Breaking randomized mixed-radix scalar multiplication algorithms

Jérémie Detrey, Laurent Imbert

Breaking randomized mixed-radix scalar multiplication algorithms

Jérémie Detrey[1] and Laurent Imbert[2]

[1] LORIA, INRIA, CNRS, Université de Lorraine, Nancy, France
jeremie.detrey@loria.fr
[2] LIRMM, CNRS, Université de Montpellier, Montpellier, France
laurent.imbert@lirmm.fr

Abstract. In this paper we present a novel, powerful attack on a recently introduced randomized scalar multiplication algorithm based on covering systems of congruences. Our attack can recover the whole key with very few traces, even when those only provide partial information on the sequence of operations. In an attempt to solve the issues raised by the broken algorithm, we designed a constant-time version with no secret dependent branching nor memory access based on the so-called mixed-radix number system. We eventually present our conclusions regarding the use of mixed-radix representations as a randomization setting.

1 Introduction

After more than twenty years of research on side-channel attacks, it is now very clear that protections of cryptographic implementation are absolutely mandatory. The level and type of protections that need to be implemented depend on the security model. In the software model, one usually simply considers timing and cache attacks. In the more delicate hardware model, where the adversary can monitor power or electromagnetic emanation, insert faults, etc., extra contermeasures are required. This advanced model should also be seriously considered when writing software for a micro-controller, where the user has full access to the device and can therefore be the adversary. In this case, randomization is considered as the number-one countermeasure.

In the context of public-key algorithms, a not so frequent randomization strategy consists in randomizing the exponentiation or scalar multiplication algorithm itself. This can be achieved by taking random decisions in the course of the algorithm. In the elliptic curve setting, Oswald and Aigner proposed the use of randomized addition-subtraction chains [11]. Their solution was broken using the so-called hidden Markov Model (HMM) cryptanalysis by Karlof and Wagner [9]. Another randomization approach of the same kind was proposed by Ha and Moon [8]. Their solution based on Binary Signed Digit (BSD) recodings [4] was broken in [5]. Méloni and Hasan generalized the fractional w-NAF method by allowing random choices for the expansion digits [10].

The first contribution of this paper is a novel attack on a recently proposed approach based on covering systems on congruences (CSC) [7]. The threat model

that we consider simply assumes that the attacker can distinguish point doublings from point additions in an execution trace. Our attack only requires a few traces to recover the whole key.

In a second part, we introduce a new randomized, yet constant-time algorithm which eludes the weaknesses of the original CSC-based approach under the same threat model. Our algorithm uses the so-called mixed-radix number system.

Finally, we sketch out some potential weaknesses of MRS-based randomization strategies in a more advanced threat model. Our conclusion is that mixed-radix based algorithms might remain vulnerable to sophisticated attacks relying on deep-learning and Coppersmith-like techniques.

2 Covering systems of congruences: presentation and weakness

In [7], Guerrini et al. use covering systems of congruences to randomize the scalar multiplication algorithm. A covering system of congruences (CSC) is defined as a set $\mathcal{S} = \{(r_1, m_1), \ldots, (r_n, m_n)\} \subset \mathbb{Z} \times \mathbb{N}$ such that, for any integer x, there exists at least one congruence $(r_i, m_i) \in \mathcal{S}$ such that $x \equiv r_i \pmod{m_i}$. In other words, $\mathcal{S}$ forms a covering of $\mathbb{Z}$, where each pair (r_i, m_i) represents the congruence class $m_i\mathbb{Z} + r_i$.

Then, given a scalar k and a point $P \in E$, the authors randomize the computation of $[k]P$ by picking a random decomposition of k in the covering system $\mathcal{S}$: starting with $k_0 = k$, they recursively compute k_j as $(k_{j-1} - r_{i_j})/m_{i_j}$, for all $j > 0$, where i_j is taken uniformly at random among all indices $i \in \{1, \ldots, n\}$ such that $k_{j-1} \equiv r_i \pmod{m_i}$. The algorithm terminates, as the resulting sequence $(k_j)_{j \geq 0}$ is strictly decreasing (in absolute value) until it eventually reaches $k_\ell = 0$. The sequence of congruences $\mathcal{R} = ((r_{i_1}, m_{i_1}), \ldots, (r_{i_\ell}, m_{i_\ell}))$ is then a random decomposition of the scalar k in the covering system $\mathcal{S}$, and computing $[k]P = [k_0]P$ relies on the fact that, for any $0 < j \leq \ell$, $[k_{j-1}]P$ can be computed recursively from $[k_j]P$ and P as $[k_{j-1}]P = [m_{i_j}]([k_j]P) + [r_{i_j}]P$, starting from $[k_\ell]P = [0]P = \mathcal{O}$.

Although the algorithm does not run in constant-time and uses non-uniform curve operations (i.e. doublings and additions reveal different patterns in an execution trace), the authors of [7] claimed that their randomization is robust against both simple and more advanced attacks.

In the next section we present a novel attack that can recover the secret scalar k by observing only a few execution traces. Our threat model simply assumes that the attacker can distinguish the pattern of a curve doubling from that of an addition. This is a very common threat model that applies to a wide variety of settings, from remote timing attacks to local power or EM observations. Using this information, we are able to deduce critical data about the sequence of congruences $\mathcal{R}$, which render the CSC-based randomization strategy totally useless.

In the sequences of curve operations proposed by the authors in [7, Table 4], one can see that some sequences of operations are specific to only a few congruence classes. Indeed, even if the function $\mu : \mathcal{S} \to \{A, D\}^*$, which maps each congruence class (r_i, m_i) to the corresponding sequence of curve operations is not injective, this is not enough: as soon as μ maps a subset (and only this subset) $\mathcal{S}'$ of $\mathcal{S}$ to some patterns of operations, then it becomes possible to determine whether or not congruences in $\mathcal{S}'$ were used in a given decomposition $\mathcal{R}$ just by checking if these patterns appear in the corresponding trace. Furthermore, if $\mathcal{S}'$ does not form a covering of $\mathbb{Z}$, then this will reveal data on the secret scalar k, by eliminating all integers not covered by $\mathcal{S}'$. For instance, in [7, Table 4], one can see that k_j is odd if and only if the sequence of operations leading to the computation of $[k_j]P$ ends with an addition (A).

3 Full-key recovery algorithm

Based on this observation, we present here an algorithm which recovers the secret scalar k by analyzing traces $\tau \in \{A, D\}^*$ corresponding to randomized scalar multiplications by k, for a given covering system $\mathcal{S}$ and a given function $\mu : \mathcal{S} \to \{A, D\}^*$ such as those described in [7].

To achieve this, the algorithm maintains, for each intercepted trace τ, a set $\mathcal{D}_\tau$ of *partially decoded traces*. Such a partially decoded trace is a triple $(\pi, r, m) \in \{A, D\}^* \times \mathbb{Z} \times \mathbb{N}$, such that π is a prefix of τ, and such that there exists a sequence $((r_{i_1}, m_{i_1}), \ldots, (r_{i_d}, m_{i_d}))$ of congruences of $\mathcal{S}$ which satisfies

$$\begin{cases} \tau = \pi \,\|\, \mu(r_{i_d}, m_{i_d}) \,\|\, \ldots \,\|\, \mu(r_{i_1}, m_{i_1}), \\ r = r_{i_d} m_{i_{d-1}} \ldots m_{i_1} + \cdots + r_{i_2} m_{i_1} + r_{i_1}, \text{ and} \\ m = m_{i_d} \ldots m_{i_1}, \end{cases}$$

where the operator $\|$ denotes the concatenation in $\{A, D\}^*$. In other words, each partially decoded trace in $(\pi, r, m) \in \mathcal{D}_\tau$ corresponds to a congruence class $m\mathbb{Z} + r$ whose elements all admit a decomposition in $\mathcal{S}$ starting with $((r_{i_1}, m_{i_1}), \ldots, (r_{i_d}, m_{i_d}))$, thus accounting for the operations observed in the final part of the trace τ, up to its prefix π which still remains to be decoded.

Therefore, for each trace τ, the algorithm will maintain the set $\mathcal{D}_\tau$ in such a way that there always exists a partially decoded trace $(\pi, r, m) \in \mathcal{D}_\tau$ so that $k \in m\mathbb{Z} + r$ or, equivalently, that $k \in \bigcup_{(\pi,r,m)\in\mathcal{D}_\tau} (m\mathbb{Z} + r)$.

When a trace τ is first acquired, $\mathcal{D}_\tau$ initially only contains the undecoded trace $(\tau, 0, 1)$. One can then repeatedly apply the decoding function δ to the elements of $\mathcal{D}_\tau$, where δ is defined as

$$\delta : (\pi, r, m) \mapsto \{(\pi', r_i m + r, m_i m) \mid (r_i, m_i) \in \mathcal{S} \text{ and } \pi = \pi' \,\|\, \mu(r_i, m_i)\}.$$

Note that fully decoding the trace τ (i.e., applying δ until the prefix π of each decoded trace in $\mathcal{D}_\tau$ is the empty string ϵ) is not directly possible, as the number of such traces grows exponentially with the length of τ. We thus need a way to keep this growth under control. This can be achieved by confronting

intercepted traces together: given two traces τ and τ', if the congruence class $m\mathbb{Z}+r$ of a partially decoded trace $(\pi, r, m) \in \mathcal{D}_\tau$ is disjoint from the union of the congruence classes of the partially decoded traces of $\mathcal{D}_{\tau'}$, then $k \notin m\mathbb{Z}+r$ and (π, r, m) can be safely discarded from $\mathcal{D}_\tau$.

To this effect, the algorithm also maintains a set $\mathcal{H}$ of congruence classes such that for any combination of partially decoded traces

$$((\pi^{(1)}, r^{(1)}, m^{(1)}), \ldots, (\pi^{(t)}, r^{(t)}, m^{(t)})) \in \mathcal{D}_{\tau^{(1)}} \times \cdots \times \mathcal{D}_{\tau^{(t)}},$$

there exists a congruence class $(\hat{r}, \hat{m}) \in \mathcal{H}$ such that

$$\bigcap_{1 \le i \le t} (m^{(i)}\mathbb{Z} + r^{(i)}) \subset \hat{m}\mathbb{Z} + \hat{r}.$$

In the following, let $\mathcal{T} = \{\tau^{(1)}, \ldots, \tau^{(t)}\}$ denote the set of intercepted traces. The complete key recovery algorithm is described in Algorithm 1. Based on the ideas given above, it progressively decodes the intercepted traces, keeping only the partially decoded traces (π, r, m) which are compatible with $\mathcal{H}$, that is, such that $(m\mathbb{Z}+r) \cap \bigcup_{(\hat{r},\hat{m})\in\mathcal{H}} (\hat{m}\mathbb{Z}+\hat{r}) \neq \emptyset$. It then updates $\mathcal{H}$ by computing all pairwise intersections between partially decoded traces and elements of $\mathcal{H}$.

Note that the inclusion tests and intersections of congruence classes required by the algorithm can be computed efficiently. Indeed, for any two congruence classes $m\mathbb{Z}+r$ and $m'\mathbb{Z}+r'$:

- $m\mathbb{Z}+r \subset m'\mathbb{Z}+r'$ if and only if m' divides m and $r \equiv r' \pmod{m'}$;
- $(m\mathbb{Z}+r) \cap (m'\mathbb{Z}+r') \neq \emptyset$ if and only if $r \equiv r' \pmod{\gcd(m, m')}$;
- if $r \equiv r' \pmod{\gcd(m, m')}$, then $(m\mathbb{Z}+r) \cap (m'\mathbb{Z}+r') = \mathrm{lcm}(m, m')\mathbb{Z} + r''$, where r'', computed using the Chinese remainder theorem, is the unique integer in $\{0, \ldots, \mathrm{lcm}(m, m') - 1\}$ such that $r'' \equiv r \pmod{m}$ and $r'' \equiv r' \pmod{m'}$.

4 Implementation and experimental results

This key recovery algorithm was implemented in C. It can be downloaded at `http://imbert.lirmm.net/cover-systems/` together with examples of covering systems of congruences from [7]. On covering systems such as `u3c-48-24` presented in [7, Table 4], it retrieves a 256-bit secret scalar k in a few seconds using between 10 and 15 traces. Using larger covering systems (such as an exact 12-cover consisting of 3315 congruences, kindly provided by the authors), the algorithm takes only slightly longer, but requires the same amount of traces.

We have also tried our key recovery algorithm with an implementation function μ' which leaks slightly less data in the trace about the chosen congruences (r_{i_j}, m_{i_j}) in the random decomposition of the secret scalar k: μ' corresponds to an implementation of the scalar multiplication algorithm which, at each step, in order to compute $Q \leftarrow [m_{i_j}]Q + [r_{i_j}]P$, first computes $[m_{i_j}]Q$ and then adds the precomputed point $[r_{i_j}]P$. This way, in the observed addition/doubling traces,

Algorithm 1 Key recovery algorithm for the CSC-based scalar multiplication [7].

Input: A covering system $\mathcal{S} = \{(r_1, m_1), \ldots, (r_n, m_n)\}$, an implementation $\mu : \mathcal{S} \to \{A, D\}^*$, an oracle Ω generating multiplication traces by an unknown scalar k, and a decoding bound N (typically, $N = 256$).

Output: A small set $\mathcal{K}$ such that $k \in \mathcal{K}$.

function PartialDecode($\mathcal{D}$, $\mathcal{H}$)
  repeat
    $\mathcal{D}' \leftarrow \{(\pi, r, m) \in \mathcal{D} \mid |\pi| > 0 \text{ and } m \leq \max_{(\hat{r},\hat{m})\in\mathcal{H}} \hat{m}\}$
      ▷ *Select only the partially decoded traces we want to decode further.*
    $\mathcal{D} \leftarrow \mathcal{D} \setminus \mathcal{D}'$
    for all $(\pi, r, m) \in \mathcal{D}'$ **do**
      $\mathcal{D} \leftarrow \mathcal{D} \cup \{(\pi', r', m') \in \delta(\pi, r, m) \mid \exists(\hat{r}, \hat{m}) \in \mathcal{H}, (m'\mathbb{Z}+r') \cap (\hat{m}\mathbb{Z}+\hat{r}) \neq \emptyset\}$
  until $|\mathcal{D}| \geq N$ **or** $\mathcal{D}' = \emptyset$
  return $\mathcal{D}$
end function

$\mathcal{H} \leftarrow \{(0, 1)\}$; $\mathcal{T} \leftarrow \{\}$
loop
  $\tau_{\text{new}} \leftarrow \Omega()$; $\mathcal{T} \leftarrow \mathcal{T} \cup \{\tau_{\text{new}}\}$; $\mathcal{D}_{\tau_{\text{new}}} \leftarrow \{(\tau_{\text{new}}, 0, 1)\}$ ▷ *Intercept new trace.*
  repeat
    done $\leftarrow \top$
    for all $\tau \in \mathcal{T}$ **do**
      $\mathcal{D}_\tau \leftarrow \{(\pi, r, m) \in \mathcal{D}_\tau \mid \exists(\hat{r}, \hat{m}) \in \mathcal{H}, (m\mathbb{Z} + r) \cap (\hat{m}\mathbb{Z} + \hat{r}) \neq \emptyset\}$
        ▷ *Remove partially decoded traces which are incompatible with $\mathcal{H}$.*
      if $\mathcal{D}_\tau$ has changed after the previous step **or if** $\mathcal{D}_\tau = \{(\tau, 0, 1)\}$ **then**
        $\mathcal{D}_\tau \leftarrow$ PartialDecode($\mathcal{D}_\tau$, $\mathcal{H}$) ▷ *Further decode trace τ.*
        if $\forall(\pi, r, m) \in \mathcal{D}_\tau, |\pi| = 0$ **then**
          return $\{r \mid (\pi, r, m) \in \mathcal{D}_\tau\}$
        $\mathcal{H} \leftarrow \{(\hat{r}', \hat{m}') \mid \exists(\pi, r, m) \in \mathcal{D}_\tau \text{ and } \exists(\hat{r}, \hat{m}) \in \mathcal{H},$
          $\hat{m}'\mathbb{Z} + \hat{r}' = (m\mathbb{Z} + r) \cap (\hat{m}\mathbb{Z} + \hat{r})\}$
          ▷ *Merge congruence classes from newly decoded traces into $\mathcal{H}$.*
        $\mathcal{H} \leftarrow \{(\hat{r}, \hat{m}) \in \mathcal{H} \mid \nexists(\hat{r}', \hat{m}') \in \mathcal{H}, m \neq m' \text{ and } \hat{m}\mathbb{Z} + \hat{r} \subset \hat{m}'\mathbb{Z} + \hat{r}'\}$
          ▷ *Remove redundant congruence classes from $\mathcal{H}$.*
        if $\mathcal{H}$ has changed after the two previous steps **then**
          done $\leftarrow \bot$
  until done
end loop

$\mu'(r_{i_j}, m_{i_j})$ only reveals the chosen modulus m_{i_j}, but not the residue r_{i_j}. However, for the covering systems mentioned in [7], the knowledge of the m_{i_j}'s only is still enough to retrieve the key. Indeed, according to our experiments, the program will take up to an hour and will require between 20 and 40 traces, but will still completely recover the secret scalar k,

Of course, the less data leaked by the implementation about the chosen congruences (r_{i_j}, m_{i_j}) in the random decomposition of k, the harder it is for our algorithm to recover the key. Ideally, a covering system in which all the moduli

m_i of $\mathcal{S}$ would have the same bit-length, paired with a constant time implementation where the function μ would map all congruences $(r_i, m_i) \in \mathcal{S}$ to the same sequence of operations would render our key recovery algorithm totally useless. The question whether such an algorithm could be efficiently turned into a robust randomized implementation is addressed in the next section.

5 A constant-time alternative

In order to fix the problems raised by our attack, we introduce an alternative scalar multiplication algorithm based on the so-called Mixed-Radix Number System (MRS) – a non-standard positional number system that uses multiple radices – that can be seen as a generalization of Guerrini et al.'s CSC-based approach.

5.1 The mixed-radix number system

In any positional system, numbers are represented using an ordered sequence of numeral symbols, called digits. The contribution of each digit is given by its value scaled by the weight of its position expressed in the base (or radix) of the system. For example, in the conventional, radix-b number system, the contribution of a digit x_i at position i is equal to $x_i b^i$. The total value of the number x represented as $(x_k \dots x_0)_b$ is the sum of the contributions assigned to each digit, i.e., $x = \sum_{i=0}^{k} x_i b^i$.

In the mixed-radix number system (MRS), the weight of each position is no longer expressed as a power of a constant radix. Instead, an MRS base is given by an ordered sequence $\mathcal{B} = (b_1, b_2, \dots, b_n)$ of integer radices all greater than 1. The weight associated to position i is equal to the product of the first $i-1$ elements of that sequence. Hence, the value of the number represented by the sequence of $n+1$ digits $(x_1, x_2, \dots, x_{n+1})$ is

$$x = x_1 + x_2 b_1 + x_3 b_1 b_2 + \cdots + x_{n+1} \prod_{j=1}^{n} b_j = \sum_{i=1}^{n+1} x_i \Pi_{\mathcal{B}}^{(i-1)}, \tag{1}$$

where $\Pi_{\mathcal{B}}^{(k)}$ denotes the product of the first k elements of $\mathcal{B}$. By convention, we set $\Pi_{\mathcal{B}}^{(0)} = 1$ and for any $0 < k \leq n$, $\Pi_{\mathcal{B}}^{(k)} = b_1 b_2 \dots b_k = \prod_{j=1}^{k} b_j$.

A number system is said to be *complete* if every integer can be represented. It is *unambiguous* if each integer admits a unique representation. The usual, radix-b number system with digits in $\{0, \dots, b-1\}$ is complete and unambiguous. In the next Lemma, we recall a known result which shows that these properties may also hold for mixed-radix systems.

Lemma 1. *Let $\mathcal{B} = (b_1, \dots, b_n)$ be a sequence of integer radices all greater than 1. Then, every integer $x \in \mathbb{N}$ can be uniquely written in the form of (1), where the "digits" x_i are integers satisfying $0 \leq x_i < b_i$, for $1 \leq i \leq n$, and $x_{n+1} \geq 0$. We say that the $(n+1)$-tuple $(x_1, \dots, x_{n+1})$ is a mixed-radix representation of x in base $\mathcal{B}$, and we write $x = (x_1, \dots, x_{n+1})_{\mathcal{B}}$.*

Proof. Let $x \in \mathbb{N}$. Starting from $t_1 = x$, one can construct the recursive sequence defined by $t_{i+1} = \lfloor t_i/b_i \rfloor = (t_i - x_i)/b_i$, with $x_i = t_i \bmod b_i$, for all $1 \leq i \leq n$, and then terminate the process by taking $x_{n+1} = t_{n+1}$. Using (1), one can then easily verify that $(x_1, \ldots, x_{n+1})$ is indeed a mixed-radix representation of x in base $\mathcal{B}$.

Suppose that there exist two distinct MRS representations $(x_1, \ldots, x_{n+1})_\mathcal{B}$ and $(x'_1, \ldots, x'_{n+1})_\mathcal{B}$ of x in base $\mathcal{B}$, with digits such that $0 \leq x_i, x'_i < b_i$ for all $1 \leq i \leq n$, and $x_{n+1}, x'_{n+1} \geq 0$. Let i_0 be the smallest integer such that $x_{i_0} \neq x'_{i_0}$. Then, by (1), we have

$$0 = x - x = (x_{i_0} - x'_{i_0})\Pi_\mathcal{B}^{(i_0-1)} + (x_{i_0+1} - x'_{i_0+1})\Pi_\mathcal{B}^{(i_0)} + \cdots + (x_{n+1} - x'_{n+1})\Pi_\mathcal{B}^{(n)}.$$

- If $i_0 = n + 1$, then this gives $(x_{n+1} - x'_{n+1})\Pi_\mathcal{B}^{(n)} = 0$ and, consequently, $x_{n+1} = x'_{n+1}$, which is a contradiction.
- Otherwise, dividing the previous equation by $\Pi_\mathcal{B}^{(i_0-1)}$ and considering it modulo b_{i_0}, we obtain $x_{i_0} - x'_{i_0} \equiv 0 \pmod{b_{i_0}}$. Since $0 \leq x_{i_0}, x'_{i_0} < b_{i_0}$, then $x_{i_0} = x'_{i_0}$, which is also a contradiction.

Therefore, x has a unique MRS representation in base $\mathcal{B}$. □

Note that, following the proof of Lemma 1, if $(x_1, \ldots, x_{n+1})_\mathcal{B}$ is the MRS representation of a given non-negative integer x, the most significant digit x_{n+1} is equal to $\lfloor x/\Pi_\mathcal{B}^{(n)} \rfloor$.

5.2 A deterministic MRS-based scalar multiplication

The algorithm we present here performs a scalar multiplication using an MRS representation of the scalar: given an MRS base $\mathcal{B}$, a scalar k, and a point P on the curve, it first computes the MRS representation of k in base $\mathcal{B}$ then proceeds to computing the actual scalar multiplication in order to obtain $[k]P$.

Recall the algorithm for computing the MRS representation of a given integer k in base $\mathcal{B} = (b_1, \ldots, b_n)$, as sketched in the proof of Lemma 1: starting from $t_1 = k$, one iteratively computes $t_{i+1} = (t_i - k_i)/b_i$, where each digit k_i is given by $t_i \bmod b_i$, for i ranging from 1 to n. Finally, the most-significant digit k_{n+1} is directly given by t_{n+1}. Then, computing $[k]P$ from the MRS representation $(k_1, \ldots, k_{n+1})_\mathcal{B}$ of k is just a matter of taking this algorithm in reverse, remarking that $t_i = b_i t_{i+1} + k_i$ and that, given $T = [t_{i+1}]P$, one can thus compute $[t_i]P$ as $[b_i]T + [k_i]P$.

For the sake of the exposition, Algorithm 2 presented below is purely deterministic. Its randomization and the ensuing issues it raises will be addressed in the next sections.

In this algorithm, special attention should be paid so as not to reveal any information about the digits k_i representing the scalar k, both when computing each of them (line 3) and when evaluating $[b_i]T + [k_i]P$ (line 8). The former will be addressed in Section 5.5. For the latter, we use a regular double-scalar multiplication algorithm such as the one proposed by Bernstein [1]. For the same security reasons, we also use a constant-time Montgomery ladder when computing the scalar multiplication by the most-significant digits k_{n+1} on line 6.

Algorithm 2 Deterministic mixed-radix scalar multiplication algorithm.

Input: An MRS base $\mathcal{B} = (b_1, \ldots, b_n)$, a scalar $k \in \mathbb{N}$, and $P \in E$.
Output: $[k]P \in E$.

1: $t \leftarrow k$
2: **for** $i \leftarrow 1$ **to** n **do** ▷ *Compute the MRS representation of k in base $\mathcal{B}$.*
3: $\quad k_i \leftarrow t \bmod b_i$
4: $\quad t \leftarrow (t - k_i)/b_i$
5: $k_{n+1} \leftarrow t$
6: $T \leftarrow [k_{n+1}]P$ ▷ *Computed using a Montgomery ladder.*
7: **for** $i \leftarrow n$ **downto** 1 **do**
8: $\quad T \leftarrow [b_i]T + [k_i]P$ ▷ *Computed using a regular double-scalar multiplication.*
9: **return** T

5.3 Regular double-scalar multiplication:

Given two points P and Q on an elliptic curve E, and two scalars a and $b \in \mathbb{N}$, the double-scalar multiplication is the operation which computes the point $[a]P + [b]Q$. In [12], Strauss was the first to suggest that this operation can be computed in at most $2\max(\log_2(a), \log_2(b))$ curve operations instead of $2(\log_2(a) + \log_2(b))$ if $[a]P$ and $[b]Q$ are evaluated independently. Unfortunately, Strauss' algorithm cannot be used when side-channel protection is required since the curve operations depend on the values of the scalars a and b.

In 2003, a regular double-scalar multiplication was published in a paper by Ciet and Joye [3]. Their algorithm uses secret-dependent memory accesses and might therefore be vulnerable to cache attacks. More problematically, the proposed algorithm is incorrect.[3]

In a 2006 preprint [1], which was then published in [2], Bernstein proposed a regular two-dimensional ladder, inspired by the Montgomery ladder, and compatible with curves where only differential additions are supported. In order to compute $[a]P + [b]Q$ for any two non-negative scalars a and b, Bernstein's algorithm requires knowledge of the two points P and Q, along with their difference $P - Q$.

Here we present a slight modification of Bernstein's ladder where the input points are P, Q, and $P + Q$. Indeed, as can be seen in Algorithm 2, the proposed MRS scalar multiplication performs a sequence of double-scalar multiplications where the first point of an iteration is the result of the previous one, and where the second point is always the same. In this context, we decided to adapt Bernstein's two-dimensional ladder algorithm so as to return both $R = [a]P + [b]Q$ and $R + Q = [a]P + [b + 1]Q$, at no extra cost, which then allows us to directly call the next double-scalar multiplication with input points R, Q, and $R + Q$.

[3] When $a_i = b_i = 0$, register R_1 is wrongly updated. A possible workaround would be to perform a dummy operation when both a_i and b_i are equal to zero (in [3, Fig. 3], add a fourth register R_4 and replace the instruction $R_b \leftarrow R_b + R_c$ with $R_{4b} \leftarrow R_b + R_c$) but the resulting algorithm would then be subject to fault attacks.

The proposed variant of this two-dimensional ladder is given in Algorithm 3. This algorithm maintains a loop invariant in which, just after iteration i (or, equivalently, just before iteration $i-1$), we have

$$\begin{cases} U = T + [\overline{a_i}]P + [\overline{b_i}]Q, \\ V = T + [a_i]P + [b_i]Q, \text{ and} \\ W = T + [a_i \oplus d_i]P + [b_i \oplus \overline{d_i}]Q, \end{cases}$$

with $T = [\lfloor a/2^i \rfloor]P + [\lfloor b/2^i \rfloor]Q$, and where $\overline{x}$ denotes the negation of bit x (i.e., $\overline{x} = 1 - x$), and $x \oplus y$ the XOR of bits x and y (i.e., $x \oplus y = (x + y) \bmod 2$). The bit sequence $(d_i)_{0 \le i \le n}$, computed on lines 3 to 7, corresponds to the quantity d defined recursively on page 9 of [1], with the initial value $d_0 = a \bmod 2 = a_0$ and

$$d_{i+1} = \begin{cases} 0 & \text{if } (a_i \oplus a_{i+1}, b_i \oplus b_{i+1}) = (0,1), \\ 1 & \text{if } (a_i \oplus a_{i+1}, b_i \oplus b_{i+1}) = (1,0), \\ d_i & \text{if } (a_i \oplus a_{i+1}, b_i \oplus b_{i+1}) = (0,0), \text{ and} \\ 1 - d_i & \text{if } (a_i \oplus a_{i+1}, b_i \oplus b_{i+1}) = (1,1). \end{cases}$$

Algorithm 3 Our variant of Bernstein's two-dimensional ladder [1].

Input: Scalars a and b such that $0 \le a, b < 2^n$, and points P, Q, and $D_+ = P + Q \in E$.
Output: $R = [a]P + [b]Q$ and $R + Q = [a]P + [b+1]Q \in E$.

1: $(a_n \dots a_0)_2 \leftarrow a$ ▷ *$(n+1)$-bit binary expansion of a, with $a_n = 0$.*
2: $(b_n \dots b_0)_2 \leftarrow b$ ▷ *$(n+1)$-bit binary expansion of b, with $b_n = 0$.*
3: $d_0 \leftarrow a_0$
4: **for** $i \leftarrow 0$ **to** $n-1$ **do**
5: $\quad \delta a_i \leftarrow a_i \oplus a_{i+1}$
6: $\quad \delta b_i \leftarrow b_i \oplus b_{i+1}$
7: $\quad d_{i+1} \leftarrow \left(\overline{(\delta a_i \oplus \delta b_i)} \cdot d_i\right) \oplus \delta a_i$ ▷ *Computation of d, as in [1, p.9].*
8: $D_- \leftarrow$ **diffadd**$(P, -Q, D_+)$ ▷ *Compute $D_- = P - Q$.*
9: $U \leftarrow D_+$; $V \leftarrow \mathcal{O}$; $W \leftarrow$ **select**(P, Q, d_n) ▷ *Initialization.*
10: **for** $i \leftarrow n-1$ **downto** 0 **do**
11: $\quad T_0 \leftarrow$ **select**$(W,$ **select**$(U, V, \delta a_i), \delta a_i \oplus \delta b_i)$ ▷ *Select operands.*
12: $\quad T_1 \leftarrow$ **select**$(U, V, d_i \oplus d_{i+1})$
13: $\quad T_2 \leftarrow$ **select**$(D_-, D_+, a_{i+1} \oplus b_{i+1})$
14: $\quad T_3 \leftarrow$ **select**(P, Q, d_i)
15: $\quad U \leftarrow$ **diffadd**(U, V, T_2) ▷ *Update U, V, and W.*
16: $\quad V \leftarrow [2]T_0$
17: $\quad W \leftarrow$ **diffadd**(W, T_1, T_3)
18: **cswap**(U, V, a_0)
19: **cswap**$(V, W, a_0 \oplus b_0)$
20: **return** V, W

Note that, as mentioned before, this algorithm supports curves with formulae for differential addition. This operation is denoted by **diffadd** in Algorithm 3: for any two points S and $T \in E$, **diffadd**$(S, T, S - T)$ computes the point

$S + T$. However, if a regular addition formulae is available on E, **diffadd** can be replaced by a normal point addition, and its third argument is not used. In that case, the computation of D_- on line 8, along with those of T_2 and T_3 on lines 13 and 14, respectively, can be ignored.

Note also that, on the very first iteration of the ladder, when $i = n - 1$, the first differential addition on line 15 is trivial, since $V = \mathcal{O}$. It can then safely be ignored as well. Consequently, putting it all together, this algorithm performs $2n$ differential additions (or $2n - 1$ regular additions, if differential additions are not available) and n doublings on E.

Finally, note that Algorithm 3 contains no secret-dependent conditional branching instruction or memory access, so as to avoid branch-prediction and cache attacks. For this purpose, the only secret-dependent conditional operations in the algorithm are selections and conditional swaps, denoted by **select** and **cswap**, respectively:

- The former is equivalent to a conditional operator: it returns its first or its second argument depending on whether its third one, a single bit, is 1 or 0, respectively. It can be implemented without conditional branching using arithmetic operations or bit-masking techniques. For instance, the operation $R \leftarrow \mathbf{select}(X, Y, c)$ can be implemented as

$$R \leftarrow Y \oplus ((X \oplus Y) \times c),$$

  where $\oplus$ denotes the bitwise XOR.
- Similarly, the operation $\mathbf{cswap}(X, Y, c)$, which swaps the values of X and Y if and only if c is 1, can be implemented as

$$\begin{aligned} M \quad &\leftarrow (X \oplus Y) \times c \\ (X, Y) &\leftarrow (X \oplus M, Y \oplus M). \end{aligned}$$

5.4 Randomizing the MRS-based scalar multiplication

The MRS digits k_i and radices b_i processed by the deterministic MRS scalar multiplication introduced in Algorithm 2 exclusively depend on the MRS base $\mathcal{B}$ used to represent k. The basic idea of our randomized version is quite simple. We simply represent the secret scalar k using an MRS base $\mathcal{B}$ chosen uniformly at random prior to each scalar multiplication. This way, each call to the algorithm manipulates different values k_i and b_i, even if the same scalar k is used as input, hence producing different execution traces. This new algorithm can be somehow seen as a refinement and an improvement of the CSC-based algorithm proposed in [7].

As will be seen in the next section, some caution is required. In particular, the base $\mathcal{B}$ is randomly chosen among a predefined family $\mathcal{F}$ of fine-tuned MRS basis. And each base $\mathcal{B}$ from $\mathcal{F}$ should be able to represent scalars in a predetermined interval whose bounds are denoted $X_{\mathcal{F}}^{\min}$ and $X_{\mathcal{F}}^{\max}$. For completeness, we give the randomized version in Algorithm 4.

Algorithm 4 Randomized mixed-radix scalar multiplication algorithm.

Input: A family $\mathcal{F}$ of MRS bases, a scalar $k \in \{X_{\mathcal{F}}^{\min}, \ldots, X_{\mathcal{F}}^{\max} - 1\}$, and $P \in E$.
Output: $[k]P \in E$.

1: $\mathcal{B} = (b_1, \ldots, b_n) \xleftarrow{\$} \mathcal{F}$ ▷ *Choose an MRS base in $\mathcal{F}$ uniformly at random.*
2: $t \leftarrow k$
3: **for** $i \leftarrow 1$ **to** n **do** ▷ *Compute the MRS representation of k in base $\mathcal{B}$.*
4: $\quad k_i \leftarrow t \bmod b_i$
5: $\quad t \leftarrow (t - k_i)/b_i$
6: $k_{n+1} \leftarrow t$
7: $T \leftarrow [k_{n+1}]P$ ▷ *Computed using a Montgomery ladder.*
8: **for** $i \leftarrow n$ **downto** 1 **do**
9: $\quad T \leftarrow [b_i]T + [k_i]P$ ▷ *Computed using Bernstein's regular double ladder.*
10: **return** T

Even though the idea behind this randomization is quite simple, it raises several additional issues, mostly because the randomized algorithm should not reveal the randomly chosen MRS base $\mathcal{B}$. Namely, we need to find suitable families $\mathcal{F}$ of MRS bases such that the following requirements are fulfilled:

1. the range $\{X_{\mathcal{F}}^{\min}, \ldots, X_{\mathcal{F}}^{\max} - 1\}$ can accommodate any scalar k of relevance for the cryptosystem at hand;
2. $\mathcal{F}$ is large enough to ensure a sufficient amount of randomization;
3. one can securely pick $\mathcal{B} \in \mathcal{F}$ at random;
4. one can securely compute the MRS representation of k in base $\mathcal{B}$;
5. one can securely compute the scalar multiplication $[k]P$.

As already stated, is it essential that any implementation of Algorithm 4 runs in constant-time. This implies in particular that the conversion of k into its MRS form and the subsequent MRS-based scalar multiplication both run in constant-time.

5.5 Secure mixed-radix decomposition

The **for** loop on lines 3 to 5 of Algorithm 4 computes the representation of the secret scalar k in the randomly selecter MRS basis $\mathcal{B}$ by iteratively computing residues and divisions by the radices b_i. Obviously, special care has to be paid here to perform this scalar decomposition so as not to reveal any information on k by side channels, but we should also ensure that no data is leaked about the radices b_i as well.

As will be seen in Section 5.6, our family of MRS bases consists of n-tuples of radices taken from a predefined set of size m. The method presented in Algorithm 5 computes each digit k_i first by computing the quotient q and the remainder r of the Euclidean division of t (the current scalar to decompose) by the corresponding radix b_i (lines 6 to 7).

Since all radices of $\mathcal{B}$ are known in advance, we can precompute the inverse b_i^{-1} of each radix b_i so that the quotient on line 6 can be computed using a single

multiplication, as proposed by Granlund and Montgomery in [6]. Note that the inverses have to be precomputed with enough precision in order to obtain the exact quotient: according to [6, Theorem 4.2], if the scalar k fits into N bits, then the inverses require $N+1$ bits of precision. Therefore, the product on line 6 is an $N \times (N+1)$-bit multiplication. On the other hand, assuming the radices of $\mathcal{B}$ fit on a single w-bit machine word, then the remainder r can be computed using only w-bit arithmetic on line 7.

Finally, in order to avoid secret-dependent memory accesses, the value of the current radix b_i and of its precomputed inverse $(b_i)^{-1}$ are loaded into variables b and b_{inv}, respectively, by the **for** loop on lines 3 to 5: this loop goes through all the radices b_j of $\mathcal{B}$, and conditionally moves the one for which $j = \sigma(i)$ into b (and similarly for b_{inv}), where σ is a selection fonction from $\{1, \ldots, n\}$ to $\{1, \ldots, m\}$. These conditional moves are performed by the **cmove** instruction: **cmove**(X, Y, c) always loads the value of Y from memory (ensuring a secret-independent memory access pattern), but only sets X to this value if the bit c is 1. Quite similarly to **cswap**, **cmove** also avoids any kind of conditional branch so as to resist branch-prediction attacks as well.

Algorithm 5 Secure mixed-radix decomposition.

Input: An MRS bases $\mathcal{B} = (b_1, \ldots, b_n)$, a scalar $k \in \{X_{\mathcal{B}}^{\min}, \ldots, X_{\mathcal{B}}^{\max} - 1\}$.
Output: $(k_1, \ldots, k_n)_{\mathcal{B}}$, the MRS representation of k in base $\mathcal{B}$.

1: $t \leftarrow k$
2: **for** $i \leftarrow 1$ **to** n **do**
3: **for** $j \leftarrow 1$ **to** m **do**
4: **cmove**(b, b_j, $j = \sigma(i)$) ▷ *Load b_j into b if and only if $j = \sigma(i)$.*
5: **cmove**(b_{inv}, b_j^{-1}, $j = \sigma(i)$) ▷ *Load the corresponding precomputed inverse.*
6: $q \leftarrow \lfloor t \cdot b_{\text{inv}} \rfloor$ ▷ *Division by b using the precomputed inverse.*
7: $r \leftarrow (t - q \cdot b) \bmod 2^w$ ▷ *The remainder is computed on a single machine word.*
8: $k_i \leftarrow r$; $t \leftarrow q$
9: **return** $(k_1, \ldots, k_n)$

Note that precomputing and storing in memory the inverse of each radix of $\mathcal{B}$ with $N+1$ bits of precision might prove too expensive on some embedded systems. An alternative here is to precompute the least-common multiplier of the radices of $\mathcal{B}$, $m = \text{lcm}(b_1, \ldots, b_n)$, along with its inverse $m_{\text{inv}} = m^{-1}$ (with $N+1$ bits of precision as well), and to always compute the quotient of t divided by m, as $\lfloor t \cdot m_{\text{inv}} \rfloor$, instead of $\lfloor t \cdot b_{\text{inv}} \rfloor$. Then, as long as the LCM m itself fits on a single machine word, the actual quotient and remainder of t divided by b can be retrieved using simple word-level arithmetic, requiring only word-precision precomputed inverses of the b_i's.

5.6 Randomized yet constant-time scalar multiplication

The regular double-scalar multiplication algorithm presented in Section 5.3 already embeds some necessary properties for thwarting side-channel attacks. Indeed, its regular structure along with its absence of secret-dependent branching and memory access patterns should prevent the scalar multiplication from revealing information about the secret digits k_i and MRS radices b_i. However, the running time of Algorithm 3 remains proportional to the bit-length of the input scalars. Therefore, without further precaution in choosing the random base $\mathcal{B}$, an execution trace of Algorithm 4 will reveal the bit-length of the successive radices b_i used in the decomposition. This would then give an attacker a serious advantage for recovering the random base $\mathcal{B}$.

In order to prevent such kind of timing attack, we impose that all the radices of $\mathcal{B}$ have the same bit-length s. For all $1 \leq i \leq n$ we thus have $2^{s-1} \leq b_i < 2^s$. Therefore, each double-scalar multiplication computed in line 8 do generate strictly identical patterns in terms of issued instructions, branches, and memory accesses, thus reducing the number of side channels available to an attacker.

As an example of adequate family of bases $\mathcal{F}$, we define $\mathcal{F}_{s,n,m}$ as the set of all n-tuples exclusively composed of s-bit radices taken from a predefined set of size m, so that $|\mathcal{F}_{s,n,m}| = m^n$. Hence, choosing a base $\mathcal{B}$ uniformly at random in $\mathcal{F} = \mathcal{F}_{s,n,m}$ as in line 1 of Algorithm 4 provides $\rho = \lfloor \log_2(m^n) \rfloor$ bits of randomization. Observe that among all possible sets of size m, those composed of the m largest s-bit integers provide shorter MRS representations. We thus define:

$$\mathcal{F}_{s,n,m} = \{(b_1, \ldots, b_n) \ : \ 2^s - m \leq b_i \leq 2^s - 1 \text{ for } 1 \leq i \leq n\}.$$

Note that the elements of any given base $\mathcal{B} \in \mathcal{F}_{s,n,m}$ need not be distinct.

Yet, choosing s-bit radices is not sufficient to guarantee constant-time. We shall further ensure that the MRS representation of k fits on exactly $n+1$ digits, no matter its actual value and the chosen base $\mathcal{B}$. As a consequence, the whole loop of Algorithm 4 (line 8) will repeat exactly n times. We achieve this property by adding to k a suitable multiple of the group order ℓ. More precisely, we compute $\hat{k} = k + \alpha\ell$ for a well-suited value α that guarantees that any such $X^{\min}_{\mathcal{F}_{s,n,m}} \leq \hat{k} < X^{\max}_{\mathcal{F}_{s,n,m}}$ can be written in any base $\mathcal{B} \in \mathcal{F}_{s,n,m}$ using exactly $n+1$ digits (i.e. with $x_{n+1} > 0$.)

If a base $\mathcal{B}$ is poorly chosen, it is possible that for some values of k, such an α does not exist, meaning that the base $\mathcal{B}$ cannot accommodate all possible values of the scalar. Fortunately, it is not difficult (although rather technical) to check whether a base $\mathcal{B}$ is legitimate and to adjust it accordingly if it is not the case.

At this point, we intentionally skip most of the details regarding the parameter selection, as well as the level of randomization that can be achieved and the efficiency of the algorithm for reasons that will become clear in the next section.

6 Discussions

Let us summarize what we have so far. We designed a randomized algorithm which runs in constant-time, contains no secret-dependent branching and memory access, and produces a very regular pattern of elementary operations (additions and doublings). The size and number of radices in the MRS bases can be easily determined so as to guarantee a prescribed level of randomization. It therefore presents many of the required characteristics of a robust randomized algorithm. In particular, our attack on the CSC-based approach presented at the beginning of the paper is totally ineffective.

Could it then be considered as a secure alternative to the traditional randomization strategies in a more advanced threat model? Unfortunately, the answer is probably no for reasons that we explain below.

Recall the MRS representation of the secret scalar k:

$$k = k_1 + k_2 b_1 + k_3 b_1 b_2 + \cdots + k_{n+1} \prod_{j=1}^{n} b_j = \sum_{i=1}^{n+1} k_i \Pi_{\mathcal{B}}^{(i-1)}.$$

Observe that the least significant digit k_1 depends solely on b_1 as $k_1 = k \bmod b_1$. Because b_1 is taken at random in a fixed set of size m, this represents $\log_2(m)$ bits of randomization. On the other hand, k_1 contains between $s-1$ and s bits of information (since $b_1 \geq 2^{s-1}$). Hence, if $m < 2^{s-1}$ the least significant digit is only partially masked and we could possibly recover $s - 1 - \log_2(m)$ bits of information. Depending on the parameter choice, this may be easily taken into account, but the situation is more problematic in the advanced hardware model, where the attacker has access to the device.

Let us assume that the attacker knows the public key, i.e. the point $R = [k]P$. She can then precompute the points $R_{b,i} = [b^{-1} \bmod \ell](R - [i]P)$ for all b in $[2^s - m, 2^s - 1]$ and for all $i = 0, \ldots, b-1$. For all b and i, the following invariant holds: $[b]R_{b,i} + [i]P = R$.

The attacker can now reprogram the hardware so that it evaluates $[b]R_{b,i} + [i]P$ for all b and i. And she stores the $m \times 2^s$ corresponding execution traces. Although the sequences of operations are identical, the bit-flips will differ depending on b and i. And it is not inconceivable at all that recent advances in deep-learning techniques for side-channel attacks could be used to differentiate these traces.

With this precomputed data at hand, the attacker may now ask the device to compute $[k]P$ many times. The last iteration of the algorithm will always go through one of the temporary points $R_{b,i}$ with $i = k \bmod b$ and will eventually evaluate $[b]R_{b,i} + [i]P$ in order to get the correct result $R = [k]P$. If the attacker runs the algorithm sufficiently many times, she should be able to distinguish m different traces which correspond to each possible b-value. By pairing these with her set of precomputed traces, she could then recover $k \bmod b$ for all $b \in [2^s - m, 2^s - 1]$. And thus, thanks to the CRT, the value $k \bmod \mathrm{lcm}(2^s - m, \ldots, 2^s - 1)$. As an example, for $s = 8$ bits, $m = 16$, this

attack would require the precomputations of less than 16×2^8 traces[4]. The value of $\mathrm{lcm}(240, 241, \ldots, 255)$ is a 93-bit integer. The attacker would then recover 93 bits of information of k. She may even be able to recover the whole secret with Coppersmith-like techniques and brute force.

7 Conclusion

We presented a very powerful attack on a recently proposed randomized algorithm based on covering systems of congruences. This algorithm uses a representation of the secret scalar which resembles and shares many similarities with the mixed-radix number system. In an attempt to design a more robust algorithm that would thwart our attack, we were able to build a randomized algorithm that runs in constant-time and is free of secret-dependent branching and memory-access. However, the intrinsic nature of the mixed-radix number system, namely its positional property, combined with randomization, may allow a virtual powerful attacker to recover much more information than what was first expected. Therefore, we do not recommend the use of mixed-radix representations for randomization.

Acknowledgments

The authours would like to thank the anonymous referees for their careful reading and constructive comments, as well as Victor Lomne and Thomas Roche (`https://ninjalab.io/team`) for their support and invaluable suggestions.

References

1. Bernstein, D.J.: Differential addition chains. `https://cr.yp.to/ecdh/diffchain-20060219.pdf` (2006)
2. Bernstein, D.J., Lange, T.: Topics in Computational Number Theory Inspired by Peter L. Montgomery, chap. Montgomery curves and the Montgomery ladder, pp. 82–115. Cambridge University Press (2017), `https://eprint.iacr.org/2017/293`
3. 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)
4. Ebeid, N., Hasan, M.A.: On binary signed digit representations of integers. Designs, Codes and Cryptography **42**(1), 43–65 (2007)
5. Fouque, P.A., Muller, F., Poupard, G., Valette, F.: Defeating countermeasures based on randomized BSD representations. In: Cryptographic hardware and Embedded Systems, CHES 2004. pp. 312–327. No. 3156 in Lecture Notes in Computer Science, Springer (2004)

[4] $240 + 241 + \cdots + 255 = 3960$ exactly.

6. Granlund, T., Montgomery, P.L.: Division by invariant integers using multiplication. In: Proceedings of the ACM SIGPLAN 1994 conference on Programming language design and implementation (PLDI '94). ACM SIGPLAN Notices, vol. 29, pp. 61–72. ACM (1994)
7. Guerrini, E., Imbert, L., Winterhalter, T.: Randomized mixed-radix scalar multiplication. IEEE Transactions on Computers **67**(3), 418–431 (2017). https://doi.org/10.1109/TC.2017.2750677
8. 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)
9. Karlof, C., Wagner, D.: Hidden markov model cryptanalysis. In: Cryptographic Hardware and Embedded Systems, CHES 2003. pp. 17–34. No. 2779 in Lecture Notes in Computer Science, Springer (2003)
10. Méloni, N., Hasan, M.A.: Random digit representation of integers. In: Proceedings of the 23rd IEEE Symposium on Computer Arithmetic, ARITH23. pp. 118–125. IEEE Computer Society (2016)
11. Oswald, E., Aigner, M.: Randomized addition-subtraction chains as a countermeasure against power attacks. In: Cryprographic Hardware and Embedded Systems, CHES 2001. pp. 39–50. No. 2162 in Lecture Notes in Computer Science, Springer (2001)
12. Strauss, E.G.: Addition chains of vectors (problem 5125). American Mathematical Monthly **70**, 806–808 (1964)