Randomizing Scalar Multiplication Using Exact Covering Systems of Congruences

Eleonora Guerrini[1], Laurent Imbert[1], and Théo Winterhalter[2]

[1] LIRMM, CNRS, University of Montpellier, France
[2] École Normale Supérieure de Cachan, France

Abstract. In this paper we present a generic, uniformly randomized scalar multiplication algorithm based on covering systems of congruences, with built-in protections against various side-channel attacks. It has been tailored to resist a recent class of attacks called horizontal attacks. These very powerful attacks exploit some unsuspected weaknesses hidden in most, if not all, highly regular and constant time algorithms. We provide a thorough complexity analysis, several arguments to support its robustness and some encouraging numerical experiments.

Keywords: Scalar Multiplication, Covering Systems of Congruences, Side-Channel Attacks, Randomized Algorithms

1 Introduction

Exponentiation is a crucial operation for most public key cryptosystems. It is used extensively in the generation/verification of electronic signatures (e.g. using DSA [26]) and in the encryption/decryption phases of RSA [24] or DL-based algorithms [13]. In many cases, data manipulated during these computations should 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.

Over the past fifteen years, side-channel attacks (SCA), first proposed by Kocher in 96 [19], have kept improving. They have diversified (simple attacks [19], differential attacks [18], collision attacks [10,3], template attacks [7], etc.), have exploited many sources of leakages (computation time, power consumption, electromagnetic emanations, sound, etc.) and have used increasingly sophisticated statistical tools, resulting in a vast side-channel arsenal. In parallel, a lot of researches have been conducted towards 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. Inevitably, each of these protection layers implies some overhead, e.g. computation time, circuit area, etc. Besides, it is essential to ensure that this combination of independent, yet good countermeasures does not weaken the overall implementation. Hence, solutions which impedes several SCA at once, at a reasonable cost, should be considered with great interest. The ultimate, all-in-one, protection is yet to be discovered!

In this paper, we present a randomized scalar multiplication algorithm over a generic group $\mathcal{G}$ using an additive notation. As an illustration, we considered the now widely used groups of point of an elliptic curve defined over a finite field. However, we do not use any property of those

particular underlying groups so that everything may be immediately transposed to any other finite (cyclic) group, in particular to the multiplicative group of a finite field. In this context, our goal is to strengthen the scalar multiplication, i.e. the operation which consists of computing an integer multiple of a group element P.

After recalling the major attacks on scalar multiplication and the principles behind the most prominent countermeasures in Section 2, we propose a novel generic randomized scalar multiplication algorithm in Section 3. In the same section we analyze its complexity on average and we provide some theoretical results regarding the cost of our approach. In section 4, we carefully analyze its robustness against several types of side-channel attacks. We provide clear claims about its level of protection. For some attacks, we suggest challenges that could lead to possible attacks. In Section 5, we present some first, yet encouraging experimental results for real-world elliptic curves. We discuss some further work in Sections 6 and 7.

2 Side-Channel Attacks on Scalar Multiplication

Differential side-channel attacks (DSCA) [18] are certainly among the most powerful side-channel attacks. They aim at partitioning a set of traces into subsets using an appropriate selection function in the hope that the difference between the average of these subsets will reveal the secret value manipulated by the algorithm. This occurs when the partitioning is correlated to the trace measurements. In the context of elliptic curve cryptography, several powerful countermeasures have been proposed by Coron [11]. These techniques comprise the randomization of the secret scalar, point blinding techniques and the use of randomized projective coordinates.

An alternative to Coron's countermeasures is to randomize the scalar multiplication algorithm itself. This is usually done by using a redundant representation of the scalar and by taking random decisions in the course of the algorithm. An approach of that type based on Binary Signed Digit (BSD) recodings [15,12], initially regarded as very promising, was eventually broken in [14]. Another example of a randomized algorithm is the Leak Resistant Arithmetic (LRA) concept proposed in [2] in the RSA context.

In general, countermeasures designed to resist differential attacks do not protect against timing attacks and simple attacks (SSCA). Thus, implementations have to be strengthened with additional protections. In his founder paper [19], Kocher stated that "The most obvious way to prevent timing attacks is to make all operations take exactly the same amount of time". Highly regular algorithms, such as the Montgomery ladder [17,23] depicted in Algorithm 1, are widely adopted examples of Kocher's strategy. Unlike other classical binary algorithms, they can run in constant time. This constant-time property is usually achieved by repeating the exact same sequence of operations through a big loop of constant length. Hence, even when the group law is not unified/complete – i.e. different formulas are used for a doubling and a general addition – the trace of the resulting sequence of operations has constant length and a very regular pattern.

Over the years, this regularity principle, under which the trace of a scalar multiplication should only reveal a repeating sequence of identical patterns, has been set in stone as the unique valid protection against timings and other simple attacks. Yet, some very recent attacks on RSA implementations tend to prove that this high regularity may in turn introduce some weaknesses. In [22], a new type of horizontal attack successfully defeated an RSA implementation based on the LRA. The authors managed to exploit correlations between well chosen portions of a unique trace of electromagnetic emanations and to recover the whole secret exponent.

Algorithm 1 Montgomery ladder

Input: $P \in \mathcal{G}$, $k = (k_{n-1}, \ldots, k_0)_2 \in \mathbb{N}$ with $k_{n-1} \neq 0$
Output: $Q = [k]P \in \mathcal{G}$
1: $R_0 := P$; $R_1 := [2]P$
2: **for** $i = n - 2$ **down to** 0 **do**
3: $\quad b := k_i$
4: $\quad R_{1-b} := R_{1-b} + R_b$
5: $\quad R_b := [2]R_b$
6: **return** R_0

The genesis of the present work was thus to design a robust, yet efficient scalar multiplication algorithm with built-in protections against DSCA, SSCA, timing, collision and these recently proposed horizontal attacks.

3 A New Randomized Scalar Multiplication

In this section, we present our novel randomized scalar multiplication algorithm. It is based on so-called covering systems of congruences.

3.1 Covering Systems of Congruences

A *covering system of congruence (CSC)* is a finite set of congruences $r_i \bmod m_i$, such that every integer satisfies at least one of them. 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

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

A covering system is called an *n-cover* if every integer is covered at least n times, and an *exact n-cover* if it covers each integer *exactly* n times. For example, the 10 congruence classes

$$1 \pmod 2;\ 2, 3 \pmod 4;\ 0, 2, 4 \pmod 6;\ 0, 1, 4, 5 \pmod 8$$

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

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 [16]. The non-existence of a distinct covering system whose moduli are all odd is still an open problem.

The link between covering systems and scalar multiplication algorithms is immediate. Let $P \in \mathcal{G}$ and $k \in \mathbb{Z}$. Then it is clear that

$$k \equiv r \pmod m \Rightarrow [k]P = [r]P + [m]([h]P),\ \text{where } h = (k - r)/m. \tag{1}$$

For example, the right-to-left double-and-add algorithm follows directly from the exact 1-cover given by the two congruence classes 0 (mod 2); 1 (mod 2). It instantly leads to

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

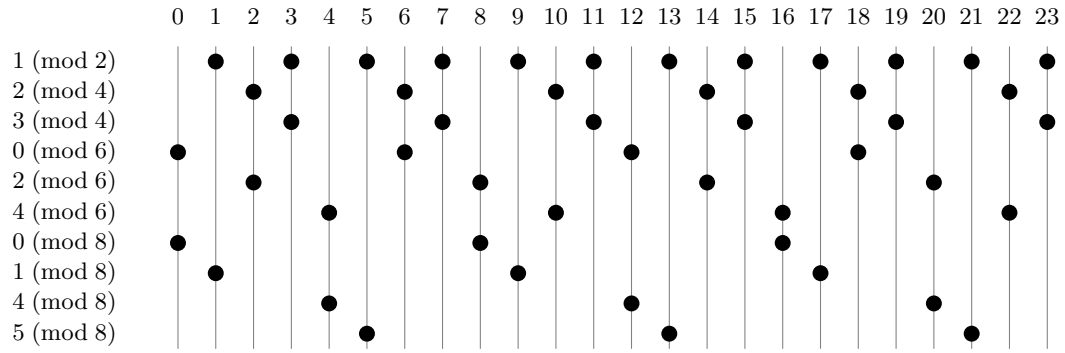


Fig. 1. $1 \pmod 2;\ 2,3 \pmod 4;\ 0,2,4 \pmod 6;\ 0,1,4,5 \pmod 8$ is an exact 2-cover. Each row corresponds to a congruence class; the dots indicate which integers are covered by each class. The number of dots in each column indicates the number of congruence classes covering the corresponding integer.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

In the world of (hyper)elliptic curves, many scalar multiplication algorithms have been proposed (m-ary, NAF, window methods, multi-base, etc.). It turns out that many of them can be expressed using covering system of congruences. We list a few of them and the corresponding covering system 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 [25]	$0\,(2);\ 1,-1\,(2^w);\ 3,-3\,(2^w);\ \ldots;\ 2^{w-1}-1,-2^{w-1}+1\,(2^w)$
frac. window [21]	$0\,(2);\ 1,-1\,(2^{w+1});\ 3,-3\,(2^{w+1});\ \ldots;\ 2^w+m;\ -2^w-m\,(2^{w+1})$
wmbNAF [20]	$0\,(a_1);\ 0\,(a_2);\ \ldots, 0\,(a_k);\ 1,-1\,(a_1^w);\ 3,-3\,(a_1^w);\ \ldots, (a_1-1)/2\,(a_1^w)$
w-HBTF [1]	$0\,(2);\ 0\,(3);\ 1,-1\,(w);\ \ldots;\ w/2-1,-w/2+1\,(w)$

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 congruence classes covering k, i.e. the set of all congruences $r \pmod m \in \mathcal{S}$ such that $k \equiv r \pmod m$. A generic CSC-based scalar multiplication is given in Algorithm 2.

The validity of Algorithm 2 is based on the fact that, for all covering system $\mathcal{S}$, we have

$$[m]Q + [r]P = [m(k-r)/m]P + [r]P = [k]P.$$

This 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 more than one congruence class, several algorithms can be deduced depending on which class is chosen.

Algorithm 2 Generic CSC-based scalar multiplication

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

The generic algorithm presented in Alg. 2 may be specialized in various ways. We sketch below the main characteristics of our randomized version designed to resist various types of side-channel attacks.

- First, we require that the covering system $\mathcal{S}$ is an exact n-cover (with $n \geq 2$). As shown in Section 4, this point is crucial. By doing so, we ensure that there are exactly n possible congruence classes $r \pmod m$ in $\mathcal{S}_{(k)}$ for every integer k.
- In line 3 of Alg. 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 4 we show that this leads to exponentially many ways to compute Q in line 4.

A concrete description is given in Algorithm 3. The exact covering system $\mathcal{S}$ is represented using a convenient data structure very close to the graphical representation shown in Figure 1. More precisely, if $\mathcal{S} = \{r_1 \pmod{m_1}, \ldots, r_t \pmod{m_t}\}$ is an exact n-cover, it is stored in memory 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 itself a one-dimensional array, of length exactly n, whose elements are the congruence classes $r \pmod m$ covering all the integers of the form $i + \ell\mathbb{Z}$. In Table 2, we give an example for the covering system `u2c` (see Section 3.2 for the explanation of its name) defined by the 10 congruence classes: $1 \pmod 2$; $2, 3 \pmod 4$; $0, 2, 4 \pmod 6$; $0, 1, 4, 5 \pmod 8$.

Algorithm 3 Exact n-cover scalar multiplication

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

Table 2. A two-dimensional representation of `u2c`

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

Remark 1. For convenience, we presented a recursive version of Algorithm 3. However this recursion is a tail-end-recursion so that the algorithm can easily be reformulated iteratively. It can thus be immediately implemented or small embedded devices which may not support recursion. To do so, first compute a sequence (m_i, r_i) for k using the covering system of our choice such that

$$k = r_0 + m_0(r_1 + m_1(\dots(r_t + m_t)\dots)).$$

The scalar multiplication $[k]P$ then simply consists of traversing this sequence in reverse order. Initialize $Q := \mathcal{O}$, then for each pair (m_i, r_i), set $Q := [m_i]Q + [r_i]P$. (Note that the elements $[r_i]P$ can be precomputed.) Generating the sequence (m_i, r_i) from k only requires integer arithmetic. In Section 5, we give some advices for performing this encoding efficiently.

3.2 Complexity Analysis

In this section, we analyze the complexity of our algorithm on the average using first order Markov chains with an approach similar to the analysis presented in [1]. This allows us to determine the average number of steps of our algorithm. We then deduce a theoretical number of field operations on the average.

Let $\mathcal{S} = \{s_1, \dots, s_k\}$ with $s_i := r_i \pmod{m_i}$ and let $\ell = \mathrm{lcm}(m_1, \dots, m_k)$. We define a Markov chain, i.e. a transition graph, such that:

- the vertices s_i are the ℓ congruence classes modulo ℓ.
- the edges (s_i, s_j) are oriented and labeled with probabilities: for $k > 0$, we know that there is at least one[3] congruence class $s_i := r_i \pmod{m_i} \in \mathcal{S}_{(k)}$ such that $k \equiv r_i \pmod{m_i}$. In line

[3] When $\mathcal{S}$ is an exact n-cover, there are exactly n such classes.

4 of Alg. 2, we compute $k' = (k - r_i)/m_i$ and call the algorithm recursively with k' 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 (s_i, s_j) is labeled with the conditional probability $P(k' \equiv r_j \pmod{m_j} | k \equiv r_i \pmod{m_i})$.

Given a covering system $\mathcal{S}$, the corresponding transition matrix can be computed using Algorithm 4. If integer i is covered by more than one congruence class, we assume that the choice of which covering class is selected is computed uniformly at random, hence with probability $1/|\mathcal{S}_{(i)}|$.

Algorithm 4 Markov chain

Input: $\mathcal{S}$ a covering system of congruences
Output: The transition matrix A associated to $\mathcal{S}$

```
A := (0)
for i = 0, ..., ℓ − 1 do
    for (r, m) ∈ S_(i) do
        for j = 0, ..., ℓ − 1; j ≡ (i − r)/m (mod ℓ/m) do
            A_{i,j} := A_{i,j} + 1/(|S_(i)|m)
return A
```

Lemma 1. *The transition matrix produced by Alg. 4 is stochastic, i.e. each row of A adds up to* 1.

Proof. For all $i = 0, \ldots, \ell - 1$, and for each $(r, m) \in \mathcal{S}_{(i)}$, there are exactly m congruence classes such that $j \equiv (i - r)/m \pmod{\ell/m}$. □

As an example, we consider the covering system given by the congruences classes

$$0 \pmod 2;\ 0 \pmod 3;\ 1 \pmod 6;\ 5 \pmod 6).$$

We have $\ell = \mathrm{lcm}(2, 3, 6) = 6$. The corresponding $\ell \times \ell$ transition matrix given by Algorithm 4 is:

$$A = \begin{pmatrix} 5/12 & 0 & 1/6 & 1/4 & 1/6 & 0 \\ 1/6 & 1/6 & 1/6 & 1/6 & 1/6 & 1/6 \\ 0 & 1/2 & 0 & 0 & 1/2 & 0 \\ 0 & 1/3 & 0 & 1/3 & 0 & 1/3 \\ 0 & 0 & 1/2 & 0 & 0 & 1/2 \\ 1/6 & 1/6 & 1/6 & 1/6 & 1/6 & 1/6 \end{pmatrix}. \tag{2}$$

In order to evaluate the average complexity of our CSC-based scalar multiplication algorithm, we are interested in the so-called stationary probability vector of the Markov chain. Let A be the transition matrix of a Markov chain. Then, its stationary probability π_∞ satisfies $\pi_\infty A = \pi_\infty$. It is known that $\pi_\infty = \lim_{n\to\infty} \pi A^n$, for any probability distribution π. In practice, it can be computed more efficiently. Let $B = \lim_{n\to\infty} A^n$. We have

$$BA = \lim_{n\to\infty} A^n A = \lim_{n\to\infty} A^{n+1} = \lim_{n\to\infty} A^n = B.$$

And thus $B(A - Id) = 0$. This equality can be seen as the union of several linear systems of equations, each of the form $BC_j = 0$. Since B is stochastic, we also have a system of stochastic

equations given by $B\mathbb{1}^t = \mathbb{1}^t$, where $\mathbb{1}$ is the 1-vector. A known trick[4] then consists of replacing the last system of equations $BC_j = 0$ by these stochastic equations. Denoting by f the application which replaces the last column of a matrix by $\mathbb{1}^t$, we get $Bf(A - Id) = f(0)$ so that

$$B = f(0)\left(f(A - Id)\right)^{-1}. \tag{3}$$

If, in addition, the Markov chain is irreducible and aperiodic, π_∞ is unique and satisfies

$$\pi_\infty = (0, \ldots, 0, 1)\left(f(A - Id)\right)^{-1}. \tag{4}$$

We illustrate this average complexity analysis on the following covering system: 0 (mod 2); 0 (mod 3); $1, -1$ (mod 6) whose transition matrix was given in (2). Using (4) we first compute

$$\pi_\infty = (4/35,\ 1/5,\ 6/35,\ 1/7,\ 6/35,\ 1/5).$$

When $k \equiv 0 \pmod 2$, the scalar k is divided by 2 (Alg. 2, line 4) and $[2]Q$ is returned (Alg. 2, line 5). This occurs exactly when $k \equiv 0, 2, 4 \pmod 6$; hence with probability $4/35 + 6/35 + 6/35 = 16/35$. Similarly, when $k \equiv 0, 3 \pmod 6$, k is divided by 3 and the returned value is $[3]Q$ with probability $9/35$. Finally, when $k \equiv -1, 1 \pmod 6$, the algorithm returns $[6]Q \pm P$ with probability $1/5$ in both cases and k is divided by 6. Using these probabilities we deduce an average base

$$\beta = \sqrt[35]{6^{14} \times 3^9 \times 2^{16}} \approx 3.72874372148555.$$

Remark 2 (A few words about this peculiar value β). For the binary covering system given by the classes 0 (mod 2); 1 (mod 2), the above analysis gives $\beta = 2$. More generally, one gets $\beta = m$ for all m-ary covering systems. Thus, in those cases β corresponds to the base of the expansion used to decompose k. In the general case, the moduli of a covering system are not all equal. Thus β can be seen as an average base. As a consequence, the average number of steps of our algorithm is approximately $\log k / \log \beta$.

Going back to our example, the average number of steps is thus

$$\log_\beta 2 \times |k| \approx 0.526679019666220 \times |k|,$$

where $|k| = 1 + \lfloor \log_2 k \rfloor$ is the bitlength of k.

Now, we have seen that a point addition is performed with probability 2/5. It happens for $k \equiv -1, 1 \pmod 6$. The average number of point additions per bit is thus given by $2/5 \times \log_\beta 2 \approx 0.210671607866488$. Likewise, we can evaluate the average number of point doubling, tripling, etc. Finally, multiplying the above probabilities by the cost of each primitive (in terms of field multiplications), we obtain an average number of field multiplications per bit.

In Table 3 we give the theoretical cost per bit for several CSC-based scalar multiplication algorithms on two widely adopted families of elliptic curves. For the curve arithmetic, we considered the best operation counts reported in [4] (for Jacobian coordinates we used $a = -3$ and $S = 0.8M$). For the Montgomery ladder on Weierstrass curves, we use the differential co-Z addition-and-doubling algorithm reported in [17, Algo. 5] of approximate cost $10M + 5S \simeq 14M$. 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 $[h]([m/h]Q + [r/h]P)$.

[4] Yet rarely explained clearly!

Table 3. Theoretical operation counts for various covering systems

(a) Short Weierstrass

Algo.	#mult./bit	#Precomp.
u12c	11.13	20+
u3c	12.29	2
u4c2	14.64	8
u4c1	12.66	7
cs3-2b	12.67	9
u3c2	12.91	3
cs3-2	12.92	7
u8c	13.10	20+
u6c	13.35	19
Montg. ladder	14.00	0
cs6	14.08	11
u2c	14.24	1
cs4	14.48	4
cs3	15.06	2
cs5	15.59	8
cs2	16.05	0
cs3-3	16.87	8

(b) Inverted Edwards

Algo.	#mult./bit	#Precomp.
u12c	12.00	20+
u8c	12.85	20+
u3c	12.88	2
cs3-2b	12.98	9
u4c2	13.02	8
u4c1	13.04	7
cs3-2	13.11	7
u3c2	13.28	3
cs6	13.46	11
cs4	13.94	4
u2c	14.17	1
cs3	14.26	2
u6c	14.59	19
cs3-3	14.64	8
Montg. ladder	16.00	0
cs5	16.18	8
cs2	16.60	4

Remark 3. When $\mathcal{S}$ is an exact n-cover, the above analysis can be greatly simplified. The following theorem holds.

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

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

We need the following Lemma.

Lemma 2. *Let $\mathcal{S}$ be an exact n-cover. Then, the transition matrix A produced by Alg. 4 is doubly stochastic, i.e. each row and column of A add up to 1.*

Proof (Lemma). 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 M/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-columns, 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 $1/nm$. □

Proof (Theorem). Since A is doubly stochastic, $\mathbb{1}A = \mathbb{1}$. Thus, the uniform probability distribution $\pi = (1/\ell)\mathbb{1}$ satisfies $\pi A = \pi$, which proves Theorem 1. □

4 Resistance to Side-Channel Attacks

In this section we try to analyze the robustness of our approach for different types of side-channel attacks, namely: differential attacks, simple attacks, timing attacks and horizontal attacks. We formulate claims about the level of protection of our algorithm and suggest open questions related to possible attacks.

4.1 Differential Attacks

As stated in Section 2, differential attacks, and more generally vertical attacks, may be defeated using randomization techniques such as Coron's blinding countermeasures [11]. Our approach uses a different strategy; instead of masking the inputs, it aims at randomizing the algorithm itself. Algorithms of that type are scarce. As already mentioned, the BSD randomization attempt [15,12] has been broken; its inherent weakness was established in [14]. In that paper Fouque et al. presented a collision attack using the fact that the probabilities of the j-th BSD digit (trit) when $k_j = k_{j+1}$

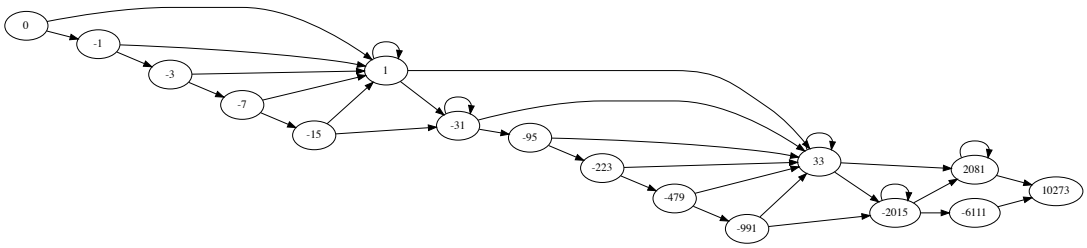


Fig. 2. The graph of internal states of the BSD randomized scalar multiplication for $k = 10273$.

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 [14] 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. 2.

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".*

In that regard, our algorithm offers a much better protection. Indeed, from each internal state, our algorithm guarantees that there exists exactly n transitions[5]; they are given by the n congruence classes covering that integer. By picking one of these n classes uniformly at random, a probability analysis similar to that from [14] will not reveal anything. Moreover, even if one could somehow observe that a given congruence class is used twice consecutively, the explicit mapping with the bits of the scalar is lost after the first division (line 4 of Alg. 2) by a non-power of 2.

In Fig. 3, we give the transition graph for $k = 10273$ obtained with an exact 3-cover. It should be compared to the graph in Fig. 2 obtained for the same scalar using the BSD randomization.

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 $B = A + A^2 + \cdots + A^t$, and where t 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.

Claim 1. Thanks to its high level of randomization, our CSC-based algorithm is immune to differential attacks.

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

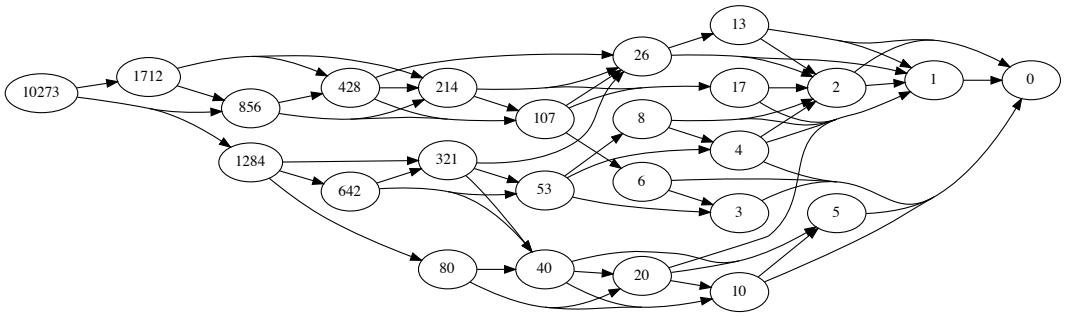


Fig. 3. The graph of internal states of an exact 3-cover for $k = 10273$.

4.2 Timing Attacks

Timing attacks are side-channel attacks that exploit dependencies between the execution time of an algorithm and some secret data manipulated through its running. Such attacks may be prevented by ensuring that the computation time is independent from that secret value ; a property commonly guaranteed by so-called constant time algorithms.

Clearly, our algorithm does *not* run in constant time. This is deliberate. It was designed to resist to horizontal attacks that precisely exploit the perfect regularity of constant time algorithms (see Section 4.4). At first glance, this seems redhibitory. But let us recall Kocher's advices from [19, Section 9, p. 7]:

> *"The most obvious way to prevent timing attacks is to make all operations take exactly the same amount of time. Unfortunately this is often difficult. [...]* ***Another approach is to make timing measurements so inaccurate that the attack becomes unfeasible."***

So, the question is: *how much information can be deduced from the running time of our algorithm?*

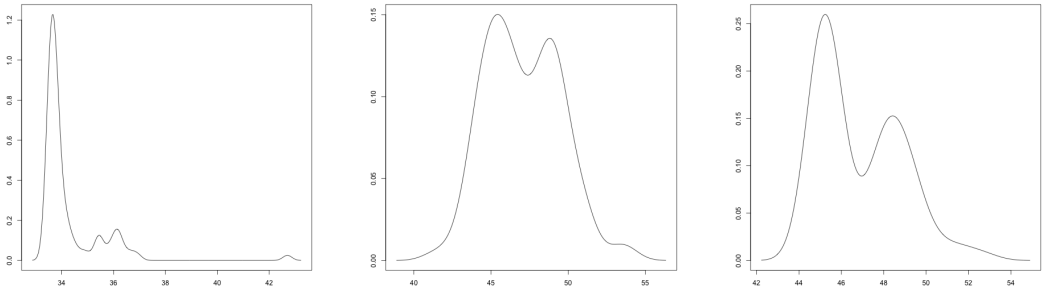
First, observe that dependency on the length of k is easily addressed. As suggested in [5], it suffices to replace k by an equivalent value $\hat{k}$ of fixed bitlength. This can be achieved by adding independently m and $2m$ to k and choosing for $\hat{k}$ that result of bitlength $1+|m|$, where $m = \mathrm{ord}(P)$ and $|m|$ stands for the bitlength of m.

So now, the question becomes: *how much information can be deduced from the running time of our algorithm for scalars of the same size?*

Even in ideal conditions, constant time algorithms do *not* run in constant time. Little time variations may be observed because of compiler optimizations, cache misses, processor instructions that run in non-fixed time, etc. With our CSC-based randomized algorithm, the time for computing $[k]P$ also fluctuates depending on the path from k to 0 that is taken in the course of the algorithm (see Figure 3).

To illustrate this fluctuation, we performed one thousand computations $[k]P$ using the same scalar k with both the double-and-add and our CSC-based algorithm and recorded the respective

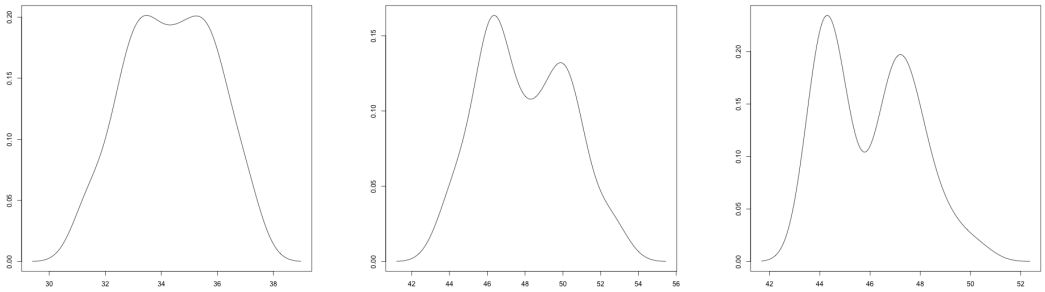
lists of computation times. We show the corresponding density functions in Figure 4. These results are not surprising. For the double-and-add algorithm (Fig. 4(a)), the minor time variations come from measurement noise. In contrary, our CSC-based algorithm (Fig. 4(b)) produces much bigger time variations because of its inherent randomization. For completeness, Figure 4(c) shows the density function for the Montgomery ladder. It is somewhat similar to 4(b).



(a) Double-and-Add (b) CSC `cs3-2b` (c) Montgomery ladder

Fig. 4. Time distributions over 1000 computations $[k]P$ with the same scalar k using the double-and-add algorithm, an exact 3-cover and a constant time implementation of the Montgomery ladder.

We then performed another thousand computations $[k]P$ by selecting a new random scalar k for each computation. (In order to obtain fair comparisons, the same random scalars were used for all three algorithms.) We present the density plots for the computation time in Figure 5.



(a) Double-and-Add (b) CSC `cs3-2b` (c) Montgomery ladder

Fig. 5. Time distributions over 1000 computations $[k]P$ with 1000 random scalars k using the double-and-add algorithm, an exact 3-cover and a constant time implementation of the Montgomery ladder.

For the double-and-add algorithm, we see a clear difference between the time distributions in Figures 4(a) and 5(a). It confirms that the computation time is very much dependent upon the scalar k. In return, the differences between Figures 4(b) and 5(b) on the one hand and between Figures 4(c) and 5(c) on the other hand are much more tenuous. The resemblance between the two Montgomery ladder curves comes from its constant time property. For our CSC-based algorithm, our assumption is that the high level of randomization of our algorithm tends to reduce the dependency between the computation time and the scalar. We thus strongly believe that our CSC-based algorithms are much less prone to timing attacks than other non constant time algorithms.

To reinforce the argument, let us consider a successful timing attack on an OpenSSL implementation of ECDSA proposed in [5] by Brumley and Tuveri. The attack targets the signature phase of ECDSA, in particular a scalar multiplication $[k]P$, where nonce k is selected uniformly at random. The attack exploits the dependency between the computation time and the bitlength of k. It operates in two phases: first, using the time dependency, a certain amount of signatures coming from "short" scalars are collected. Then, the missing bits are recovered using lattice based techniques. The attack works because the first phase is effective. The filtered signatures correspond to scalars shorter than some fixed threshold with very high probability. As mentioned in [5], the attack success rate decreases dramatically with the increase of false positives.

We claim that Brumley and Tuveri's attack is unfeasible with our randomized algorithm. Indeed, if one filters signatures based on the computation time, it is very unclear whether some useful characteristic can be exploited to reduce the complexity of the lattice reduction phase. In particular, there does not seem to be any immediate correlation between those filtered scalars which have led to the fastest computations.

Claim 2. Although not *provably* secure, we conjecture that our CSC-based algorithm seriously hinders the applicability of timing attacks.

Challenge 3. Find useful correlation between the elements of the set of filtered signatures (using the computation time of our algorithm) in order to reduce the complexity of the lattice reduction phase.

4.3 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 segments which correspond to doublings from those which correspond to general additions. Indeed, since a bit 1 corresponds to a doubling followed by an addition, whereas a bit 0 corresponds to a doubling, recovering the entire exponent from that trace is immediate.

As stated in Section 3, 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 Figure 3 (given in Appendix A) , a possible sequence for $k = 10273$ is: $(1 \pmod{12}, 0 \pmod 4, 10 \pmod{12}, 5 \pmod{12}, 1 \pmod{12})$. It correspond to the path

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

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

$$[10273]P = P + [12]([4]([10]P + [12]([5]P + [12](P + [12]\mathcal{O})))) \tag{6}$$

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 $[m_i]P$, the execution trace $T(k)$ looks like:

$$T(10273) = \texttt{DADDADADDADADDADDDADDA}. \tag{7}$$

Clearly, the mapping T from k to the pattern of the trace 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. It is not difficult to check that the above expressions map to the trace given in (7).

However, this property might not be sufficient to guarantee that the mapping T cannot be inverted. From the congruence classes of `u3c` listed in Appendix A, one remarks that the pattern `DADDA`, which repeats three times at the beginning of trace (7), can only correspond to one of the 6 congruence classes modulo 12. The ending pattern `DDDADDA` however, may be split into `DD` followed by `DADDA` or into `DDDA` followed by `DDA`. The former option corresponds to any combination of the form 0 (mod 4), $a \neq 0 \pmod{12}$, whereas the latter agrees with any combination of the form $a \neq 0 \pmod 8$, 3 (mod 4). For large k, the combinatorial explosion will probably be too important to recover the correct scalar with reasonably high probability. Nevertheless, for some covering systems, pattern matching techniques may be applied to significantly reduce the complexity of a brute force attack. Fortunately, there exist several implementation options that would safeguard our algorithm against simple attacks.

Unified Group Law: If the group admits a complete group law, no particular care seems to be necessary since addition and doubling cannot be differentiated. Hence, the computation trace only shows a sequence of identical operations, namely `AAA...AA`. There are several recognized families of elliptic curves that enjoy this property. As shown in Table 3(b) for Edwards curves, our covering system approach offers an excellent ratio performance/robustness in that case.

Side-Channel Atomicity: For (hyper)elliptic curves, another option is to use the concept of side-channel atomicity first suggested by Chevalier-Mames et al. [8]. An atomic block Γ is a sequence of (possibly dummy) field operations, such that all curve operations (addition, doubling, tripling, etc.) can be expressed using a repetition of that block Γ. The different curve operations involved in the scalar multiplication are thus indistinguishable. Note that this approach offers the ability to implement optimized formulæ for small scalar multiplication $[m_i]P$ for each modulo appearing in the covering system. For example, the fast m-tupling formulæ proposed in the literature [4], mainly for genus 1 curves, may be used to increase efficiency.

Regular Double Ladder: If one does not precompute the point $[r_i]P$, a third (virtual) approach would be to implement $[m_i]Q + [r_i]P$ using a regular double ladder. An efficient regular variant of Shamir's double ladder was proposed in [9]; unfortunately the proposed algorithm is incorrect. The various known alternatives require at least one extra register and are thus less efficient. If such a regular efficient double ladder does exist, it seems perfectly suited to our CSC-based algorithm.

Low Level Randomization: The trace $T(10273)$ in (7) was obtained by considering a left-to-right double-and-add algorithm for computing $[m_i]Q$ and precomputed points $[r_i]P$. To further randomize the algorithm, we could implement several efficient addition chains for those small values (m_i, r_i) and select one of those at random at each step.

Claim 4. Some care must be taken but several implementation options do exist to protect our algorithm against simple attacks.

Challenge 5. Let us assume that the attacker knows the entire sequence of moduli m_i but does not know the corresponding sequence of residues r_i. What could be the idea behind a generic attack (which does not depend on the covering system) to recover some useful information on the scalar?

4.4 Horizontal Attacks:

As seen above, several approaches can be used to ensure that our algorithm behaves very regularly at the field level. However, it is no longer the case when we analyze its behaviour at an upper level. Indeed, for any covering system $\mathcal{S}$, we have seen that our algorithm reduces to a sequence of computations of the form $[m_i]Q + [r_i]P$, where the possible values for m_i and r_i depend uniquely on $\mathcal{S}$. Thus, the number of clock cycles necessary to compute $[m_i]Q + [r_i]P$ for all possible pairs (m_i, r_i) is not constant.

This characteristic constitutes a major difference between our approach and all existing constant time algorithms. Because of this inherent, pursued irregularity, the horizontal spliting of the trace used in [22] to break an RSA implementation is not applicable to break our algorithm.

Besides, even if it was possible to split the trace properly, with parts of different sizes, revealing correlations between sections of that trace, of various lengths, which corresponds to different computations, seem very difficult.

Claim 6. Our algorithm is robust against the recently proposed horizontal attacks.

5 Experimental Results

In order to get a better insight on its practical efficiency, we implemented our algorithm and ran some simulations with various elliptic curves[6] and many covering systems of congruences. In Table 4, we give some timings obtained with our proof-of-concept implementation using curve ANSI FRP256V1 defined over a 256-bit prime field ; one of the safe standardized elliptic curves according to `http://safecurves.cr.yp.to/`. (Results obtained with other curves are similar.) The numbers given in Table 4 correspond to an average time taken over 100 independent experiments. Each experiment consists of 10 computations $[k]P$ with the same scalar k. Since our algorithm is randomized, we reset the random seed to a fixed value before each experiment so that each of these 10 computations follow the exact same path along the transition graph (see Figure 3).

We acknowledge that this is a very preliminary implementation. In particular the field arithmetic is very basic. The absolute values from Table 4 are therefore not very relevant per se. What is important, however, is the comparison with the Montgomery ladder which shows that our algorithm can be competitive.

[6] For those experiments, we only considered short Weierstrass curves defined over prime fields.

Table 4. Average computation time of several covering systems.

Algorithms	Avg. time (in ms)
`u3c`	43.83
`u2c`	44.05
Montg. ladder	46.04
`u4c2`	46.50
`cs3-2b`	47.96
`u12c`	48.84

The cost of integer arithmetic, intentionally put aside for the theoretical analysis, partly explains the small inconsistencies with the theoretical results presented in Tables 3. Two operations in algorithm 3 deserves some attention: $k \bmod \ell$ 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 $\ell = \gcd(m_1, \ldots, m_{|\mathcal{S}|})$ factors into many different primes, it seems more profitable 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 [6]. The exact division by m in line 9 can also be implemented very efficiently, for example using Jebelean's exact division[7]. Our conclusion is that the extra cost induced by integer arithmetic can be significantly reduced. In a fine implementation, it should be negligible.

6 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 $\{m_1, \ldots, m_k\}$ and a covering degree n, the problem consists of assigning integer values to $r_1, \ldots, r_k$ such that:

i) $r_i \in \{0, \ldots, m_i - 1\}$ for all $i \in \{1, \ldots, k\}$;
ii) $m_i = m_j \Rightarrow r_i \neq r_j$ for all $i, j \in \{1, \ldots, k\}$; and
iii) 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.

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`. 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 Figure 6. This way, we also ensure that our covering systems do not contain the whole set of congruence classes for a given modulo such as 0 (mod 2); 1 (mod 2).

[7] See for example GMP's exact division by 3 in `mpn_divexact_by3`.

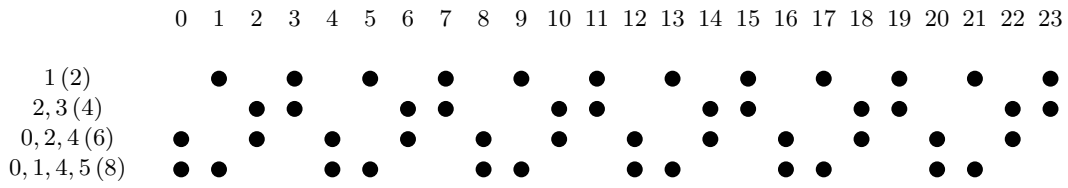


Fig. 6. 1 (mod 2); 2, 3 (mod 4); 0, 2, 4 (mod 6); 0, 1, 4, 5 (mod 8) is a uniform exact 2-cover. Each modulo covers exactly 12 integers out of 24.

We acknowledge that this generation strategy is pretty basic. It proved sufficient to generate exact n-covers of reasonably large covering degree[8] but can certainly be improved using more sophisticated tools, for example using efficient heuristics for constraint satisfaction problems.

7 Conclusions

We proposed a new randomized, fully generic, scalar multiplication algorithm with built-in protections against various types of side-channel attacks. Our experimental results are presented for elliptic curves but our approach immediately translates to other types of finite groups used in public key cryptography. Randomization is provided by the use of exact covering systems of congruences. Even for small covering degrees, we showed that it guarantees a very large number of possible internal states and a large number of possible transitions from each state.

We provided solid arguments and claims regarding the security level of our generic algorithm against differential attacks, timing attacks, simple attacks, and horizontal attacks. Based on our proof-of-concept implementation, the speed performance of our algorithm is encouraging. For our experiments, we generated exact n-covers using a greedy strategy. This can certainly be improved. Short-term future work includes an optimized implementation and the generation of optimal covering systems.

In Section 4, we left some open challenges for cryptanalysts. If they happen to remain unsolved, our algorithm could be considered as an extra step towards the ultimate, all-in-one protection.

References

1. Adikari, J., Dimitrov, V., Imbert, L.: Hybrid binary-ternary number system for elliptic curve cryptosystems. IEEE Transactions on Computers 60(2), 254–265 (2011)
2. Bajard, J.C., Imbert, L., Liardet, P.Y., Teglia, Y.: Leak resistant arithmetic. In: Cryptographic Hardware and Embedded Systems, CHES'04. Lecture Notes in Computer Science, vol. 3156, pp. 62–75. Springer (2004)
3. Bauer, A., Jaulmes, E., Prouff, E., Wild, J.: Horizontal collision correlation attack on elliptic curves. In: Selected Areas in Cryptography - SAC 2013 - 20th International Conference, Revised Selected Papers. pp. 553–570 (2013)
4. Bernstein, D.J., Lange, T.: Explicit-formulas database. URL: `http://www.hyperelliptic.org/EFD/`, joint work by Daniel J. Bernstein and Tanja Lange, building on work by many authors.

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

5. Brumley, B.B., Tuveri, N.: Remote timing attacks are still practical. In: Computer Security, ESORICS 2011. Lecture Notes in Computer Science, vol. 6879, pp. 355–371. Springer (2011)
6. Chabrier, T., Tisserand, A.: On-the-fly multi-base recoding for ECC scalar multiplication without precomputations. In: 21st IEEE Symposium on Computer Arithmetic, ARITH 2013. pp. 219–228. IEEE Computer Society (2013)
7. Chari, S., Rao, J.R., Rohatgi, P.: Template attacks. In: Cryptographic Hardware and Embedded Systems, CHES'02. Lecture Notes in Computer Science, vol. 2523, pp. 13–28. Springer (2002)
8. Chevalier-Mames, B., Ciet, M., Joye, M.: Low-cost solutions for preventing simple side-channel analysis: Side-channel atomicity. IEEE Transactions on Computers 53(6), 760–768 (Jun 2004)
9. Ciet, M., Joye, M.: (virtually) free randomization techniques for elliptic curve cryptography. In: Information and Communications Security, 5th International Conference, ICICS 2003, Proceedings. Lecture Notes in Computer Science, vol. 2836, pp. 348–359. Springer (2003)
10. Clavier, C., Feix, B., Gagnerot, G., Giraud, C., Roussellet, M., Verneuil, V.: ROSETTA for single trace analysis. In: Progress in Cryptology - INDOCRYPT 2012, 13th International Conference on Cryptology in India, Proceedings. pp. 140–155 (2012)
11. Coron, J.S.: Resistance against differential power analysis for elliptic curve cryptosystems. In: Cryptographic Hardware and Embedded Systems, CHES'99. vol. 1717, pp. 292–302 (1999)
12. Ebeid, N.M., Hasan, M.A.: On binary signed digit representations of integers. Designs, Codes and Cryptography 42(1), 43–65 (2007)
13. ElGamal, T.: A public key cryptosystem and a signature scheme based on discrete logarithms. IEEE Transactions on Information Theory 31(4), 469–472 (Jul 1985)
14. Fouque, P.A., Muller, F., Poupard, G., Valette, F.: Defeating countermeasures based on randomized bsd representations. In: Cryptographic Hardware and Embedded Systems, CHES 2004. vol. 3156, pp. 312–327. Springer (2004)
15. Ha, J., Moon, S.J.: 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, pp. 551–563 (2002)
16. Hough, B.: Solution of the minimum modulus problem for covering systems. arXiv:1307.0874v2 [math.NT] (2014), `http://arxiv.org/abs/1307.0874`
17. Hutter, M., Joye, M., Sierra, Y.: Memory-constrained implementations of elliptic curve cryptography in Co-Z coordinate representation. In: Progress in Cryptology, AFRICACRYPT 2011. Lecture Notes in Computer Science, vol. 6737, pp. 170–187. Springer (2011)
18. Kocher, P., , Jaffe, J., Jun, B.: Differential power analysis. In: Advances in Cryptology, CRYPTO'99. pp. 388–397. Springer (1999)
19. Kocher, P.C.: Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In: Advances in Cryptology, CRYPTO '96. vol. 1109, pp. 104–113 (1996)
20. Longa, P., Miri, A.: 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/`
21. Möller, B.: Improved techniques for fast exponentiation. In: Information Security and Cryptology, ICISC 2002. Lecture Notes in Computer Science, vol. 2587, pp. 298–312. Springer (2003)
22. Perin, G., Imbert, L., Torres, L., Maurine, P.: Attacking randomized exponentiation using unsupervised learning. In: Constructive Side-Channel Analysis and Secure Design - 5th International Workshop, COSADE 2014. Revised Selected Papers. Lecture Notes in Computer Science, vol. 8622, pp. 144–160. Springer (2014)
23. Rivain, M.: Fast and regular algorithms for scalar multiplication over elliptic curves. Cryptology ePrint Archive, Report 2011/338 (2011), `http://eprint.iacr.org/`
24. Rivest, R., Shamir, A., Adleman, L.: A method for obtaining digital signatures and public key cryptosystems. Communications of the ACM 21(2), 120–126 (Feb 1978)
25. Solinas, J.A.: Efficient arithmetic on koblitz curves. Designs, Codes and Cryptography 19(2-3), 195–249 (2000)

26. U.S. Department of Commerce / National Institute of Standards and Technology (ed.): Digital Signature Standard (DSS). National Institute of Standards and Technology, Washington (Jan 2013), http://csrc.nist.gov/publications/fips/. Note: Federal Information Processing Standard 186-4

A The uniform exact 3-cover `u3c`

$$
\begin{aligned}
0 \pmod 2 \\
0, 3 \pmod 4 \\
1, 3, 5 \pmod 6 \\
0, 1, 6, 7 \pmod 8 \\
1, 2, 5, 6, 9, 10 \pmod{12} \\
2, 3, 4, 5, 10, 11, 12, 13 \pmod{16}
\end{aligned}
$$