probability $1/\ell$ and independently from the starting value k.

2. For small values, although an integer, say j, is still covered by exactly n congruence classes, there might be fewer transitions (see Fig. 5 for the states 1, 2, 3, 5, 6, 10, 17). This is because when k is small, it happens that $k = r$ for some $(m, r) \in \mathcal{S}_{(j)}$.

The level of randomization of a given covering system can be evaluated by counting the number of paths from k to 0 in the transition graph corresponding to k. It is equal to $B_{k,0}$, where $B = A + A^2 + \cdots + A^v$, and where v is the length of the longest path in that direct acyclic graph (easily obtained by topological ordering). Our numerical experiments suggest that this number of paths grows exponentially in both k and the degree n of the covering system. For example, an exact 4-cover produces roughly 2^{44} paths for a 64-bit scalar and 2^{89} paths for a 128-bit scalar, whereas an exact 8-cover leads to 2^{49} and 2^{101} paths respectively for scalars of the same sizes. Even for small values of n, the number of paths seems large enough to guarantee a high level of randomization: an exact 2-cover produces 2^{61} paths for a 128-bit scalar and 2^{121} paths for a 256-bit scalar.

We think that the above analysis gives solid arguments in favour of the robustness of our randomized algorithm against differential attacks.

3.2 HMM attacks

Finite state stochastic processes may be analyzed using Hidden Markov Models (HMMs) [37], [38]. An execution of an HMM consists of a sequence of hidden, unobserved states and a corresponding sequence of related, observable outputs. HMM cryptanalysis [13], [39] aims at solving the so-called *inference problem*, i.e. infering the sequence of hidden states given only the sequence of, possibly noisy, observable outputs. This problem may be solved efficiently using the Viterbi algorithm [40]. Since our algorithm rewrites trivially as a

finite state stochastic process, it seemed natural to analyze its robustness regarding HMM attacks. To do so, we adapted and implemented the attacks from [13] and [39].

As seen in Section 2.2, an execution of our algorithm consists of s computations of the form $Q := [m_i]Q + [r_i]P$, where the loop-length s and the symbols (r_i, m_i) are given by a randomly generated mixed-radix representation of k. Given such a representation of k, the execution of the algorithm runs through a sequence $(q_0, q_1, \ldots, q_{s-1})$ of internal state. In the following, the set of all internal states of the HMM is denoted by S.

On each state $q_i \in S$, the algorithm performs a computation denoted $C(q_i)$ and outputs a value $O(q_i)$, so that $O(q_0) = [m_0]P_\infty + [r_0]P = [r_0]P$ and $O(q_i) = [m_i]O(q_{i-1}) + [r_i]P$ for $i > 0$. Note that the transition from state q_{i-1} to state q_i is uniquely determined by the pair (r_i, m_i).

A *trace* is a sequence of observations $(y_0, y_1, \ldots, y_{s-1})$, where each y_i belongs to a finite set $\mathcal{O}$ of so-called *observables*; a finite set of symbols that represent operations observable over the side-channel. For example, for the right-to-left binary scalar multiplication considered in [13], $\mathcal{O} = \{D, AD\}$, where D denotes a point doubling and A a point addition. Each element of $\mathcal{O}$ is in one-to-one correspondence with the internal states of the algorithm and with the bits of k. Improving upon the attack from [13], Green at al [39] let $\mathcal{O}$ be a set of finite words over the alphabet $\{D, A, \emptyset, \perp\}$, where $\emptyset$ and $\perp$ denote a zero-length observable and an unknown respectively. This generalization allows the authors of [39] to handle observables outputs that are not in one-to-one correspondence with the internal states ; in particular, to deal with errors in the sequence of observations. In both [13] and [39], the elements from $\mathcal{O}$ are probabilistically distinguishable from each other and each trace of the side-channel can be uniquely written as $(y_0, y_1, \ldots, y_{s-1})$ with $y_i \in \mathcal{O}$ for all $i \in \{0, \ldots, s-1\}$. We shall see later that this represents a major difference with our algorithm.

We illustrate an implementation of an HMM attack on our algorithm through an example. Let us consider the simple covering set `u2c-24-10`:

$$\begin{aligned}\mathcal{S} = \{&1 \pmod 2; -1, 2 \pmod 4;\\ &-2, 0, 2 \pmod 6; -3, 0, 1, 4 \pmod 8\}\end{aligned} \tag{6}$$

For now, we assume that the set of internal states S is equal to the set of congruence classes from $\mathcal{S}$. In order to abbreviate notations, the elements of S are given by the pairs (r, m) in place of the congruence classes $r \pmod m$ from $\mathcal{S}$.

Our algorithm rewrites trivially into a probabilistic state machine. The vertices are the hidden states $q_0, \ldots, q_{|S|}$. The edges (q_i, q_j) are labeled with the transition probabilities $p_{i,j}$ and the input terms of the form (r, m). As in [13], we converted this edge-annotated state machine into a semantically equivalent state-annotated one. For each edge (q_i, q_j) labeled with $p_{i,j} : (r, m)$, a new state $q_{j,(r,m)}$ is created so that the output are observed on the states instead of the edges.

The elements of $\mathcal{O}$ are easily deduced from the (low level) description of the algorithm. For each internal state $(r, m) \in S$, the observed operations are derived from the computation $Q := h([m/h]Q + [r/h]P)$ where $h = \gcd(r, m)$. We assume that $[m/h]Q$ is evaluated using a left-to-right binary addition chain and that $[r/h]P$ is precomputed. For example, for the internal state $(1, 2)$, which corresponds to the congruence class $1 \pmod 2$, the attacker should observe the trace leaked from the computation $Q := 2Q + P$, i.e. a doubling followed by an addition. Hence $\mathcal{O} \supset \{DA\}$. For the covering set `u2c-24-10` given in (6), the resulting set of observable is given by:

$$\mathcal{O} = \{DA, DDA, DAD, DAAD, DDD, DDDA, DADD\}.$$

Let $\mu : S \mapsto \mathcal{O}$ the mapping from the set of internal states of the HMM to the set $\mathcal{O}$, such that $\mu(S) = \mathcal{O}$. Unlike the attacks from [13] and [39], a first obstruction for an attacker is that μ is not injective in our case. Indeed, for `u2c-24-10`, we have $\mu((2, 4)) = \mu((0, 6)) = DAD$; $\mu((-2, 6)) = \mu((2, 6)) = DAAD$; $\mu((-3, 8)) = \mu((1, 8)) = DDDA$.

The inference problem may be solved rather efficiently using the Viterbi algorithm. For a given trace $(y_0, \ldots, y_{s-1})$, this algorithm outputs the most likely execution sequence $(q_0, \ldots, q_{s-1})$ together with the likelihood of the result, i.e. the probability of that particular sequence among all possible execution sequences matching the observed trace $(y_0, \ldots, y_{s-1})$.

We implemented this version of an HMM attack on various covering systems of congruences. In all cases, we were unable to recover the execution sequences correctly. The Viterbi algorithm was able to guess correctly the internal states in one-to-one correspondence with the elements of $\mathcal{O}$ but was unable to recover with better probability than a random draw those states for which μ cannot be inverted uniquely. As a consequence, we observed very small likelihood scores.

At this point, it is important to notice that the attack failed even though the setting was extremely favorable to the attacker. Indeed, not only we considered that the trace was acquired without any error, but also that the attacker was able to break it up properly. However, contrary to the sets of observables from [13] and [39], a second major stumbling block is that a trace of the side-channel cannot always be written uniquely as $(y_0, \ldots, y_{s-1})$ with $y_i \in \mathcal{O}$. For example, the sequence $DADDDA$ can be observed from different execution sequences. It could come from $DA|DDDA$ obtained by $(1, 2)$ followed by either $(-3, 8)$ or $(1, 8)$; from $DADD|DA$ corresponding uniquely to the sequence of internal states $((4, 8), (1, 2))$; or from $DAD|DDA$ complying with either $(2, 4)$ or $(0, 6)$ followed by $(-1, 4)$.

In order to better reflect the conditions of a more realistic attack, we modified the previously constructed state-annotated probabilistic state machine so that the observable output for each internal state was either a doubling or an addition. Each state $q_{j,(r,m)}$ was split into exactly t states $q_{j_0,(r,m),s_0}, \ldots, q_{j_t,(r,m),s_t}$, where $s_0 s_1 \ldots s_t = \mu((r, m))$ and $s_i \in \{D, A\}$. We set to 1 the transition probabilities between those newly created states and adjusted the input and output ones with the initial probabilities. This transformation allowed us to run the Viterbi algorithm on any observed sequence, without any knowledge on its decomposition into words from $\mathcal{O}$. Here again, we set the attacker in ideal conditions by assuming that the observed traces contain no errors.

As an example of an HMM execution, we give the outputs of our Sage implementation of the Viterbi algorithm for the covering set `u3c-48-24` and a random 256-bit scalar in Fig. 6.

In order to measure the difference between the real sequence and the most likely sequence returned by the Viterbi algorithm, we considered the Levensthein distance (aka edit distance). Informally, the Levenshtein distance between two words is the minimum number of single-character edits (i.e.

```
sage: viterbi (256, '../csc/u3c-48-24')
Reading CSC... [Done]
Computing HMM matrices... [Done]
Generating the discrete HMM... [Done]
Collecting an execution trace for k [Done]
k = 113280982734524645135658082899406751332655632772245315887318249972569794866207
['D', 'D', 'A', 'D', 'D', 'D', 'D', 'D', 'A', 'D', 'D', 'D', 'A', 'D', 'A', 'D', 'D', 'A', 'D', 'A', 'D', 'D', 'D', 'A',
'D', 'D', 'A', 'D', 'D', 'A', 'D', 'D', 'A', 'D', 'A', 'D', 'D', 'D', 'A', 'D', 'D', 'D', 'A', 'D', 'D', 'D', 'D', 'D',
'D', 'A', 'D', 'D', 'D', 'D', 'D', 'A', 'D', 'A', 'D', 'A', 'D', 'A', 'D', 'A', 'D', 'D', 'A', 'D', 'D', 'A', 'D', 'D',
'A', 'D', 'D', 'D', 'D', 'D', 'A', 'D', 'D', 'D', 'A', 'D', 'D', 'A', 'D', 'A', 'D', 'A', 'D', 'D', 'D', 'D', 'D', 'A',
'D', 'A', 'D', 'A', 'D', 'D', 'A', 'D', 'A', 'D', 'D', 'D', 'A', 'D', 'D', 'A', 'D', 'D', 'A', 'D', 'D', 'D', 'A', 'D', 'A',
'D', 'D', 'D', 'D', 'A', 'D', 'D', 'A', 'D', 'D', 'D', 'A', 'D', 'D', 'A', 'D', 'A', 'D', 'A', 'D', 'D', 'D', 'D', 'A',
'D', 'D', 'D', 'A', 'D', 'D', 'A', 'D', 'D', 'D', 'D', 'D', 'D', 'D', 'D', 'A', 'D', 'A', 'D', 'D', 'D', 'D', 'A', 'D',
'A', 'D', 'A', 'D', 'D', 'A', 'D', 'A', 'D', 'D', 'A', 'D', 'D', 'D', 'A', 'D', 'A', 'D', 'A', 'D', 'D', 'D', 'D', 'A',
'D', 'D', 'D', 'A', 'D', 'A', 'D', 'A', 'D', 'A', 'D', 'A', 'D', 'D', 'D', 'A', 'D', 'A', 'D', 'A', 'D', 'D', 'D', 'D', 'A', 'D',
'D', 'D', 'A', 'D', 'D', 'D', 'D', 'A', 'D', 'A', 'D', 'A', 'D', 'D', 'A', 'D', 'A', 'D', 'A', 'D', 'D', 'D', 'D', 'D',
'D', 'D', 'A', 'D', 'D', 'D', 'D', 'D', 'D', 'D', 'A', 'D', 'A', 'D', 'D', 'A', 'D', 'D', 'A', 'D', 'D', 'A', 'D', 'A', 'D',
'D', 'A', 'D', 'D', 'D', 'A', 'D', 'A', 'D', 'A', 'D', 'A', 'D', 'D', 'D', 'D', 'A', 'D', 'D', 'A', 'D', 'D', 'A', 'D',
'A', 'D', 'D', 'D', 'D', 'A', 'D', 'D', 'D', 'A', 'D', 'A', 'D', 'A', 'D', 'D', 'D', 'D', 'D', 'D', 'D', 'A', 'D', 'D',
'A', 'D', 'D', 'D', 'A', 'D', 'A', 'D', 'D', 'A', 'D', 'A', 'D', 'A', 'D', 'A', 'D', 'D', 'A', 'D', 'D', 'A', 'D', 'A',
'D', 'D', 'D', 'D', 'D', 'A']
Running Viterbi on HMM... [Done]
Real sequence: [(-1, 4), (0, 4), (2, 16), (0, 2), (-2, 12), (2, 12), (-2, 8), (1, 12), (0, 2), (-1, 6), (-6, 16), (-1,
4), (0, 2), (0, 2), (3, 16), (0, 2), (0, 2), (1, 8), (3, 6), (2, 12), (5, 12), (-2, 8), (0, 2), (2, 16), (-1, 4), (-1,
4), (3, 6), (0, 2), (0, 2), (0, 2), (-1, 4), (3, 6), (-3, 12), (0, 2), (-3, 12), (-4, 16), (-2, 12), (1, 8), (-1, 4),
(-1, 8), (-1, 4), (-1, 6), (0, 2), (2, 16), (0, 2), (5, 12), (0, 2), (0, 8), (0, 4), (0, 2), (2, 12), (-1, 8), (-2, 12),
(-1, 6), (-2, 8), (-3, 16), (0, 2), (0, 2), (0, 2), (6, 12), (0, 2), (-1, 6), (-1, 6), (-1, 8), (3, 6), (-3, 16), (-6,
16), (0, 2), (-3, 12), (5, 12), (6, 12), (0, 8), (-6, 16), (0, 4), (-1, 8), (1, 6), (-4, 16), (-1, 6), (4, 16), (-1, 6),
(-2, 12), (0, 4), (1, 12), (0, 2), (3, 6), (0, 2), (1, 8), (1, 8), (-2, 12), (0, 4), (5, 16), (-2, 8), (0, 2), (1, 6),
(0, 2), (1, 6), (3, 6), (-2, 8), (-2, 12), (3, 16)]
length: 100
Most likely sequence: [(4, 16), (2, 16), (-3, 12), (-3, 12), (2, 16), (1, 12), (-3, 12), (1, 8), (2, 16), (0, 4), (2,
16), (3, 16), (1, 6), (2, 12), (1, 12), (-2, 8), (3, 16), (1, 8), (-1, 4), (2, 12), (3, 16), (2, 12), (2, 12), (-3, 12),
(4, 16), (2, 12), (1, 8), (4, 16), (1, 12), (1, 6), (3, 16), (1, 8), (4, 16), (0, 8), (0, 4), (2, 12), (1, 8), (1, 6),
(-3, 12), (4, 16), (2, 16), (0, 4), (2, 12), (-3, 12), (2, 12), (-1, 4), (2, 12), (2, 16), (4, 16), (-3, 12), (1, 12),
(2, 12), (0, 8), (2, 16), (0, 8), (-1, 4), (1, 6), (-2, 8), (-3, 12), (4, 16), (1, 6), (1, 6), (3, 16), (-2, 8), (2, 12),
(2, 16), (-1, 4), (2, 12), (0, 8), (1, 8), (-2, 8), (-3, 12), (-3, 12), (1, 6), (-2, 8), (2, 12), (3, 16)]
length: 77
Log likelyhood: -266.526042707
Levensthein dist: 81
```

Figure 6. Execution of the Viterbi algorithm in the context of an HMM attack on u3c−48−24 for a random 256-bit scalar.

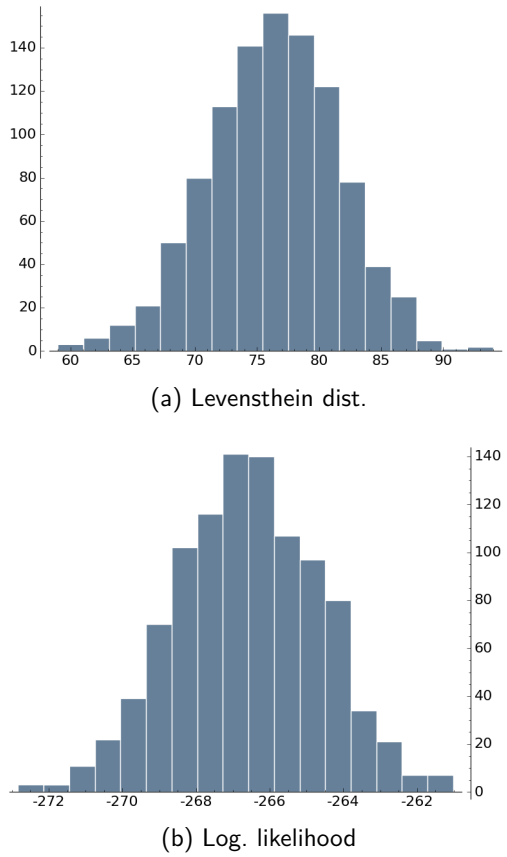
insertions, deletions or substitutions) required to change one word into the other. In the example above, the Levensthein distance between the two sequences is 81. It is thus impossible to recover the real sequence from the one returned by the Viterbi algorithm. The Log likelihood value expresses the logarithm of the probability of that particular returned sequence among all sequences compatible with the observed trace. In the example, it tells us that the probability of recovering k from the returned sequence is smaller than that of guessing k by selecting each bit at random!

The results observed for the above example are not isolated. In Fig. 7, we give the distribution of the Levensthein distance and the Log likelihood values obtained over 1000 HMM simulations on `u3c-48-24` for 256-bit random scalars.

3.3 Timing attacks

Timing attacks require multiple executions of the algorithm. They exploit dependencies between the execution time of the algorithm and some secret data processed through its running. These attacks may be prevented by ensuring that the computation time is independent from this secret value; a property commonly guaranteed using constant time algorithms. Clearly, in the present form, our algorithm does *not* run in constant time as the time for computing $[k]P$ may vary depending on the randomly selected mixed radix representation of k. Therefore, the fundamental question is: how much information can be deduced from the varying running time of our algorithm?

To the best of our knowledge, the only publicly known timing attack on ECC is due to Brumley and Tuveri [41]. In 2011, they presented a successful timing attack on an OpenSSL implementation of the signature phase of ECDSA, in particular a scalar multiplication $[k]P$, where nonce k is selected uniformly at random. Their attack exploits the dependency between the computation time and the bitlength of k. It is effective because of the loop optimization strategy implemented (at the time) in OpenSSL. The attack operates in two phases: first, using the time dependency, a certain amount of signatures coming from "short" scalars are collected. Then, a lattice attack using the set of signatures filtered in the collection phase is mounted to recover the secret key used to generate the ECDSA signatures. As pointed out by the authors, the attack is successful when the first phase is effective, i.e. when the filtered signatures actually correspond to scalars shorter than some fixed threshold with very high probability. Logically, the attack success rate decreases dramatically with the increase of



(a) Levensthein dist.

(b) Log. likelihood

Figure 7. Distribution of Levensthein distances (top) and log likelihood (bottom) over 1000 HMM attacks on u3c-48-24 for 256-bit scalars.

false positives.

The countermeasure proposed by Brumley and Tuveri consists in replacing k by an equivalent value $\hat{k}$ of fixed bitlength. This can be achieved by adding independently $\text{ord}(P)$ and $2\text{ord}(P)$ to k and choosing for $\hat{k}$ that result of bitlength $1 + \text{size}(\text{ord}(P))$. Obviously, the same countermeasure does apply to any scalar multiplication algorithm. Thus, the above question becomes: how much information can be deduced from the running time of our algorithm for scalars of the same size?

As seen in Section 3.1, for any given scalar k, the execution trace of the algorithm corresponds to a path of maximal length from k to 0 in a direct acyclic graph. For any given covering system, the DAG which represents all possible executions of the algorithm for all possible input values of fixed bitlength is a multi-source DAG with single-sink 0. (In Fig 5, we presented a sub-graph of that DAG whose single source is the node 10273.)

In order to simulate timing measurements, we considered weighted DAGs, where the weight $w_{(r_i,m_i)}$ given to the edge (r_i, m_i) corresponds to the cost for computing $[m_i]Q + [r_i]P$. This cost $w_{(r_i,m_i)}$ could be anything meaningful, e.g. a number of curve operations or a number of field operations or even a number of clock cycles. For our experiments, we considered the number of field multiplications. Observe that for a given covering set there are only finitely many different weights that may appear in the associated DAG. For example, with `u3c-48-24`, and assuming that the points $[r_i]P$ are precomputed, there are only 13 different sequences of curve operations. These sequences are listed in Table 4 together with the corresponding number of field multiplications. (We refer the reader back to Section 3.2 for details on how the sequences of curve operations are obtained. For the "#mult" columns, we used the best operation counts for short Weierstrass curves from Bernstein and Lange's compilation of Explicit-Formulas [32] assuming $a_4 = -3$ and $S = 0.8M$; that is $A = 10.2, D = 7$.)

Let $W_{\mathcal{S}} = \{w_{s_1}, \ldots, w_{s_t}\}$ denote the set of possible weights for the covering set $\mathcal{S} = \{s_1, \ldots, s_t\}$. Clearly, any possible running time $w_{k\to 0}$ for computing $[k]P$ is equal to the sum of the weights along a randomly chosen path from k to 0. It can also be expressed as $w_{k\to 0} = \sum_{i=1}^{|W_{\mathcal{S}}|} \alpha_i w_i$, where each α_i corresponds to the number of times an edge of weight w_i was encountered. For a given k, there are many possible values for $w_{k\to 0}$. Counting the exact number of possible values for $w_{k\to 0}$ is a difficult combinatorial problem. And counting the cardinal of the set of all possible $w_{k\to 0}$ for all possible sources k of fixed bitlength is even harder.

Therefore, in order to asses the applicability of a potential timing attack on our randomized algorithm, we simulated the execution of our algorithm on 100000 different random 256-bit scalars and bucket-sorted these scalars based on the (theoretical) running times (using the number of field multiplications from Table 4). The results given in the next paragraphs have been obtained with the exact 3-cover `u3c-48-24`. Our simulations resulted in 469 buckets, ranging from 2487 to 2915.6 multiplications.

3.3.1 *Long sequences of 0 and 1 and common patterns:*

Brumley and Tuveri's attack [41] is successful when the filtered signatures correspond to short scalars, i.e. scalars whose leading bits are all zeros. With the proposed countermeasure, all nonce k used in the first phase of ECDSA have the same bitlength, namely $k \in [2^{j-1}, 2^j - 1]$. However, we believe that the lattice attack exploited by Brumley and Tuveri may still be effective if the filtered signatures correspond to scalars with identified patterns, e.g. long sequences of zeros (resp. ones). Hence, we checked whether any such pattern was leaking from the timing variations of our algorithm. To do so, we drawn t scalars, one-by-one, from the smaller buckets, i.e. those containing the scalars that had led to the smallest costs, and bitwise ORed them together until reaching $2^j - 1$. Indeed, if two scalars share a common zero-pattern such as a long sequence of zeros, then this pattern/sequence will still appear after a bitwise OR operation.

In our experiment, the value $2^{256} - 1$ was reached after only 9 scalars; the last one picked in the bucket of cost 2497.8. We performed the same computations with scalars picked in the larger buckets. The value $2^{256} - 1$ was reached after only 8 scalars; the last one picked in the bucket of cost 2881.8. For completeness, we ran several similar computations with scalars picked randomly. On average, the value $2^{256} - 1$ was reached with only 9 scalars.

Similarly, we checked the presence of common one-patterns (e.g. long sequences of ones) using a bitwise AND strategy. The minimal value (2^{255}) was reached after 9 scalars for the smallest buckets, 8 scalars for the largest ones and for 8 scalars on average for scalars picked at random.

Therefore, we can safely conclude that the time variations of our algorithm do not provide any information on potential

Table 4
Sequences of curve operations and associated number of field multiplications (on short Weierstrass curves with $a_4 = -3$) for each congruence class of the exact 3-cover u3c-48-24. Observe that the "#mult" column only contains 8 different values

$r \pmod m$	Curve ops.	#mult
$0 \pmod 2$	D	7.0
$-1 \pmod 4$	DDA	24.2
$0 \pmod 4$	DD	14.0
$-1 \pmod 6$	$DADA$	34.4
$1 \pmod 6$	$DADA$	34.4
$3 \pmod 6$	$DADA$	34.4
$-2 \pmod 8$	$DDAD$	31.2
$-1 \pmod 8$	$DDDA$	31.2

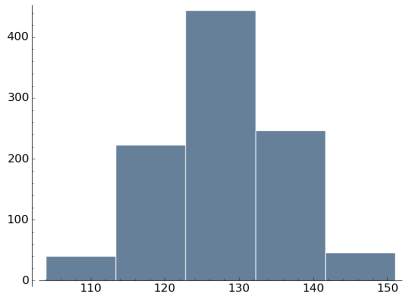
$r \pmod m$	Curve ops.	#mult
$0 \pmod 8$	DDD	21.0
$1 \pmod 8$	$DDDA$	31.2
$-3 \pmod{12}$	$DDADA$	41.4
$-2 \pmod{12}$	$DADAD$	41.4
$1 \pmod{12}$	$DADDA$	41.4
$2 \pmod{12}$	$DADAD$	41.4
$5 \pmod{12}$	$DADDA$	41.4
$6 \pmod{12}$	$DADAD$	41.4

$r \pmod m$	Curve ops.	#mult
$-6 \pmod{16}$	$DDDAD$	38.2
$-5 \pmod{16}$	$DDDDA$	38.2
$-4 \pmod{16}$	$DDADD$	38.2
$-3 \pmod{16}$	$DDDDA$	38.2
$2 \pmod{16}$	$DDDAD$	38.2
$3 \pmod{16}$	$DDDDA$	38.2
$4 \pmod{16}$	$DDADD$	38.2
$5 \pmod{16}$	$DDDDA$	38.2

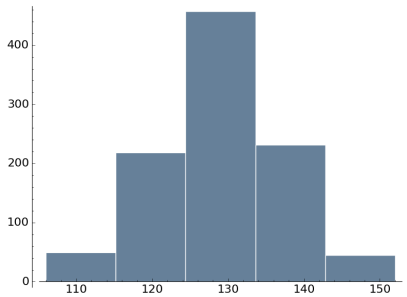
patterns of identical bits. Therefore, the filtering phase does not allow to reduce the size of the lattice used in the second phase of the attack. Hence, we claim that the timing attack proposed in [41] does not apply.

3.3.2 *Hamming weight:*

We also checked whether the Hamming weight of k was leaking from the time variations of our algorithm. To do so, we collected the Hamming weights of $t = 1000$ scalars picked from the smaller buckets and we compared them to the Hamming weights of 1000 scalars picked at random. In Fig. 8, we show the distributions of these Hamming weights in both cases.



(a) "faster" scalars



(b) random scalars

Figure 8. Distribution of Hamming weights of 1000 scalars from the smaller buckets (top) and randomly among all buckets (bottom)

Our conclusion is that, unlike the double-and-add or w-NAF algorithms, the time variations do not reveal any information on the Hamming weight of the scalar k. Neither the span, nor the distribution seem to provide any useful information to an attacker.

3.4 Horizontal and simple attacks

In the previous sections, we proved that advanced attacks which require several execution of the algorithm are defeated by our randomization strategy. Let us now focus on horizontal and simple attacks.

In order to protect an algorithm against SPA-type attacks, one needs to guarantee that the observation of a single trace does not provide any hint to an attacker. For example, double-and-add algorithms are vulnerable to SPA when the execution trace allows to distinguish point doublings from general additions.

Several implementation options have been proposed to thwart simple attacks. Until recently, the atomicity principle [21], [23], [22], or the use of complete/unified group laws were both considered efficient and robust against SPA since an attacker could not distinguish a point doubling from a point addition. In [6], Bauer et al. presented a novel horizontal attack which defeats all these celebrated countermeasures. Their attack exploits the fact that point additions and point doublings may still be identified if the adversary can detect when two field multiplications have at least one operand in common (Assumption 1 in [6]). They show that their attack indeed applies to different atomic implementations and to the unified formula on Edward's curves from [42]. They evaluate the soundness of their attack and provide some experimental results showing convincing success rates. As in [43], their distinguisher targets the operands of long integer multiplications.

The overall philosophy of their attack is based on the fact that, if the adversary can guess the entire sequence of operations (C_i) without any error, then she can immediately read all the bits of the secret scalar k from that sequence since the order of those operations in the sequence is a one-to-one function of k.

With our randomized algorithm, even if the attacker was in this very favorable situation, Bauer et al. horizontal attack does not apply because the order of the operations C_i is *not* in one-to-one correspondence with the secret scalar k. Indeed, as stated in Section 2, a CSC-based algorithm reduces to a sequence of operations of the form $[m_i]Q + [r_i]P$, where the set of possible values (r_i, m_i) depends on the covering system. For example, with the exact 3-cover used to produce Fig. 5, a possible sequence for $k = 10273$ is:

$$(1 \pmod{12}, 0 \pmod 4, 10 \pmod{12}, 5 \pmod{12}, 1 \pmod{12}),$$

which corresponds to the following path in the above mentioned DAG:

$$10273 \to 856 \to 214 \to 17 \to 1 \to 0.$$

The recursion can easily be rewritten as: $10273 = 1 + 12(0 + 4(10 + 12(5 + 12(1 + 12.0))))$ so that:

$$[10273]P = P + [12]([4]([10]P + [12]([5]P + [12](P + [12]P_\infty))))$$

If one assumes that the points $[r_i]P$ are precomputed and if one uses a left-to-right double-and-add algorithm to evaluate the terms $[m_i]Q$, the execution trace $Tr(k)$ looks like:

$$Tr(10273) = \texttt{DADDADADDADADDADDDADDA}. \quad (7)$$

Clearly, the mapping Tr from $\mathrm{MRS}(\mathbb{Z})$ to `(D|A)*` is not injective. For example, with the same exact 3-cover, the above pattern could be attained starting from

$$43455 = 3 + 4(7 + 8(1 + 12(5 + 12(9 + 12.0)))),$$

or

$$14649 = 9 + 12(0 + 4(5 + 12(1 + 12(2 + 12.0)))),$$

and many other scalars. (Checking that the above expressions map to the trace given in (7) is immediate.)

In general, a given trace can be produced by very many different scalars. Counting the exact number of those scalars seems to be a difficult, yet interesting combinatorial question that we will not tackle in this work. However, in order to assess the robustness of our algorithm, we tried to determine that number of different scalars experimentally. We processed as follow: for a given length (i.e. number of symbols), we first computed all the possible traces of that length. Then, for each of these traces we computed its preimage, i.e. the subset of $\mathbb{Z}$ of all integers that maps to that trace. Finally, we computed the average size of these preimage subsets over all traces of that given length. Obviously, this brute-force strategy did not allow us to reach cryptographic sizes. However, as the trace size grows, we observed (using `u3c-48-24`) that the sequence of these average values, is in geometric progression with common ratio $\simeq 1.28$. It was therefore possible to derive an estimate for that number of possible integers that correspond to a given trace of length 335, which corresponds to the average length of a trace produced by a 256-bit integer. It is greater than $10^{35} > 2^{116}$, which is quite an encouraging result. We acknowledge that this is a preliminary analysis, but given this experimental results we are hopeful that our scalar multiplication protects the circuit against horizontal attacks and simple attacks.

4 Covering systems generation

In order to run our numerical experiments, we had to generate exact n-covers. For that purpose we chose to generate them randomly. Given a set of predefined moduli $\{m_1, \ldots, m_t\}$ and a covering degree n, the problem consists of assigning integer values to $r_1, \ldots, r_t$ such that all the following conditions are fulfilled:

- $r_i \in \{0, \ldots, m_i - 1\}$ for all $i \in \{1, \ldots, t\}$;
- $m_i = m_j \Rightarrow r_i \neq r_j$ for all $i, j \in \{1, \ldots, t\}$;
- for all $k \in \{0, \ldots, \ell\}$, $|\mathcal{S}_{(k)}| = n$.

We used a very elementary greedy approach. Starting with the smallest moduli, we selected values r_i at random until a solution is found. When the value assigned to a residue produces an integer in $\{0, \ldots, \ell\}$ that is covered by more than n congruence classes, we backtrack by selecting another value for the most recently assigned residue. To speed up the process, we use a restart heuristic after a small number of backtrack steps. The resulting covering systems such as those listed in Table 3 are denoted `cs`n-ℓ-t, where n stands for the covering degree, $\ell = \mathrm{lcm}(m_1, \ldots, m_t)$ and $t = |\mathcal{S}|$.

In order to simplify the generation of exact n-covers, we may require that each modulo should cover the same proportion of integers. We called the resulting covering systems "uniform" and named them `u`n`c-`ℓ-t. In our experiments, all the uniform covering systems contain exactly $2n$ moduli, all of which are even, so that each modulo covers exactly half of the integers as illustrated in Fig. 9. This way, we

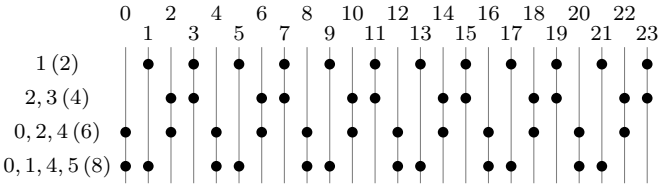


Figure 9. $\{1 \pmod 2;\ 2, 3 \pmod 4;\ 0, 2, 4 \pmod 6;\ 0, 1, 4, 5 \pmod 8\}$ is a uniform exact 2-cover. Each modulo covers exactly 12 integers out of 24.

also ensure that our covering systems do not contain the whole set of congruence classes for a given modulo such as $0 \pmod 2$; $1 \pmod 2$. We acknowledge that this generation strategy is pretty basic. It proved sufficient to generate exact n-covers of reasonably large covering degree[3] but can certainly be improved using more sophisticated tools, for example using efficient CSP[4] heuristics.

Determining the main characteristics of a "good" CSC is clearly an important direction for future work. At this stage, our observations are very preliminary. For efficiency, a good CSC would take advantage of fast tripling operations. But on the other hand, one has to ensure that it does not significantly simplify the identification of patterns in the collected traces. The list of moduli should not contain powers of 2 only as a clever horizontal splitting of the trace would reveal informations on the bits of k. The choice of which congruence classes cover 0 may also affect the robustness of a system regrading simple attacks. It seems that covering degrees as small as 2 or 3 could suffice. However, a basic recommendation would be to choose n as large as possible depending on the memory available (both for precomputations[5] and code size). There are certainly many more questions behind the choice of a good CSC. We leave these open for now.

5 Conclusions

In this paper, we proposed a new randomized scalar multiplication algorithm with built-in protection against various side-channel attacks. It is an efficient alternative to Coron's scalar randomization method, in particular for curves defined modulo primes that are close to powers of two. Compared to

3. Our largest covering system is an exact 12-cover comprised of more than 3000 congruence classes.

4. constraint satisfaction problem

5. See Section 2.3.

classical addition chains (D&A, wNAF, etc.) the extra cost implied by integer arithmetic remains negligible.

Going from this theoretical description to efficient and secure implementations, both in hardware and in software, will be the subject of future research. Many questions will have to be addressed, such as securing the inner randomization steps, designing an efficient yet robust overall control (HW) and instruction decoding (SW), protecting the memory addressing, etc.

Our algorithm is purposely designed to inhibit several side-channel attacks at once. Contrary to previous solutions aiming at randomizing the addition chain [12], [14], differential attacks are defeated by the proper randomization of our algorithm following the requirements given in [16], i.e. a very large number of possible internal states and a large number of possible transitions from each state. Despite some fitting adjustments to the previously published HMM attacks on finite state stochastic processes [13], [39], our simulation results also demonstrated the robustness of our algorithm against this eminently relevant class of attacks. Likewise, we proved that simple attacks and the recent horizontal collision correlation attack from Bauer et al. [6] remain worthless, even when the adversary were able to distinguish, without any error, a point doubling from a general addition. Finally, although our algorithm does *not* run in constant time, we presented some experimental results assessing the inapplicability of Brumley and Tuveri's timing attack on an OpenSSL implementation of ECDSA [41]. We showed that the timing variations implied by the randomization do not provide any useful information on the bits of k.

Nevertheless, past and recent advances in side-channel attacks imply a close attentiveness. The fact that none of the previously published attacks seem to operate does not mean that our randomized algorithm will remain unscathed forever. In fact, we hope that side-channel experts and cryptanalysts will consider the challenging questions behind our proposal. Sound side-channel attacks may exist but, at this point, are still to be discovered.

Acknowledgments

We would like to thank the anonymous reviewers for their careful reading and constructive comments. We also express our gratitude to Benoît De-Saporta and Alain Jean-Marie for answering our questions, as well as Cyril Bouvier, Peter Schwabe, Damien Vergnaud and the ECo group at LIRMM for their invaluable feedback and support.

References

[1] P. C. Kocher, "Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems," in *Advances in Cryptology, CRYPTO 1996*, ser. Lecture Notes in Computer Science, vol. 1109. Springer, 1996, pp. 104–113.

[2] D. Boneh, R. A. DeMillo, and R. J. Lipton, "On the importance of checking cryptographic protocols for faults," in *Advances in Cryptology, EUROCRYPT 1997*, ser. Lecture Notes in Computer Science, vol. 1233. Springer, 1997, pp. 37–51.

[3] P. C. Kocher, J. Jaffe, and B. Jun, "Differential power analysis," in *Advances in Cryptology, CRYPTO 1999*, ser. Lecture Notes in Computer Science, vol. 1666. Springer, 1999, pp. 388–397.

[4] S. Chari, J. R. Rao, and P. Rohatgi, "Template attacks," in *Cryptographic Hardware and Embedded Systems, CHES 2002*, ser. Lecture Notes in Computer Science, vol. 2523. Springer, 2002, pp. 13–28.

[5] C. Clavier, B. Feix, G. Gagnerot, C. Giraud, M. Roussellet, and V. Verneuil, "ROSETTA for single trace analysis," in *Progress in Cryptology - INDOCRYPT 2012, 13th International Conference on Cryptology in India, Proceedings*, 2012, pp. 140–155.

[6] A. Bauer, E. Jaulmes, E. Prouff, and J. Wild, "Horizontal collision correlation attack on elliptic curves," in *Selected Areas in Cryptography - SAC 2013 - 20th International Conference, Revised Selected Papers*, 2013, pp. 553–570.

[7] J.-S. Coron, "Resistance against differential power analysis for elliptic curve cryptography," in *Cryptographic Hardware and Embedded Systems, CHES 1999*, ser. Lecture Notes in Computer Science, Ç. K. Koç and C. Paar, Eds., vol. 1717. Springer, 1999, pp. 292–302.

[8] J. Fan, X. Guo, E. D. Mulder, P. Schaumont, B. Preneel, and I. Verbauwhede, "State-of-the-art of secure ECC implementations: a survey on known side-channel attacks and countermeasures," in *Proceedings of the 2010 IEEE International Symposium on Hardware-Oriented Security and Trust, HOST 2010*. IEEE, 2010, pp. 76–87.

[9] J. Fan and I. Verbauwhede, "An updated survey on secure ECC implementations: Attacks, countermeasures and cost," in *Cryptography and Security: From Theory to Applications*, ser. Lecture Notes in Computer Science. Springer, 2012, vol. 6805, pp. 265–282.

[10] C. D. Walter, "MIST: An efficient, randomized exponentiation algorithm for resisting power analysis," in *Topics in Cryptology – CT-RSA 2002*, ser. Lecture Notes in Computer Science, B. Preenel, Ed., vol. 2271. Springer, 2002, pp. 53–66.

[11] J.-C. Bajard, L. Imbert, P.-Y. Liardet, and Y. Teglia, "Leak resistant arithmetic," in *Cryptographic Hardware and Embedded Systems, CHES 2004*, ser. Lecture Notes in Computer Science, vol. 3156. Springer, 2004, pp. 62–75.

[12] E. Oswald and M. Aigner, "Randomized addition-subtraction chains as a countermeasure against power attacks," in *Cryprographic Hardware and Embedded Systems, CHES 2001*, ser. Lecture Notes in Computer Science, no. 2162. Springer, 2001, pp. 39–50.

[13] C. Karlof and D. Wagner, "Hidden markov model cryptanalysis," in *Cryptographic Hardware and Embedded Systems, CHES 2003*, ser. Lecture Notes in Computer Science, no. 2779. Springer, 2003, pp. 17–34.

[14] J. Ha and S.-J. Moon, "Randomized signed-scalar multiplication of ECC to resist power attacks," in *Cryptographic Hardware and Embedded Systems, CHES 2002, 4th International Workshop, Redwood Shores, CA, USA, August 13-15, 2002, Revised Papers*, vol. 2523, 2002, pp. 551–563.

[15] N. Ebeid and M. A. Hasan, "On binary signed digit representations of integers," *Designs, Codes and Cryptography*, vol. 42, no. 1, pp. 43–65, 2007.

[16] P.-A. Fouque, F. Muller, G. Poupard, and F. Valette, "Defeating countermeasures based on randomized BSD representations," in *Cryptographic hardware and Embedded Systems, CHES 2004*, ser. Lecture Notes in Computer Science, no. 3156. Springer, 2004, pp. 312–327.

[17] N. Méloni and M. A. Hasan, "Random digit representation of integers," in *Proceedings of the 23rd IEEE Symposium on Computer Arithmetic, ARITH23*. IEEE Computer Society, 2016, pp. 118–125.

[18] D. J. Bernstein, "Curve25519: New Diffie-Hellman speed records," in *Proceedings of Public Key Cryptography, PKC 2006*, ser. Lecture Notes in Computer Science, vol. 3958. Springer, 2006, pp. 207–228.

[19] D. J. Bernstein and T. Lange, "SafeCurves: choosing safe curves for elliptic-curve cryptography," http://safecurves.cr.yp.to.

[20] J. W. Bos, C. Costello, P. Longa, and M. Naehrig, "Selecting elliptic curves for cryptography: an efficiency and security analysis," *Journal of Cryptographic Engineering*, vol. 6, no. 4, pp. 259–286, 2015.

[21] B. Chevalier-Mames, M. Ciet, and M. Joye, "Low-cost solutions for preventing simple side-channel analysis: Side-channel atomicity," *IEEE Transactions on Computers*, vol. 53, no. 6, pp. 760–768, Jun. 2004.

[22] C. Giraud and V. Verneuil, "Atomicity improvements for elliptic curve scalar multiplication," in *Smart Card Research and Advanced Applications, CARDIS 2010*, ser. Lecture Notes in Computer Science, no. 6035. Springer, 2010, pp. 80–101.

[23] P. Longa, "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.

[24] D. J. Bernstein and T. Lange, "Analysis and optimization of elliptic-curve single-scalar multiplication," Cryptology ePrint Archive, Report 2007/455, 2007, http://eprint.iacr.org/.

[25] E. Brier and M. Joye, "Weierstras elliptic curves and side-channel attacks," in *Public Key Cryptography, PKC 2002*, ser. Lecture Notes in Computer Science, D. Naccache and P. Paillier, Eds., vol. 2274. Springer, 2002, pp. 335–345.

[26] E. Nascimento, L. Chmielewski, D. Oswald, and P. Schwabe, "Attacking embedded ECC implementations through cmov side channels," Cryptology ePrint Archive, Report 2016/923, 2016, http://eprint.iacr.org/2016/923.

[27] B. Hough, "Solution of the minimum modulus problem for covering systems," arXiv:1307.0874v2 [math.NT], 2014, http://arxiv.org/abs/1307.0874.

[28] J. A. Solinas, "Efficient arithmetic on Koblitz curves," *Designs, Codes and Cryptography*, vol. 19, no. 2–3, pp. 195–249, 2000.

[29] B. Möller, "Improved techniques for fast exponentiation," in *Information Security and Cryptology, ICISC 2002*, ser. Lecture Notes in Computer Science, vol. 2587. Springer, 2003, pp. 298–312.

[30] 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/.

[31] J. Adikari, V. Dimitrov, and L. Imbert, "Hybrid binary-ternary number system for elliptic curve cryptosystems," *IEEE Transactions on Computers*, vol. 60, no. 2, pp. 254–265, 2011.

[32] D. J. Bernstein and T. Lange, "Explicit-formulas database," URL: http://www.hyperelliptic.org/EFD/, joint work by Daniel J. Bernstein and Tanja Lange, building on work by many authors.

[33] M. Hutter, M. Joye, and Y. Sierra, "Memory-constrained implementations of elliptic curve cryptography in Co-Z coordinate representation," in *Progress in Cryptology, AFRICACRYPT 2011*, ser. Lecture Notes in Computer Science, vol. 6737. Springer, 2011, pp. 170–187.

[34] T. Chabrier and A. Tisserand, "On-the-fly multi-base recoding for ECC scalar multiplication without pre-computations," in *21st IEEE Symposium on Computer Arithmetic, ARITH21*. IEEE Computer Society, 2013, pp. 219–228.

[35] T. Jebelean, "An algorithm for exact division," *Journal of Symbolic Computation*, vol. 15, no. 2, pp. 169–180, 1993.

[36] A. D. Booth, "A signed binary multiplication technique," *Quarterly Journal of Mechanics and Applied Mathematics*, vol. 4, no. 2, pp. 236–240, 1951, reprinted in E. E. Swartzlander, *Computer Arithmetic*, Vol. 1, IEEE Computer Society Press Tutorial, Los Alamitos, CA, 1990.

[37] R. L. Stratonovich, "Conditional markov processes," *Theory of Probability and its Applications*, vol. 5, no. 2, pp. 156–178, 1960.

[38] L. E. Baum and T. Petrie, "Statistical inference for probabilistic functions of finite state markov chains," *The Annals of Mathematical Statistics*, vol. 37, no. 6, pp. 1554–1563, 1966.

[39] P. J. Green, R. Noad, and N. P. Smart, "Further hidden markov model cryptanalysis," in *Cryptographic Hardware and Embedded Systems, CHES 2005*, ser. Lecture Notes in Computer Science, no. 3659. Springer, 2005, pp. 61–74.

[40] A. Viterbi, "Error bounds for convolutional codes and an asymptotically optimum decoding algorithm," *IEEE Transactions on Information Theory*, vol. 13, no. 2, pp. 260–269, 1967.

[41] B. B. Brumley and N. Tuveri, "Remote timing attacks are still practical," in *Computer Security, ESORICS 2011*, ser. Lecture Notes in Computer Science, vol. 6879. Springer, 2011, pp. 355–371.

[42] D. J. Bernstein and T. Lange, "Faster addition and doubling on elliptic curves," in *Advances in cryptology, ASIACRYPT 2007*, ser. Lecture Notes in Computer Science, vol. 4833. Springer, 2007, pp. 29–50.

[43] C. D. Walter, "Sliding windows succumbs to Big Mac attack," in *Cryptographic Hardware and Embedded Systems, CHES 2001*, ser. Lecture Notes in Computer Science, vol. 2162. Springer, 2001, pp. 286–299.

Eleonora Guerrini Eleonora Guerrini received the M.S. degree in 2005 from the university of Pisa, Pisa, Italy, and the PhD degree in 2009 from the University of Trento, Trento, Italy. From 2009 to 2012 she has been a postdoc research associate in the Graph Theory team of the Laboratory of Computer Science of Grenoble and Bordeaux (France) and in the Algorithm and Cryptography team of the laboratory of Computer Science of Caen (France). She is currently an Assistant Professor at the University of Montpellier, working in the Laboratory of Computer science LIRMM of Montpellier, Montpellier, France. Her research interests are both in combinatorial and computer algebra aspect of coding theory.

Laurent Imbert received the PhD degree in Computer sciences from the University of Marseille in 2000, and the "habilitation" degree from the University of Montpellier in 2008. He is a senior researcher at the Centre National de la Recherche Scientifique (CNRS), France and a member of the Laboratoire d'Informatique, Robotique et Microélectronique de Montpellier (LIRMM). Since July 2013, he has also been an adjunct professor at the University of Calgary, Canada. His research interests concern the design and analysis of arithmetic algorithms, modular and finite field arithmetic, foundations of number systems, computational number theory, elliptic curves and applications in public key cryptography, side-channel attacks of cryptographic devices and countermeasures.

Theo Winterhalter received the M.S. degree in theoretical computer sciences in 2016 from the École Normale Supérieure Paris-Saclay. He is currently a PhD student at the Institut Mines-Télécom (IMT) of Nantes, France. His research focuses on Universes and Higher Inductive Types in Homotopy Type Theory.