Randomizing scalar multiplication using exact covering systems of congruences

No Author Given

No Institute Given

Abstract. A covering system of congruences can be defined as a set of congruence relations of the form: $\{r_1 \pmod{m_1}, r_2 \pmod{m_2}, \ldots, r_t \pmod{m_t}\}$ for $m_1, \ldots, m_t \in \mathbb{N}$ satisfying the property that for every integer k in $\mathbb{Z}$, there exists at least an index $i \in \{1, \ldots, t\}$ such that $k \equiv r_i \pmod{m_i}$. First, we 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 that may be used to counter differential side-channel attacks, and more generally physical attacks that require multiple executions of the algorithm. This algorithm can be an alternative to Coron's scalar blinding technique for elliptic curves, in particular when the choice of a particular finite field tailored for speed compels to use a large random factor.

Keywords: Scalar multiplication, side-channel attacks, randomized algorithms, covering systems of congruences

1 Introduction

Exponentiation in multiplicative subgroups of finite fields and jacobians of low genus hyperelliptic curves is a crucial operation for public key cryptosystems. 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 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 [20], collision attacks [11,3], template attacks [8], 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 research 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, 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 SCA at once should be considered with great interest. The ultimate, all-in-one, protection is yet to be discovered!

The type and level of countermeasures that should be implemented depend on the security model. In the software model, one mainly focuses on timing and cache attacks. In the more advanced hardware model, and for software implementations that run on micro-controllers, the adversary has entire control of the device. In order to retrieve (part of) the secret manipulated by the cryptographic algorithm, she can monitor various sources of leakage (power consumption, time, electromagnetic radiations, noise, etc.) and insert faults. These two strategies are respectively classified into passive and active attacks.

This paper deals with passive attacks in the hardware model. In this case, the most prominent 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 proposed by Coron [12] (scalar randomization, base point blinding, random projective coordinates) may be used to counteract differential power/EM analysis, refined power analysis, zero-value analysis, etc. Timing attacks may be thwarted using regular algorithms such as the Montgomery ladder and/or curves allowing a unified/complete group law. For a good survey on secure ECC implementations, see [14,15].

A less frequently considered 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 [25] or the Leak Resistant Arithmetic (LRA) concept by Bajard et al. [2] are two examples of such randomized algorithms proposed in the RSA context. In the elliptic curve setting, an approach of that type based on Binary Signed Digit (BSD) recodings [16,13], initially regarded as very promising, was eventually broken in [23].

Among the randomization options proposed by Coron, scalar randomization is probably the most important. It may be used to block several type of passive attacks [14,15]. The private scalar is masked by adding a random multiple of the group order. Let E be an elliptic curve of order $\#E$ and P a point on E. Instead of computing $[k]P$, one computes the same point as $[k + r\#E]P$ for some random integer r. The bits of k are thus masked using a different random value at each execution. At first, the computational overhead of this simple countermeasure seems rather low. However, this is not always the case and one has to be very careful. For the sake of efficiency, elliptic curves are usually defined over particular 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 of the curve is also of that special form. In order to blind all the bits of k, a random value r of roughly the size of k may be needed. For example Bernstein recommends a 256-bit random value, i.e. a 512-bit scalar, for curve25519 [4], yielding a 100% overhead! With that in mind, solutions based on randomizing the addition chain may provide efficient alternatives.

The main contribution of this paper is a new randomized scalar multiplication algorithm based on covering system of congruences (CSC). It is presented in Section 2. Interestingly, many algorithms from the literature can be expressed in this CSC framework. In Section 2.3 we analyze its complexity on the average from which we deduce a theoretical operation counts per bit. We discuss its robustness against differential attacks in Section 3 and suggest some possible implementation strategies to further harden simple and timing attacks. Finally, in Section 5 we present a generic greedy approach to generate such covering systems.

2 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.

2.1 Covering systems of congruences

A *covering system of congruences* (CSC) is a finite set $\{r_1 \pmod{m_1}, \ldots, r_t \pmod{m_t}\}$ such that every integer satisfies at least one of its congruence relations. 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 the set:

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

A covering system is called an *n-cover* if every integer is covered at least n times ; it is called 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. The covering property is guaranteed as soon as all the integers modulo $\mathrm{lcm}(2, 4, 6, 8) = 24$ are covered.

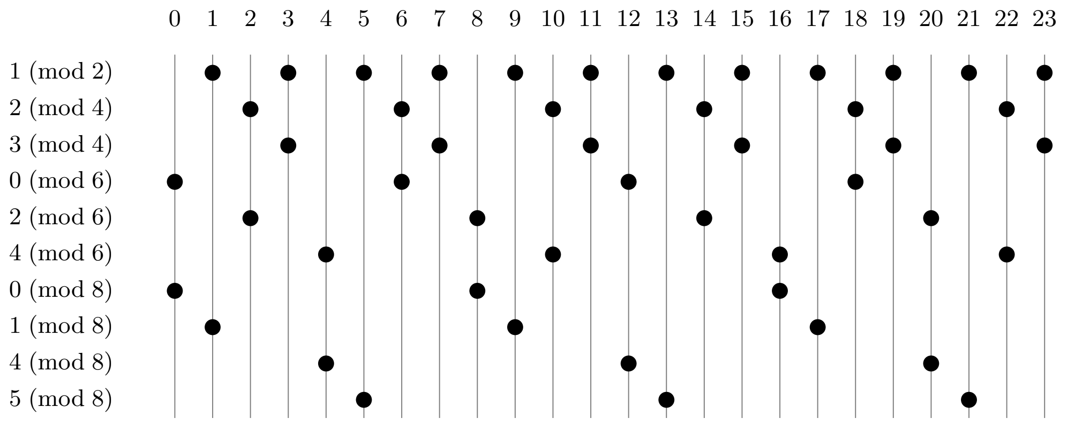

Fig. 1. The set $\{1 \pmod 2;\ 2, 3 \pmod 4;\ 0, 2, 4 \pmod 6;\ 0, 1, 4, 5 \pmod 8\}$ is an exact 2-cover. Each row corresponds to a congruence class ; the dots indicate which integers are covered by each class. The number of dots in each column indicates the number of congruence classes covering the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

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 [17]. 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 $P \in E$ 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 trivial covering set $\{0 \pmod 2;\ 1 \pmod 2\}$. It instantly leads to

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

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

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

Algorithm	Covering System of Congruences
m-ary	$0, \dots, m-1\,(m)$
NAF	$0\,(2);\ 1, -1\,(4)$
w-NAF [24]	$0\,(2);\ 1, -1\,(2^w);\ 3, -3\,(2^w);\ \dots;\ 2^{w-1}-1, -2^{w-1}+1\,(2^w)$
frac. win. [22]	$0\,(2);\ 1, -1\,(2^{w+1});\ 3, -3\,(2^{w+1});\ \dots;\ 2^w+m;\ -2^w-m\,(2^{w+1})$
wmbNAF [21]	$0\,(a_1);\ \dots; 0\,(a_k);\ 1, -1\,(a_1^w);\ 3, -3\,(a_1^w);\ \dots, (a_1-1)/2\,(a_1^w)$
w-HBTF [1]	$0\,(2);\ 0\,(3);\ 1, -1\,(w);\ \dots;\ w/2-1, -w/2+1\,(w)$

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.

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 $r \pmod m \in \mathcal{S}$ such that $k \equiv r \pmod m$. A generic CSC-based scalar multiplication is given in Algorithm 1.

Algorithm 1 Generic CSC-based scalar multiplication

```
Input: $\mathcal{S}$, $k \in \mathbb{Z}, P \in E$
Output: $[k]P \in E$
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 validity of Algorithm 1 is based on the fact that, for any covering system $\mathcal{S}$, we have

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

Clearly, equation (2) 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 Alg. 1 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 possible congruence classes $r \pmod m$ in $\mathcal{S}_{(k)}$ for every integer k.
- Second, in line 3 of Alg. 1, 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 $\ell = \operatorname{lcm}(m_1, \dots, m_{|\mathcal{S}|})$. In Section 3 we shall see that this leads to exponentially many ways to compute Q in line 4.

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

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

is an exact n-cover, it may be stored in memory as a two-dimensional array of size $\ell \times n$, where $\ell = \operatorname{lcm}(m_1, \dots, m_t)$. For each $i \in \{1, \dots, \ell\}$, the entry $S[i]$ is a one-dimensional array, of length exactly n, whose elements are the congruence classes 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	$0 \pmod 8$	$0 \pmod 6$	12	$4 \pmod 8$	$0 \pmod 6$
1	$1 \pmod 8$	$1 \pmod 2$	13	$5 \pmod 8$	$1 \pmod 2$
2	$2 \pmod 4$	$2 \pmod 6$	14	$2 \pmod 4$	$2 \pmod 6$
3	$1 \pmod 2$	$3 \pmod 4$	15	$1 \pmod 2$	$3 \pmod 4$
4	$4 \pmod 8$	$4 \pmod 6$	16	$0 \pmod 8$	$4 \pmod 6$
5	$5 \pmod 8$	$1 \pmod 2$	17	$1 \pmod 8$	$1 \pmod 2$
6	$2 \pmod 4$	$0 \pmod 6$	18	$2 \pmod 4$	$0 \pmod 6$
7	$1 \pmod 2$	$3 \pmod 4$	19	$1 \pmod 2$	$3 \pmod 4$
8	$0 \pmod 8$	$2 \pmod 6$	20	$4 \pmod 8$	$2 \pmod 6$
9	$1 \pmod 8$	$1 \pmod 2$	21	$5 \pmod 8$	$1 \pmod 2$
10	$2 \pmod 4$	$4 \pmod 6$	22	$2 \pmod 4$	$4 \pmod 6$
11	$1 \pmod 2$	$3 \pmod 4$	23	$1 \pmod 2$	$3 \pmod 4$

We use this data structure in the description of Algorithm 2.

Algorithm 2 Exact n-cover scalar multiplication

```
Input: $S$ as described above, $\ell = \mathrm{lcm}(m_1, \dots, m_{|S|})$, $k \in \mathbb{N}, P \in E$
Output: $[k]P \in E$
 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 $\{0, \dots, n-1\}$
 7: $(r, m) := S[i][j]$
 8: compute $R := [r]P$
 9: compute $Q := [(k-r)/m]P$ recursively
10: return $[m]Q + R$
```

Remark 1. For convenience, we presented a recursive version of Algorithm 2. However this recursion is a tail-end-recursion so that the algorithm can easily be reformulated iteratively. It can thus be immediately implemented on 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 your choice such that

$$k = r_0 + m_0(r_1 + m_1(\dots(r_s + m_s)\dots)). \tag{3}$$

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 $[r_i]P$ may be precomputed. Generating the sequence (m_i, r_i) from k only requires integer arithmetic.

2.3 Complexity analysis

The average complexity of Algorithm 2 can be analyzed using a first order Markov chain. Let $\mathcal{S} = \{s_1, \dots, s_t\}$ with $s_i := r_i \pmod{m_i}$ and let $\ell = \mathrm{lcm}(m_1, \dots, m_t)$. We define the transition graph of the Markov chain as follow:

- the vertices s_i are the ℓ congruence classes modulo ℓ.
- the edges (s_i, s_j) 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 Alg. 2 (line 4 of Alg. 1 in the generic case), 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 (s_i, s_j) is labeled with the conditional probability $P(k' \equiv r_j \pmod{m_j} | k \equiv r_i \pmod{m_i})$.

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 $\pi_\infty A = \pi_\infty$. When $\mathcal{S}$ is an exact n-cover however, neither A nor π_∞ need be computed. (For completeness, we provide algorithms for computing both A and π_∞ in Appendix A.) The following theorem holds.

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

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

We need the following Lemma.

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

Proof (Lemma). 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 $j = 0, \dots, \ell - 1$. We will do so by showing that for all $j = 0, \dots, \ell - 1$

$$\sum_i A_{i,j} = \sum_{(r,m)\in\mathcal{S}} \frac{1}{nm} = \frac{1}{n} \sum_{(r,m)\in\mathcal{S}} \frac{1}{m}. \tag{4}$$

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 (4), observe that for each congruence class $(r, m) \in \mathcal{S}$, there are exactly ℓ/m integers i in $\{0, \dots, \ell - 1\}$ that are covered by (r, m). For each of those, there are exactly m integers j in $\{0, \dots, \ell - 1\}$ 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 $j \in \{0, \dots, \ell - 1\}$. 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 $i = 0, \dots, \ell - 1$ so that each contribution amounts to $1/nm$. $\square$

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

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

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 doubling, tripling, quintupling, etc., as well as the average number of point addition 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 $\sigma = ((r_i, m_i))_{i=0\dots s}$, be a precomputed sequence for the given scalar k. The iterative version of Algorithm 2 rewrites: set $Q = \mathcal{O}$ 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, 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 $\{0, \dots, \ell - 1\}$ 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. We denote by N_{p_i} the average number of scalar multiplications by p_i. Then, using the same arguments as above we get

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

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 explain the terminology in Section 5).

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

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 [5], 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 [18, 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 $5P$.

Table 3. Average operation counts per bit and number of precomputed points for short Weierstrass curves.

Covering system	#mult./bit	#Precomputations
u12c-2304-3315	8.84	355
u8c-432-600	9.33	66
cs3-48-48	9.84	8
u3c-48-24	9.87	2
cs3-48-38	9.95	7
cs6-72-103	9.95	12
u4c-48-60	9.96	7
u6c-120-168	10.48	19
cs3-54-47	10.61	8
cs4-24-37	10.64	4
cs3-24-23	10.93	2
u2c-24-10	11.23	1
cs5-60-73	11.5	8
cs2-30-19	11.8	4

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 2, 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 $\ell = \operatorname{lcm}(m_1, \ldots, m_{|\mathcal{S}|})$ 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 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 [7]. The exact division by m in line 9 can also be implemented very efficiently, for example using Jebelean's exact division[2].

3 Resistance to differential attacks

Algorithms based on randomizing the sequence of operations to compute $[k]P$ are rather scarce. As already mentioned in the introduction, the BSD randomization attempt [16,13] has been broken; its inherent weakness was established in [23]. 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 [23] 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 for $k = 10273$.

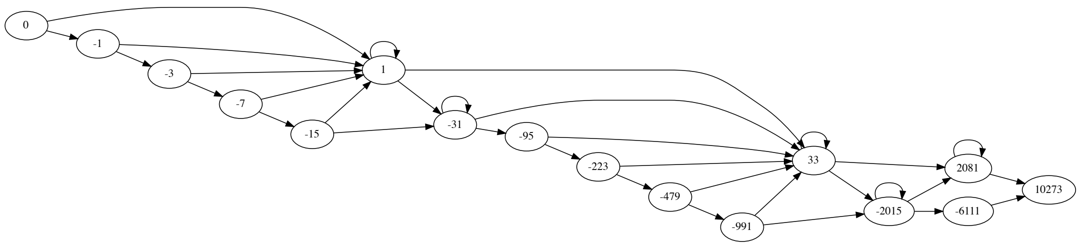

Fig. 2. The graph of internal states of the BSD randomized scalar multiplication for $k = 10273$.

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 satisfies both conditions. From each internal state, there exists exactly n transitions[3] given by the n congruence classes covering that integer. By choosing one of these n possible transitions uniformly at random, i.e. with probability $1/n$, our algorithm is locally robust. 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 Markov analysis from Section 2.3 provides some insight into the global randomness aspects of the algorithm. It is known that the stationary distribution $\pi_\infty = \lim_{n\to\infty} \pi A^n$ for any probability distribution π. A corollary of theorem 1 is that any infinite walk across the transition graph of the Markov chain ends on any congruence classes modulo ℓ with equal probability $1/\ell$. Therefore, the class of congruence modulo (any factor of) ℓ of the

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

[3] 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 $(m, r) \in \mathcal{S}_{(j)}$.

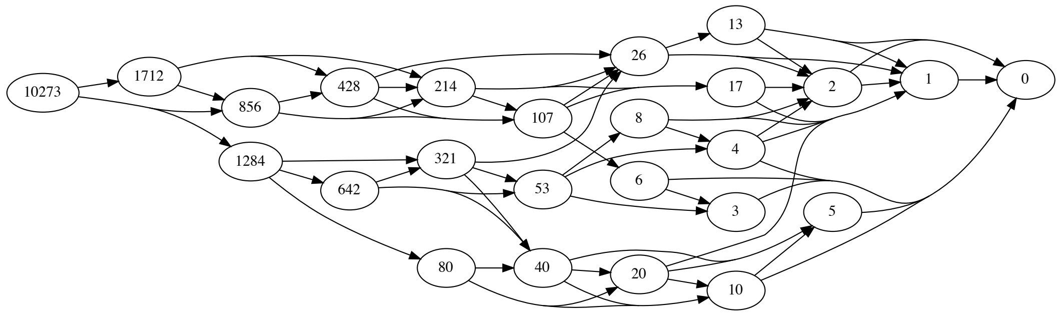

Fig. 3. The graph of internal states of an exact 3-cover for $k = 10273$.

internal state reached after sufficiently many steps is independent from the starting value k. This alone may not be sufficient, but certainly reinforce the resistance of our algorithm against attacks that require multiple executions of the algorithm.

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 randomization: an exact 2-cover produces 2^{61} paths for a 128-bit scalar and 2^{121} paths for a 256-bit scalar.

4 What about other attacks?

4.1 Timing attacks

In general timing attacks are mainly considered in the software model. However, as pointed out in [14] the threat should not be ignored in the hardware model.

Timing attacks require multiple execution of the algorithm. They exploit dependencies between the execution time of the algorithm and the 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 using constant time algorithms.

Clearly, in the present form, our algorithm does *not* run in constant time. The time for computing $[k]P$ may vary depending on the path from k to 0 taken at random in the course of the algorithm (see Figure 3). Let us recall Kocher's advices from [19, page 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."

Therefore, a natural question is: how much information can be deduced from the varying running time of our algorithm?

In [6], Brumley and Tuveri 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. 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 [6], the attack success rate decreases dramatically with the increase of false positives. The proposed countermeasure consists in replacing 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 $m = \operatorname{ord}(P)$ and $|m|$ stands for the bitlength of m. Obviously, the same countermeasure does apply to any scalar multiplication algorithm.

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

Exploring the correlations between the computation time of our algorithm and some useful information on the scalar is certainly a question of utmost importance. A possible strategy towards a more regular algorithm would be to implement each step using the same sequence of operations. For instance, this could be achieved by defining, for each $i \in \{0, \dots, \ell - 1\}$, a dedicated sequence of operation, possibly not optimal, for computing $[m_i]Q + [r_i]P$ for each $(m_i, r_i) \in \mathcal{S}_{(i)}$. The computational cost and effectiveness of such an approach will be the subject of further research.

4.2 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.

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 Figure 3, 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:

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

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:

$$T(10273) = \mathtt{D\,A\,D\,D\,A\,D\,A\,D\,D\,A\,D\,A\,D\,D\,A\,D\,D\,D\,A\,D\,D\,A}. \tag{6}$$

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 (6).

However, this property might not be sufficient to guarantee that the mapping T cannot be inverted. From the congruence classes of `u3c-48-24`, one remarks that the pattern `D A D D A`, which repeats three times at the beginning of trace (6), can only correspond to one of the 6 congruence classes modulo 12. The ending pattern `D D D A D D A` however, may be split into `D D` followed by `D A D D A` or into `D D D A` followed by `D D A`. The former option corresponds to any combination of the form 0 (mod 4), $a \neq 0$ (mod 12), whereas the latter agrees with any combination of the form $a \neq 0$ (mod 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. It is therefore important to add some countermeasures against simple attacks, for instance using any of the following implementation options. Note that the later two are specific to our algorithm.

Unified/complete 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 `A A A . . . A A`. There are several recognized families of elliptic curves that enjoy this property.

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. [9]. 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 [5], 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. Such a regular variant of the Straus-Shamir double ladder was proposed in [10] by Ciet and Joye; unfortunately their algorithm is incorrect. The various known alternatives require at least one extra register and are thus much 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 (6) 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 addition chains for those small values (m_i, r_i) and select one of those at random at each step.

5 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 $r_1, \dots, r_t$ such that:

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

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 $\{0, \dots, \ell\}$ 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, and $\ell = \text{lcm}(m_1, \dots, m_t)$.

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 unc–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 Figure 4. This way, we also ensure that our covering

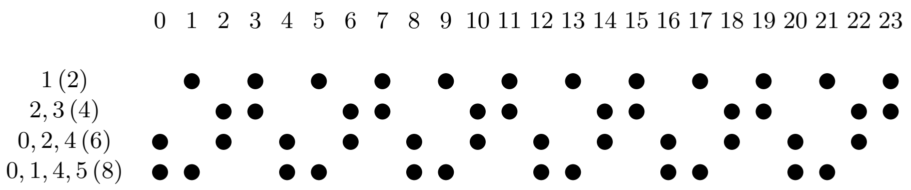

Fig. 4. 1 (mod 2); 2, 3 (mod 4); 0, 2, 4 (mod 6); 0, 1, 4, 5 (mod 8) is a uniform exact 2-cover. Each modulo covers exactly 12 integers out of 24.

systems do not contain the whole set of congruence classes for a given modulo such as 0 (mod 2); 1 (mod 2). We acknowledge that this generation strategy is pretty basic. It proved sufficient to generate exact n-covers of reasonably large covering degree[4] but can certainly be improved using more sophisticated tools, for example using efficient CSP[5] heuristics.

6 Conclusions

In this paper, we proposed a new randomized scalar multiplication algorithm with built-in protection against differential side-channel attacks. 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. Our analysis is presented for elliptic curves but the approach immediately translates to other types of finite groups used in public key cryptography.

It is well known that Coron's scalar blinding procedure may oblige to use a random factor r of size roughly equivalent to that of the group order $\#E$ if one wants $k + r\#E$ to mask all the bits of k. This is in general the case when the prime chosen to define the elliptic curve is very close to a power of 2. In this case, a randomized 256-bit scalar yields

[4] Our largest covering system is an exact 12-cover comprised of more than 3000 congruence classes.
[5] constraint satisfaction problem

a 512-bit scalar multiplication. Regarding differential attacks, the built-in randomization of our algorithm may not necessitate such a large random factor r. The extra cost implied by integer arithmetic could then be largely compensated.

How much information can be deduced from the varying running time of our algorithm remains a question of primary importance currently under investigation. For our experiments, we generated exact n-covers using a greedy strategy. This can certainly be improved. Short-term future work also includes an optimized implementation and the generation of optimal covering systems.

References

1. J. Adikari, V. Dimitrov, and L. Imbert. Hybrid binary-ternary number system for elliptic curve cryptosystems. *IEEE Transactions on Computers*, 60(2):254–265, 2011.
2. J.-C. Bajard, L. Imbert, P.-Y. Liardet, and Y. Teglia. Leak resistant arithmetic. In *Cryptographic Hardware and Embedded Systems, CHES'04*, volume 3156 of *Lecture Notes in Computer Science*, pages 62–75. Springer, 2004.
3. 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*, pages 553–570, 2013.
4. D. J. Bernstein. Curve25519: New Diffie-Hellman speed records. In *Proceedings of Public Key Cryptography, PKC 2006*, volume 3958 of *Lecture Notes in Computer Science*, pages 207–228. Springer, 2006.
5. 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.
6. B. B. Brumley and N. Tuveri. Remote timing attacks are still practical. In *Computer Security, ESORICS 2011*, volume 6879 of *Lecture Notes in Computer Science*, pages 355–371. Springer, 2011.
7. 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, ARITH 2013*, pages 219–228. IEEE Computer Society, 2013.
8. S. Chari, J. R. Rao, and P. Rohatgi. Template attacks. In *Cryptographic Hardware and Embedded Systems, CHES'02*, volume 2523 of *Lecture Notes in Computer Science*, pages 13–28. Springer, 2002.
9. B. Chevalier-Mames, M. Ciet, and M. Joye. Low-cost solutions for preventing simple side-channel analysis: Side-channel atomicity. *IEEE Transactions on Computers*, 53(6):760–768, June 2004.
10. M. Ciet and M. Joye. (virtually) free randomization techniques for elliptic curve cryptography. In *Information and Communications Security, 5th International Conference, ICICS 2003, Proceedings.*, volume 2836 of *Lecture Notes in Computer Science*, pages 348–359. Springer, 2003.
11. 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*, pages 140–155, 2012.
12. J.-S. Coron. Resistance against differential power analysis for elliptic curve cryptosystems. In *Cryptographic Hardware and Embedded Systems, CHES'99*, volume 1717, pages 292–302, 1999.
13. N. M. Ebeid and M. Anwar Hasan. On binary signed digit representations of integers. *Designs, Codes and Cryptography*, 42(1):43–65, 2007.
14. J. Fan, X. Guo, E. De 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*, pages 76–87. IEEE, 2010.
15. J. Fan and I. Verbauwhede. An updated survey on secure ECC implementations: Attacks, countermeasures and cost. In *Cryptography and Security: From Theory to Applications*, volume 6805 of *Lecture Notes in Computer Science*, pages 265–282. Springer, 2012.
16. 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*, volume 2523, pages 551–563, 2002.
17. B. Hough. Solution of the minimum modulus problem for covering systems. arXiv:1307.0874v2 [math.NT], 2014. `http://arxiv.org/abs/1307.0874`.
18. 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*, volume 6737 of *Lecture Notes in Computer Science*, pages 170–187. Springer, 2011.

19. P. C. Kocher. Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In *Advances in Cryptology, CRYPTO'96*, volume 1109 of *Lecture Notes in Computer Science*, pages 104–113. Springer, 1996.
20. P. C. Kocher, J. Jaffe, and B. Jun. Differential power analysis. In *Advances in Cryptology, CRYPTO'99*, volume 1666 of *Lecture Notes in Computer Science*, pages 388–397. Springer, 1999.
21. 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/`.
22. B. Möller. Improved techniques for fast exponentiation. In *Information Security and Cryptology, ICISC 2002*, volume 2587 of *Lecture Notes in Computer Science*, pages 298–312. Springer, 2003.
23. Fouque P.-A, F. Muller, G. Poupard, and F. Valette. Defeating countermeasures based on randomized BSD representations. In *Cryptographic hardware and Embedded Systems, CHES 2004*, number 3156 in Lecture Notes in Computer Science, pages 312–327. Springer, 2004.
24. J. A. Solinas. Efficient arithmetic on Koblitz curves. *Designs, Codes and Cryptography*, 19(2-3):195–249, 2000.
25. C. D. Walter. MIST: An efficient, randomized exponentiation algorithm for resisting power analysis. In B. Preenel, editor, *Topics in Cryptology – CT-RSA 2002*, volume 2271 of *Lecture Notes in Computer Science*, pages 53–66. Springer, 2002.

A Transition matrix and stationary probability

Given any covering system $\mathcal{S}$, the corresponding transition matrix can be computed using Algorithm 3. 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 3 Transition matrix of Markov chain

```
Input: $\mathcal{S}$ a covering system of congruences
Output: The transition matrix $A$ associated to $\mathcal{S}$
 1: $A := (0)$
 2: for $i = 0, \dots, \ell - 1$ do
 3:     for $(r, m) \in \mathcal{S}_{(i)}$ do
 4:         for $j = 0, \dots, \ell - 1$; $j \equiv (i - r)/m \pmod{\ell/m}$ do
 5:             $A_{i,j} := A_{i,j} + 1/(|\mathcal{S}_{(i)}|m)$
 6: return $A$
```

The stationary probability vector of A is the vector π_∞ such that $\pi_\infty A = \pi_\infty$. It is known that $\pi_\infty = \lim_{n\to\infty} \pi A^n$, for any probability distribution π. 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 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[6] 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

$$B = f(0)\,(f(A - Id))^{-1}\,. \tag{7}$$

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

$$\pi_\infty = (0, \dots, 0, 1)\,(f(A - Id))^{-1}\,. \tag{8}$$

[6] Yet rarely explained clearly!