Lazy Leak Resistant Exponentiation in RNS

Andrea Lesavourey, Christophe Negre, Thomas Plantard

Lazy Leak Resistant Exponentiation in RNS

A. Lesavourey[1,2], C. Negre[1,2] and T. Plantard[3]

[1] Team DALI, Université de Perpignan, France
[2] LIRMM, UMR 5506, Université Montpellier 2 and CNRS, France
[3] CCISR, SCIT, (University of Wollongong), Australia

Abstract. In [1] the authors introduced the leak resistant arithmetic in RNS to randomize RSA modular exponentiation. This randomization is meant to protect implementations on embedded device from side channel analysis. We propose in this paper a lazy version of the approach of [1] in the case of right-to-left square-and-multiply exponentiation. We show that this saves roughly 30% of the computation when the randomization is done at each loop iteration. We also show that the level of randomization of the proposed approach is better than the one of [1] after a few number of loop iterations.

Keywords. RSA, modular exponentiation, randomization, side channel analysis, RNS.

1 Introduction

RSA cryptosystem is nowadays the most used cryptosystem. The basic operation in RSA encryption or signature is the modular exponentiation. Specifically, given an RSA modulus N, a message X and an exponent E, we have to compute $X^E \mod N$. In practice N, X and E are integers of bit size $2048 - 4096$. The exponentiation can be computed with a few thousands modular multiplications and squarings with the square-and-multiply algorithm.

When such a computation is done on an embedded device it is under the threat of side channel analysis. Such attacks monitor power consumption, electromagnetic emanation or computation time and then try to extract the secret key from leaked information in such data. The simplest of this attack is the simple power analysis (SPA) which threatens implementations based on square-and-multiply algorithm and such that squaring and multiplication have different power traces. Then an attacker can decompose the power trace into the sequence of traces corresponding to squarings and multiplications. Then he can deduce the sequence of bit of the exponent since a multiplication is computed only when the bit exponent $e_i = 1$. SPA can be easily defeated by using a regular exponentiation algorithm like the square-and-multiply-always proposed by Coron [3] and the Montgomery-ladder [4].

Differential and correlation power analyses (DPA and CPA) are more advanced attacks: they can threaten implementations protected with a regular exponentiation algorithm. The idea is to guess the bits of the exponent sequentially and for each guess we compute the data in the next loop iteration of an exponentiation. The correct guess leads to data which are correlated to the power trace. If we have a large number of power traces, a statistical analysis accurately determines this correlation and we can proceed to the guess of the next bit.

Such attacks are generally defeated by randomizing data and exponentiation. Coron suggests in [3] to randomize the exponent by adding a random multiple of $\phi(N) = (q-1)(p-1)$ or to randomize the message X by multiplying it by a

random multiplicative mask. In [1], the authors proposed a randomization based on modular arithmetic in residue number system (RNS). RNS is based on the Chinese remainder theorem: given t pairwise coprime moduli m_i, an integer X is represented by its residue modulo m_i. Modular arithmetic is implemented in this system by a modified version of the modular multiplication of Montgomery [9]. The basic idea of [1] is to randomize the set of moduli m_i: this leads to a randomization of the computations and also of the multiplicative factor induced by Montgomery modular multiplication. The authors in [1] proposed to use this randomization all along the exponentiation to get a stronger counter-measure against DPA and CPA.

In this paper we propose an modified version of the counter-measure of [1]. Specifically, we consider the right-to-left version of the square-and-multiply-always exponentiation. We remove the Update operation which was initially used in [1] to keep the data in the correct Montgomery representation. This reduces the complexity of the randomization. Also, this modifies the form the multiplicative mask of the data involved in the exponentiation algorithm. We study this mask and give a strategy to remove it at the end of the exponentiation. We also evaluate the level of randomization induced by this modified version of the modular exponentiation with leak resistant arithmetic.

This paper is organized as follows: In Section 2 we briefly review modular exponentiation and side channel analysis. In Section 3 we review the residue number system and the randomization based on leak resistant arithmetic. In Section 4 we provide our modified exponentiation algorithm with strategy render the last mask equal to 1. In Section 5 we evaluate the level of randomization and compare it to[1]. Finally, in Section 6 we give some concluding remarks.

2 Modular exponentiation and side channel analysis

The main operation in RSA protocols is the modular exponentiation: given a modulus N, an exponent E and $X \in \{0, \ldots, N-1\}$ we have to compute $R_0 = X^E \mod N$. This exponentiation can be computed with a sequence of squarings and multiplications using the so-called square-and-multiply approach. This approach reconstructs the exponent bit after bit either from left-to-right or from right-to-left. The right-to-left version computes sequentially $X^2, X^{2^2}, \ldots, X^{2^i}, \ldots$, temporary stored in the variable Z and if the bit $e_i = 1$ in E a multiplication $R_0 \leftarrow R_0 \times Z$ is performed to set this bit to 1 in the exponent. This approach is shown in Algorithm 1.

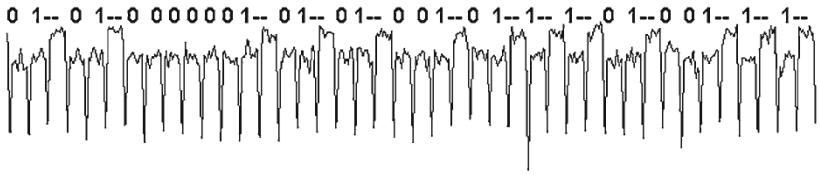
Algorithm 1 Right-to-left Square-and-multiply

Require: An RSA modulus N, an integer $X \in \{0, \ldots, N-1\}$ and an exponent $E = (e_{\ell-1}, \ldots, e_0)_2$
Ensure: $R_0 = X^E \mod N$
1: $R_0 \leftarrow 1,\ Z \leftarrow X$
2: **for** i **from** 0 **to** $\ell - 1$ **do**
3: **if** $e_i = 1$ **then**
4: $R_0 \leftarrow R_0 \times X \mod N$
5: $Z \leftarrow Z^2 \mod N$
6: **return** R_0

When such algorithms are implemented on an embedded device they can be threatened by side channel analysis. Such attacks monitor either the computation time [6], power consumption [5] or electromagnetic emanation [8] to derive the secret exponent E. The main attacks are the following:

– **Simple power analysis (SPA).** This attack is based on the different shapes of the power trace of a squaring and a multiplication. The power trace of an exponentiation can be decomposed as sub-traces corresponding to the sequence of squarings and multiplications computed during an exponentiation. Then the bits of the exponent can be determined as follows: if a squaring is followed by a multiplication the corresponding bit is "1" and if a squaring is followed by a squaring then the corresponding bit is "0". Figure 1 is an example provided by Kocher *et al.* in [7] of an SPA on a modular exponentiation.

Fig. 1. SPA attack from [7]



To counteract SPA the basic approach is to break the correlation between the sequence of operations done in an exponentiation and the secret exponent. A first solution is the approach proposed by Coron in [3] which consists to add a dummy multiplication in the right-to-left exponentiation when the exponent bit is $e_i = 0$. Indeed, the sequence operations becomes a regular sequence of ℓ squarings always followed by a multiplication. This approach is called the square-and-multiply-always exponentiation. We provide in Algorithm 2 the right-to-left version of this method. Another popular regular exponentiation is the Montgomery-ladder which can be found in [4].

Algorithm 2 Right-to-left Square-and-multiply-always

Require: A modulus N, an integer $X \in \{0, \ldots, N-1\}$ and $E = (e_{\ell-1}, \ldots, e_0)_2$
Ensure: $R_1 = X^E \mod N$
1: $R_0 \leftarrow 1,\ R_1 \leftarrow 1,\ Z \leftarrow X$
2: **for** i **from** 0 **to** $\ell - 1$ **do**
3: **if** $e_i = 0$ **then**
4: $R_0 \leftarrow R_0 \times Z \mod N$
5: **else**
6: $R_1 \leftarrow R_1 \times Z \mod N$
7: $Z \leftarrow Z^2 \mod N$
8: **return** R_1

– **Correlation and differential power analyses.** These attacks are meant to recover the exponent E in the modular exponentiation even if it is protected against SPA with the use of the square-and-multiply-always scheme. The goal is to track the different values $R_0^{(0)}, R_0^{(1)}, \ldots, R_0^{(\ell)}$ taken by R_0 during the exponen-

tiation. We assume that we know $R_0^{(j)}$ for $j = 0, \ldots, i$ and e_j for $j = 0, \ldots, i-1$:

$$\underbrace{R_0^{(0)} \longrightarrow^{e_0} R_0^{(1)} \rightarrow^{e_1} R_0^{(2)} \rightarrow^{e_2} \cdots \rightarrow^{e_{i-1}} R_0^{(i)}}_{\text{known bits and data}} \overbrace{\begin{array}{l} \nearrow R_0^{(i)} \times X^{2^{i+1}} \text{ if } e_i = 1 \\ \searrow R_0^{(i)} \text{ if } e_i = 0 \end{array}}^{\text{guessing } e_i \text{ gives } R_0^{(i+1)}}$$

By guessing the bit e_i we can compute the next value $R_0^{(i+1)}$. The correct guess for e_{i+1} is the one such that to the power trace is correlated to $R_0^{(i+1)}$. In practice it is necessary to have multiple power traces in order to have good statistical evaluation of this correlation. This statistical evaluation of the correlation is done in [5] by computing a differential of the power traces and in [2] by computing the co-variance between with the Hamming weight of $R_0^{(i+1)}$ and the power traces.

Counter-measures. The main strategies were proposed by Coron in [3] and consist in randomizing the data and the computations at different level. First Coron proposes to randomize the exponent E by adding a random multiple of $\phi(N) = (p-1)(q-1)$ where p and q are the two factors of N:

$$E' = E + \beta(p-1)(q-1)$$

and E' satisfies $X^{E'} \mod N = X^E \mod N$. Coron also proposes to randomize the data by randomly blinding the message X: the message is multiplied it by a random value α

$$X' = X \times \alpha \mod N.$$

Then $X^E \mod N$ can be recovered from $X'^E \mod N = X^E \times \alpha^E \mod N$ if the value $\alpha^E \mod N$ is precomputed. The message blinding can also be achieved by using a randomized representation of X. Indeed in [1] the authors suggest to use the residue number system (RNS) for the representation of integers modulo N. They could then randomize the representation by randomly permuting the moduli of the RNS basis.

3 Leak resistant modular exponentiation in RNS

We review in this section the Montgomery multiplication in residue number system. Then we will briefly present the approach of Bajard *et al.* [1] for the randomization of the modular exponentiation.

3.1 Modular multiplication in RNS

Montgomery modular multiplication. Let N be an RSA modulus and let X and Y be two integers such that $0 \leq X, Y < N$. Montgomery proposed in [9] a method for modular multiplication which avoids Euclidean division. This approach uses an integer A such that $A > N$ and $\gcd(A, N) = 1$ and computes $Z = XYA^{-1} \mod N$ as follows:

$$\begin{array}{l} Q \leftarrow -XYN^{-1} \mod A \\ Z \leftarrow (XY + QN)/A \end{array} \tag{1}$$

In the above computation we have $(XY + QN) \mod A = 0$, this means that the division by A is exact in the computation of Z and then $Z \equiv XYA^{-1} \mod N$. The

integer Z is almost reduced modulo N since $Z = (XY + QN)/A < (N^2 + AN)/A < 2N$: if $Z > N$ we subtract N to get $Z < N$.

For a long sequence of multiplications, we can use the following Montgomery representation

$$\widetilde{X} = (X \times A) \mod N. \tag{2}$$

in order to absorb the factor A^{-1} appearing when we perform a Montgomery multiplication. Indeed, the Montgomery multiplication applied to $\widetilde{X}$ and $\widetilde{Y}$ outputs $\widetilde{Z} = XYA \mod N$, i.e., the Montgomery representation of the product of $Z = X \times Y \mod N$. In the sequel the factor A in (2) will be called the *Montgomery factor*.

Residue number system. Now we review the residue number system (RNS). Let $a_1, \ldots, a_t$ be t coprime integers. In the RNS base $\mathcal{A} = \{a_1, \ldots, a_t\}$ an integer X such that $0 \leq X < A = \prod_{i=1} a_i$ is represented by the t residues

$$x_i = X \mod a_i \text{ for } i = 1, \ldots, t.$$

X can be reconstructed from the t residues x_i as follows

$$X = \left(\sum_{i=1}^{t} \left[x_i \times A_i^{-1} \right]_{a_i} \times A_i \right) \mod A \tag{3}$$

where $A_i = \prod_{j=1, j \neq i}^{t} a_i$ and the brackets $[\,\cdot\,]_{a_i}$ denotes a reduction modulo a_i.

Let $X = (x_1, \ldots, x_t)_{\mathcal{A}}$ and $Y = (y_1, \ldots, y_t)_{\mathcal{A}}$ be two integers given in an RNS base $\mathcal{A}$. Then, the Chinese remainder theorem provides that an integer addition $X + Y$ or multiplication $X \times Y$ consists in RNS in t independent additions $x_i + y_i \mod a_i$ and multiplications $x_i \times y_i \mod a_i$ for $i = 1, \ldots, t$. These t operations modulo a_i can be implemented in parallel since each operation modulo a_i are independent from the others.

Montgomery multiplication in RNS. In [10] Posch and Posch proposed to perform a Montgomery multiplication in RNS. To reach their goal they first modified the second step of the Montgomery multiplication (1) as follows:

$$Z \leftarrow (XY + QN)A^{-1} \mod B$$

where B is an integer such that

$$\gcd(A, B) = \gcd(N, B) = 1 \text{ and } B \geq 2N. \tag{4}$$

Then they used two RNS bases $\mathcal{A} = \{a_1, \ldots, a_t\}$ and $\mathcal{B} = \{b_1, \ldots, b_t\}$ along with their products $A = \prod_{i=1}^{t} a_i$ and $B = \prod_{i=1}^{t} b_i$ which satisfy (4). Then the first step of a Montgomery multiplication becomes $Q \leftarrow -X \times Y \times N^{-1} \mod A$ in $\mathcal{A}$ and the second step becomes $Z \leftarrow (XY + QN)A^{-1} \mod B$ in the base $\mathcal{B}$. To have a fully functional algorithm Posch and Posch needed some conversions, called base extensions (BE), to convert Q from $\mathcal{A}$ to $\mathcal{B}$ and to convert Z from $\mathcal{B}$ to $\mathcal{A}$. Their Montgomery multiplication in RNS is shown in Algorithm 3.

3.2 Leak resistant arithmetic

In [1] the authors noticed that we can take advantage of the Montgomery multiplication in RNS to randomize a modular exponentiation. Specifically, when we

Algorithm 3 MM_RNS$(X, Y, \mathcal{A}, \mathcal{B})$

Require: X, Y in $\mathcal{A} \cup \mathcal{B}$
Ensure: $XYA^{-1} \mod N$ in $\mathcal{A} \cup \mathcal{B}$
1: $[Q]_{\mathcal{A}} \leftarrow [-XYN^{-1}]_{\mathcal{A}}$
2: $BE_{\mathcal{A}\to\mathcal{B}}([Q]_{\mathcal{A}})$
3: $[Z]_{\mathcal{B}} \leftarrow [(XY + QN)A^{-1}]_{\mathcal{B}}$
4: $BE_{\mathcal{B}\to\mathcal{A}}([Z]_{\mathcal{B}})$
5: **return** $(Z_{\mathcal{A}\cup\mathcal{B}})$

compute a modular exponentiation using MM_RNS for modular multiplication, the data are set in Montgomery representation $\widetilde{X} = X \times A \mod N$ and in RNS representation $[\widetilde{X}]_{\mathcal{A}\cup\mathcal{B}}$ in the two bases $\mathcal{A}$ and $\mathcal{B}$. The authors of [1] noticed that a way to randomize data X is to randomly change the Montgomery factor A in

$$\widetilde{X} = X \times A \mod N.$$

They proposed to randomize this factor by randomly permuting the moduli of $\mathcal{A}\cup\mathcal{B}$, and then by updating the Montgomery factor in $\widetilde{X}$. They update this Montgomery representation in RNS as it is described in Algorithm 4. Step 1 of the algorithm computes $X \times A_{new} \times A_{old} \mod N$ and in Step 2 we get $X \times A_{new} \mod N$.

Algorithm 4 Update$(X, \mathcal{A}_{old}, \mathcal{B}_{old}, \mathcal{A}_{new}, \mathcal{B}_{new})$

Require: $\widetilde{X}_{old} = X \times A_{old} \mod N$ in $\mathcal{A}_{old} \cup \mathcal{B}_{old}$ and $M = \left(\prod_{i=1}^{t} a_i\right)\left(\prod_{i=1}^{t} b_i\right)$
Ensure: $\widetilde{X}_{new} = XA_{new} \mod N$ in $\mathcal{A}_{new} \cup \mathcal{B}_{new}$
1: $T \leftarrow$ MM_RNS$(\widetilde{X}_{old}, M \mod N, \mathcal{B}_{new}, \mathcal{A}_{new})$
2: $\widetilde{X}_{new} \leftarrow$ MM_RNS$(T, 1, \mathcal{B}_{old}, \mathcal{A}_{old})$
3: **return** $\widetilde{X}_{new}$

This update of the Montgomery representation in RNS makes it possible to randomize any modular exponentiation in RNS. We provide in Algorithm 5 the randomized Right-to-left Square-and-multiply-always exponentiation in RNS. In this algorithm the data are randomized as follows: the variable $\widetilde{Z}$ is updated at each iteration by changing the Montgomery factor A_{old} to A_{new} using the Update algorithm. Then the two variables $\widetilde{R}_0$ and $\widetilde{R}_1$ are updated before each multiplication by $\widetilde{Z}$: this update is from the RNS bases used the last time $\widetilde{R}_0$ (resp. $\widetilde{R}_1$) was updated: we denote $\mathcal{A}_0$ and $\mathcal{B}_0$ (resp. $\mathcal{A}_1$ and $\mathcal{B}_1$) these bases. This approach requires four multiplications per iteration: two per variable update (cf. Algorithm 3).

4 Proposed randomized right-to-left exponentiation in RNS

In this section we present a modified version of the randomized right-to-left square-and-multiply-always approach reviewed in Subsection 3.2. The main idea is quite simple: we remove the Update for the two variables $\widetilde{R}_1$ and $\widetilde{R}_0$ in order to reduce the cost of the randomization. We will call this approach the *lazy leak resistant square-and-multiply-always exponentiation*. It is shown in details in Algorithm 6.

The main thing we have to deal with in this lazy version of Algorithm 5 is the following: we do not control anymore the Montgomery factor in $\widetilde{R}_1$. Indeed if $\widetilde{R}_1^{(i)}$

Algorithm 5 Right-to-left Square-and-multiply-always randomized with LRA

Require: An exponent $E = (e_{\ell-1}, \ldots, e_0)_2$, N and $X \in \{0, \ldots, N-1\}$ expressed in the RNS base $\mathcal{M} = \{m_1, \ldots, m_{2t}\}$.
Ensure: $R_1 = X^E \mod N$

$\mathcal{A}_{old}, \mathcal{B}_{old} \leftarrow^{\text{random split}} \mathcal{M}$
$Z \leftarrow \text{MM\_RNS}(X, M, \mathcal{A}_{old}, \mathcal{B}_{old})$
$\widetilde{R}_0 \leftarrow \text{MM\_RNS}(M, 1, \mathcal{B}_{old}, \mathcal{A}_{old})$
$\mathcal{A}_0, \mathcal{B}_0 \leftarrow \mathcal{A}_{old}, \mathcal{B}_{old}$
$\widetilde{R}_1 \leftarrow \widetilde{R}_0$
$\mathcal{A}_1, \mathcal{B}_1 \leftarrow \mathcal{A}_{old}, \mathcal{B}_{old}$
for i **from** 0 **to** $\ell - 1$ **do**
    $\widetilde{R}_{e_i} \leftarrow \text{Update}(R_{e_i}, \mathcal{A}_{e_i}, \mathcal{B}_{e_i}, \mathcal{A}_{old}, \mathcal{B}_{old})$
    $\mathcal{A}_{e_i}, \mathcal{B}_{e_i} \leftarrow \mathcal{A}_{old}, \mathcal{B}_{old}$
    $\widetilde{R}_{e_i} \leftarrow \text{MM\_RNS}(\widetilde{R}_{e_i}, \widetilde{Z}, \mathcal{A}_{old}, \mathcal{B}_{old})$
    $\widetilde{Z} \leftarrow \text{MM\_RNS}(\widetilde{Z}, \widetilde{Z}, \mathcal{A}_{old}, \mathcal{B}_{old})$
    $\mathcal{A}_{new}, \mathcal{B}_{new} \leftarrow^{\text{random split}} \mathcal{M}$
    $\widetilde{Z} \leftarrow \text{Update}(\widetilde{Z}, \mathcal{A}_{old}, \mathcal{B}_{old}, \mathcal{A}_{new}, \mathcal{B}_{new})$
    $\mathcal{A}_{old}, \mathcal{B}_{old} \leftarrow \mathcal{A}_{new}, \mathcal{B}_{new}$
$R_1 \leftarrow MM\_RNS(\widetilde{R}_1, 1, \mathcal{B}_1, \mathcal{A}_1)$
return R_1

Algorithm 6 Lazy Leak Resistant Right-to-left Square-and-multiply-always

Require: $X \in \{0, \ldots, N-1\}$ and $E = (e_{\ell-1}, \ldots, e_0)_2$ and $\mathcal{M} \mod N$ a moduli set
Ensure: $X^E \mod N$

$\widetilde{R}_0 \leftarrow 1, \widetilde{R}_1 \leftarrow 1$
$\mathcal{A}_{old}, \mathcal{B}_{old} \leftarrow^{\text{random split}} \mathcal{M}$
$\widetilde{Z} \leftarrow \text{MM\_RNS}(X, M, \mathcal{A}_{old}, \mathcal{B}_{old})$
for i **from** 0 **to** $\ell - 1$ **do**
    $\mathcal{A}', \mathcal{B}' \leftarrow^{\text{random split}} \mathcal{M}$
    if $e_i = 0$ **then**
        $\widetilde{R}_0 \leftarrow \text{MM\_RNS}(\widetilde{R}_0, \widetilde{Z}, \mathcal{A}', \mathcal{B}')$
    else
        $\widetilde{R}_1 \leftarrow \text{MM\_RNS}(\widetilde{R}_1, \widetilde{Z}, \mathcal{A}', \mathcal{B}')$
    $\widetilde{Z} \leftarrow \text{MM\_RNS}(\widetilde{Z}, \widetilde{Z}, \mathcal{A}_{old}, \mathcal{B}_{old})$
    $\mathcal{A}_{new}, \mathcal{B}_{new} \leftarrow^{\text{random split}} \mathcal{M}$
    $\widetilde{Z} \leftarrow \text{Update}(\widetilde{Z}, \mathcal{A}_{new}, \mathcal{B}_{new}, \mathcal{A}_{old}, \mathcal{B}_{old})$
    $\mathcal{A}_{old}, \mathcal{B}_{old} \leftarrow \mathcal{A}_{new}, \mathcal{B}_{new}$
return $\widetilde{R}_1$

is the value of $\widetilde{R}_1$ at the beginning of the of loop iteration i it satisfies

$$\widetilde{R}_1^{(i)} = X^{\sum_{j=0}^{i-1} e_j 2^j} \times \prod_{j=0}^{2t} m_j^{\gamma_j^{(i)}}$$

for some $\gamma_j^{(i)} \in \mathbb{Z}$ for $j = 1, \ldots, 2t$. This can be proven by induction. For $i = 0$ we can see it at the initialization of $\widetilde{R}_1 = 1$. Now if we suppose that it is true for i, we show it for $i + 1$. We process loop i to get $\widetilde{R}_1^{(i+1)}$ as follows:

- If $e_i = 0$ then $\widetilde{R}_1$ is not modified and then

$$\widetilde{R}_1^{(i+1)} = \widetilde{R}_1^{(i)} = X^{\sum_{j=0}^{i} e_j 2^j} \times \prod_{j=0}^{2t} m_j^{\gamma_j^{(i)}}.$$

  Thus taking $\gamma_j^{(i+1)} = \gamma_j^{(i)}$ provides the correct result.
- If $e_i = 1$ we have

$$\begin{aligned} \widetilde{R}_1^{(i+1)} &= \widetilde{R}_1^{(i)} \times \widetilde{Z} \times (A'^{(i)}) \mod N \\ &= \widetilde{R}_1^{(i)} \times X^{2^i} \times A^{(i)} \times (A'^{(i)})^{-1} \mod N \end{aligned}$$

  where $A^{(i)}$ and $A'^{(i)}$ are both products of t moduli m_j among $\mathcal{M}$, i.e., the ones of $\mathcal{A}_{old}^{(i)}$ and $\mathcal{A}'^{(i)}$ respectively. This leads to

$$\widetilde{R}_1^{(i+1)} = X^{\sum_{j=0}^{i} e_j 2^j} \times \prod_{j=0}^{2t} m_i^{\gamma_j^{(i)} + \delta_j^{(i)}}$$

  where

$$\delta_j^{(i)} = \begin{cases} 1 \text{ if } m_j | A^{(i)} \text{ and } m_j \not| A'^{(i)}, \\ -1 \text{ if } m_j \not| A^{(i)} \text{ and } m_j | A'^{(i)}, \\ 0 \text{ otherwise.} \end{cases} \quad (5)$$

  Then taking $\gamma_j^{(i+1)} = \gamma_j^{(i)} + \delta_j^{(i)}$, leads to the required expression of $\widetilde{R}_1^{(i+1)}$.

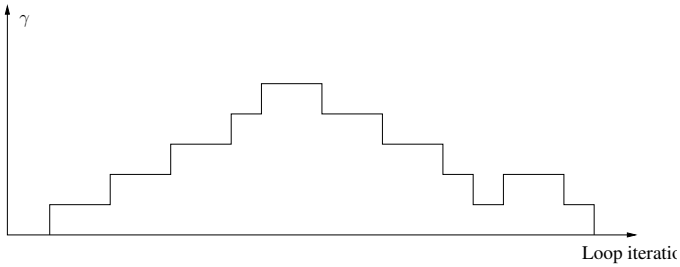
In the sequel we will call $\prod_{j=0}^{2t} m_j^{\gamma_j^{(i)}}$ the multiplicative mask of $\widetilde{R}_1^{(i)}$ since it is a random data masking the real value of $R_1^{(i)}$. The above analysis shows that the exponents $\gamma_j^{(i)}$ of the multiplicative mask in $\widetilde{R}_1^{(i)}$ have always an absolute value $\leq i$. This means that at the end of the loop **for** of Algorithm 6 we have $|\gamma_j^{(\ell)}| \leq \ell$ for all j.

4.1 Strategy to remove the final multiplicative mask

We present in this section a strategy to remove the final multiplicative mask of $\widetilde{R}_1$ at the end of Algorithm 6. We investigate a strategy which reduces the level of randomization, but ensures that this final multiplicative mask is trivial and avoid a final correction. We first notice that the multiplicative mask is modified only when $e_i = 1$. Then let h_E be the Hamming weight of the exponent E. The proposed strategy lets the exponent evolve freely for the $\lfloor h_E/2 \rfloor$ iterations such that $e_i = 1$. Then for the last $\lceil h_E/2 \rceil$ bits $e_i = 1$ we force the exponent $\gamma_j^{(i)}$ to decrease toward 0 as it is shown in Figure 2.

The details of this strategy is given in the following lemma.

Fig. 2. Strategy 2 for the final correction



Lemma 1. *Let $E = (e_{\ell-1}, e_{\ell-2}, \ldots, e_0)$ be an exponent and let $E_i = (e_{i-1}, \ldots, e_0)_2$. Let*

$$\widetilde{R}_1^{(i)} = x^{E_i} \prod_{j=0}^{2t} m_j^{\gamma_j^{(i)}} \mod N$$

be the value of $\widetilde{R}_1^{(i)}$ at the beginning of iteration i in Algorithm 6. Let h_E be the Hamming weight of E and let $i_1, \ldots, i_{h_E}$ be the subscripts such that $e_{i_k} = 1$. If the randomizations of the bases $\mathcal{A}, \mathcal{B}, \mathcal{A}'$ and $\mathcal{B}'$ we $e_i = 1$ are done under the following restrictions:

i) if $\gamma_j^{(i_k)} = h_E - k$ then $j \in \mathcal{A}'$ and $j \notin \mathcal{A}$,
ii) if $\gamma_j^{(i_k)} = -(h_E - k)$ then $j \in \mathcal{A}$ and $j \notin \mathcal{A}'$,
iii) if $\gamma_j^{(i_k)} = h_E - k - 1$ then $j \notin \mathcal{A}$,
iv) if $\gamma_j^{(i_k)} = -(h_E - k - 1)$ then $j \notin \mathcal{A}'$,

then for $j = 1, \ldots, 2t$ and $i_k \in \{i_1, \ldots, i_{h_E}\}$ we have

$$|\gamma_j^{(i_k)}| \leq (h_E - k).$$

Proof. The idea of the proof is quite simple: we always have to be at distance at most $h_E - k$ from 0, since from loop i_k to loop i_{h_E} we can only do $h_E - k$ non-zero steps, i.e., at loop iterations $i_{k+1}, \ldots, i_{h_E}$. This implies the following:

- If at step i_k we are at distance $h_E - k$ we have to make a non-zero step toward 0 ("-1" if $\gamma_j^{(i_k)} = h_E - k$ and "+1" if $\gamma_j^{(i_k)} = -(h_E - k)$) this exactly what i) and ii) do.
- If at step i_k we are at distance $h_E - k - 1$ we have to either stay at distance $h_E - k - 1$ or make step toward 0 this is exactly what iii) and iv) do.
- Otherwise we can move freely.

Then by induction on k we easily see that the claim of the lemma is satisfied.

The previous lemma shows that under the restrictions given by i) to iv) we ensure that all final exponents $\gamma_j^{(\ell)} = 0$. Which implies $\widetilde{R}_1^{(\ell)} = R_1$. The disadvantage of this approach is that it reduces the level of randomization in the second phase of the exponentiation.

Table 1. Complexities of randomized exponentiation

Algorithm	# MM_RNS
Montgomery-ladder [1]	6ℓ
Proposed Leak Resistant Right-to-left	4ℓ

4.2 Complexity comparison

Table 1 contains the complexity of the proposed approach along with the one of [1] which was originally proposed for the Montgomery-ladder.

The complexities in Table 1 shows that the proposed approach is always the best option: it saves 30% of the amount of computation.

Remark 1. In their paper [1] Bajard *et al.* propose also a trade-off approach which randomizes the data only every f loop iterations. We can apply this trade-off approach to Algorithm 6, and this will reduce the complexity by $2\ell/f$ MM_RNS compared to [1].

5 Level of randomization

The goal of this section is to evaluate the level of randomization induced by the multiplicative mask of $\widetilde{R}_1^{(i)}$ in our modified randomized modular exponentiation. We first analyze the behavior of the exponent $\gamma_j^{(i)}$.

5.1 Random walk of the exponents $\gamma_j^{(i)}$

We first study the way the exponents $\gamma_j^{(i)}$ of $\prod_{j=0}^{2t} m_j^{\gamma_j^{(i)}}$ evolve during the proposed randomized exponentiation. We do not study the evolution of $\widetilde{R}_0^{(i)}$ since we do not have to correct the multiplicative mask at the end, but it evolves similarly as $\widetilde{R}_1^{(i)}$ does. We denote $\Gamma^{(i)} = (\gamma_1^{(i)}, \ldots, \gamma_{2k}^{(i)})$ the vector of the exponents of the multiplicative mask of $\widetilde{R}_1^{(i)}$. We can notice that each $\gamma_j^{(i)}$ for a fixed j behaves as a random walk for $i = 1, 2, \ldots, \ell$. For any loop iteration then $\gamma_j^{(i)}$ is modified as

$$\gamma_j^{(i+1)} = \gamma_j^{(i)} + \delta_j^{(i)}$$

and $\delta_j^{(i)}$ is chosen at random at follows:

$$\begin{aligned} \mathbb{P}(\delta_j^{(i)} = 1) &= 1/8, \\ \mathbb{P}(\delta_j^{(i)} = -1) &= 1/8, \\ \mathbb{P}(\delta_j^{(i)} = 0) &= 3/4. \end{aligned} \tag{6}$$

Indeed, we first notice that $\mathcal{A}^{(i)}, \mathcal{A}'^{(i)}$ and $(\mathcal{A}^{(i)})^c$ and $(\mathcal{A}'^{(i)})^c$ the RNS bases used in iteration i have all t elements. Now since the events $\{e_i = 1\}$, $\{m_j \in \mathcal{A}^{(i)}\}$ and $\{m_j \in \mathcal{A}'^{(i)}\}$ are independent we have:

$$\begin{aligned}
\mathbb{P}(\delta_j^{(i)}=1) &= \mathbb{P}(e_i=1, m_j \in \mathcal{A}^{(i)} \backslash \mathcal{A}'^{(i)}) \\
&= \mathbb{P}(e_i=1) \times \mathbb{P}(m_j \in \mathcal{A}^{(i)}) \times \mathbb{P}(m_j \in (\mathcal{A}'^{(i)})^c) \\
&= \tfrac{1}{2} \times \tfrac{1}{2} \times \tfrac{1}{2} = \tfrac{1}{8}, \\
\mathbb{P}(\delta_j^{(i)}=-1) &= \mathbb{P}(e_i=1, m_j \in \mathcal{A}'^{(i)} \backslash \mathcal{A}^{(i)}) \\
&= \mathbb{P}(e_i=1) \times \mathbb{P}(m_j \in \mathcal{A}'^{(i)}) \times \mathbb{P}(m_j \in (\mathcal{A}^{(i)})^c) \\
&= \tfrac{1}{2} \times \tfrac{1}{2} \times \tfrac{1}{2} = \tfrac{1}{8}, \\
\mathbb{P}(\delta_j^{(i)}=0) &= \mathbb{P}(e_i=0) + \mathbb{P}(e_i=1, m_j \in \mathcal{A}'^{(i)} \cap \mathcal{A}^{(i)}) \\
&\quad + \mathbb{P}(e_i=1, m_j \in (\mathcal{A}'^{(i)})^c \cap (\mathcal{A}^{(i)})^c) \\
&= \tfrac{1}{2} + \tfrac{1}{2} \times \tfrac{1}{2} \times \tfrac{1}{2} + \tfrac{1}{2} \times \tfrac{1}{2} \times \tfrac{1}{2} = \tfrac{3}{4}.
\end{aligned}$$

Finally, the exponent $\gamma_j^{(i)}$ behaves as a random walk with step size $\delta_j^{(i)} \in \{1, -1, 0\}$ following the probability given in (6). The following lemma provides the probability $\mathbb{P}(\gamma_j^{(i)} = d)$ for any $d \in \{-i, -(i-1), \ldots, i-1, i\}$.

Lemma 2. *Let $\gamma_j^{(i)}$ be the exponent of m_j in the factor of $\widetilde{R}_1^{(i)}$ at the beginning of the i-th loop iterations. Then, for a given $d \in \{-i, -(i-1), \ldots, (i-1), i\}$ we have*

$$\mathbb{P}(\gamma_j^{(i)} = d) = \sum_{k=d}^{d+\lfloor (i-d)/2 \rfloor} \binom{i}{k} \binom{i-k}{k-d} \left(\frac{1}{8}\right)^{2k-d} \left(\frac{3}{4}\right)^{i-2k+d}. \tag{7}$$

Proof. We consider the random walks which have k steps "1" and $k-d$ steps "-1" and $i-k-(k-d)$ step "0". The number of these random walks is

$$\binom{i}{k}\binom{i-k}{k-d}.$$

Each of these random walks has a probability of

$$\left(\frac{1}{8}\right)^{k} \times \left(\frac{1}{8}\right)^{k-d} \times \left(\frac{3}{4}\right)^{i-2k+d}.$$

We get the probability for $\gamma_j^{(i)} = d$ by summing over all acceptable values for k and this leads to (7).

We can also evaluate the expected value $\mathbb{E}(\gamma_j^{(i)})$ and the variance $\mathbb{V}(\gamma_j^{(i)})$ of the random variable $\gamma_j^{(i)}$ at iteration i. This can be easily obtained since $\gamma_j^{(i)}$ is the sum of i independent random variables $\delta_j^{(k)}, k = 1, \ldots, i$ with the probability law of (6). In other words:

$$\begin{aligned}
\mathbb{E}(\gamma_j^{(i)}) &= \textstyle\sum_{k=0}^{i-1} \mathbb{E}(\delta_j^{(k)}) = \ell \times 0 = 0 \\
\mathbb{V}(\gamma_j^{(i)}) &= \textstyle\sum_{k=0}^{i-1} \mathbb{V}(\delta_j^{(k)}) = i/4
\end{aligned}$$

And then the standard deviation is $\mathrm{SD}(\gamma_j^{(i)}) = \sqrt{i}/2$. In other words $\gamma_j^{(i)}$ is in $\{-i, \ldots, i\}$, but it is more likely that its absolute value is less than $\sqrt{i}/2$.

The following lemma shows that for a fixed i, this probability $\mathbb{P}(\delta_j^{(i)} = d)$ decreases as $|d|$ increases.

Lemma 3. *Let j be a fixed integer in $\{1, \ldots, 2t\}$. We have for any i and any d, d' such that $|d| \leq |d'|$:*

$$\mathbb{P}(\lambda_j^{(i)} = d) \geq \mathbb{P}(\lambda_j^{(i)} = d').$$

The proof of Lemma 3 is given in the appendix.

5.2 Bounding the level of randomization

Let $\Gamma \in \mathbb{Z}^{2t}$, then we denote $\sigma(\Gamma) = \sum_{j=0}^{2t} \gamma_j$. In this subsection we will first give the exact condition for a vector $\Gamma = (\gamma_1, \ldots, \gamma_{2t}) \in \mathbb{Z}^{2t}$ to be i-th term of a sequence of exponent vectors in Algorithm 6. In other word we will consider the vectors for which there exists a path $\Gamma^{(k)}, k = 1, \ldots, i$ with $\Gamma^{(i)} = \Gamma$ and

$$\Gamma^{(k)} = \Gamma^{(k-1)} + \Delta^{(k)}$$

and $\Delta^{(i)} \in \{0, 1, -1\}^{2t}$ and $\sigma(\Delta) = 0$. We will say that such vector can be reached by such sequence. This is the goal of the next proposition.

Proposition 1. *We consider a vector $\Gamma = (\gamma_1, \ldots, \gamma_{2t}) \in \mathbb{Z}^{2t}$. The vector Γ can be reached by a sequence of length i of exponent vector produced by Algorithm 6*

$$\Gamma^{(0)} = 0, \Gamma^{(1)}, \ldots, \Gamma^{(i)} = \Gamma$$

if and only if we have

$$\begin{aligned} \|\Gamma\|_\infty &\leq i \\ \sigma(\Gamma) &= 0 \end{aligned}$$

The proof of this proposition is given in the appendix. Our goal now is to bound the probability to obtain a given multiplicative mask after a certain number of iterations. The following lemma states the basic result we will use to bound this probability.

Lemma 4. *Let $\Gamma = (\gamma_1, \ldots, \gamma_{2t})$ be a vector that can be reached after i loop iterations. Then we have:*

$$\mathbb{P}(\Gamma^{(i)} = \Gamma) \leq \prod_{j=1}^{2t} \mathbb{P}(\gamma_j^{(i)} = \gamma_j)$$

Proof. We notice that if $C : \Gamma^{(0)} = 0, \Gamma^{(1)}, \ldots, \Gamma^{(i)} = \Gamma$ is walk reaching Γ then $c_j : \gamma^{(0)} = 0, \gamma^{(1)}, \ldots, \gamma^{(i)} = \gamma_j$ is a walk reaching γ_j in $\mathbb{Z}$. In other words, if $Walk^{(i)}(\Gamma)$ (resp. $Walk^{(i)}(\gamma_j)$) is the set of walks reaching Γ (resp. γ_j), then we have:

$$\{C \in Walk^{(i)}(\Gamma)\} \subset \prod_{j=1}^{2t} \{c_j \in Walk^{(i)}(\gamma_j)\}$$

Taking the probability of both sets leads to:

$$\begin{aligned} \mathbb{P}(\Gamma^{(i)} = \Gamma) &= \mathbb{P}(\{C \in Walk^{(i)}(\Gamma)\}) \\ &\leq \mathbb{P}(\textstyle\prod_{j=1}^{2t} \{c_j \in Walk^{(i)}(\gamma_j)\}) \\ &= \textstyle\prod_{j=1}^{2t} \mathbb{P}(\gamma^{(i)} = \gamma_j) \end{aligned}$$

Now we combine the results of Lemma 3 and Lemma 4 to get a bound on the probability to get Γ after i loop iterations:

$$\mathbb{P}(\Gamma^{(i)} = \Gamma) \leq \prod_{j=1}^{2t} \mathbb{P}(\gamma^{(i)} = \gamma_j) \leq \mathbb{P}(\gamma^{(i)} = 0)^{2t}. \tag{8}$$

5.3 Comparison

With the results of Subsection 5.2 we are able to compare the level of randomization of the proposed approach with the one of [1]. In [1] the multiplicative mask is taken at random in a set of $\binom{2t}{t}$ masks, in other words, with a probability of $1/\binom{2t}{t}$. Then we can say that the level of randomization of the proposed approach after i loop iterations is better than the one of [1] if the probability of any mask is smaller than $1/\binom{2t}{t}$.

We were not able to compare these two probabilities formally. But we could compute the probabilities or probability bounds for practical sizes of N and t in Table 2. For each value i we computed $\mathbb{P}(\lambda^{(i)} = 0)$ using the formula of Lemma 2, we then could compute a bound of the probability $\mathbb{P}(\Gamma^{(i)} = \Gamma)$ for any Γ with (8).

Table 2. Level of randomization for 2048 bits and $t = 32$

Approach	loop 1	loop 2	loop 3	loop 4	loop 5	loop 10
Montg.-ladder [1]	$4.17 \cdot 10^{-38}$	$4.17 \cdot 10^{-38}$	$4.17 \cdot 10^{-38}$	$4.17 \cdot 10^{-38}$	$4.17 \cdot 10^{-38}$	$4.17 \cdot 10^{-38}$
Proposed	10^{-8}	$3 \cdot 10^{-15}$	$2 \cdot 10^{-20}$	$1.3 \cdot 10^{-24}$	$5 \cdot 10^{-28}$	$1.7 \cdot 10^{-38}$
Approach	loop 15	loop 20	loop 50	loop 100	loop 500	loop 1000
Montg.-ladder [1]	$4.17 \cdot 10^{-38}$	$4.17 \cdot 10^{-38}$	$4.17 \cdot 10^{-38}$	$4.17 \cdot 10^{-38}$	$4.17 \cdot 10^{-38}$	$4.17 \cdot 10^{-38}$
Proposed	$2.4 \cdot 10^{-44}$	$2 \cdot 10^{-48}$	$2.69 \cdot 10^{-61}$	$5.75 \cdot 10^{-71}$	$2.31 \cdot 10^{-93}$	$5.34 \cdot 10^{-103}$

The probability for Montgomery-ladder with [1] is the same for all iteration and is equal to $1\binom{128}{64} \cong 4.17 \cdot 10^{-38}$. The values in Table 2 shows that we cannot assert the randomization is better for the first 10 loop iterations. But after the i-th loop it becomes clearly better. This problem can be easily overcome by using the randomization of [1] for these few first 10 loop iterations and then use our proposed approach for the other iterations.

6 Comparison and conclusion

In this paper we proposed a lazy version of the leak resistant square-and-multiply-always approach of [1]. We used a right-to-left version of the square-and-multiply exponentiation and remove the Update on the Montgomery representation of the two variables $\widetilde{R}_0$ and $\widetilde{R}_1$. This lead to a reduction of the complexity. We also studied the impact of proposed modification on the randomization level. The proposed approach has the advantage to offer a higher level of randomization after a small number of loop iterations.

References

1. J.-C. Bajard, L. Imbert, P.-Y. Liardet, and Y. Teglia. Leak Resistant Arithmetic. In *CHES*, volume 3156 of *LNCS*, pages 62–75. Springer, 2004.
2. E. Brier, C. Clavier, and F. Olivier. Correlation Power Analysis with a Leakage Model. In *CHES 2004*, volume 3156 of *LNCS*, pages 16–29. Springer, 2004.
3. J.-S. Coron. Resistance against Differential Power Analysis for Elliptic Curve Cryptosystems. In *CHES 1999*, pages 292–302, 1999.
4. M. Joye and S.-M. Yen. The Montgomery Powering Ladder. In *CHES 2002*, volume 2523 of *LNCS*, pages 291–302. Springer, 2002.
5. P. C. Kocher, J. Jaffe, and B. Jun. Differential Power Analysis. In *Advances in Cryptology, CRYPTO'99*, volume 1666 of *LNCS*, pages 388–397. Springer, 1999.

6. P.C. Kocher. Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. In *Advances in Cryptology - CRYPTO '96*, volume 1109 of *LNCS*, pages 104–113. Springer, 1996.
7. P.C. Kocher, J. Jaffe, B. Jun, and P. Rohatgi. Introduction to differential power analysis. *J. Cryptographic Engineering*, 1(1):5–27, 2011.
8. S. Mangard. Exploiting Radiated Emissions - EM Attacks on Cryptographic ICs. In *Austrochip 2003, Linz, Austria, October 1st*, pages 13–16, 2003.
9. P. Montgomery. Modular Multiplication Without Trial Division. *Math. Computation*, 44:519–521, 1985.
10. K.C. Posch and R. Posch. Modulo Reduction in Residue Number Systems. *IEEE Trans. Parallel Distrib. Syst.*, 6(5):449–454, 1995.

A Proof of Lemma 3

Before proceeding we notice that $\mathbb{P}(\lambda_j^{(i)} = d) = 0$ for any $|d| > i$ this justified by the fact that $|\lambda_j^{(i)}| \leq i$. Now we prove the lemma by induction on i.

- For $i = 1$ we have $\mathbb{P}(\lambda_j^{(1)} = 0) = 3/4$ is larger than $\mathbb{P}(\lambda_j^{(1)} = 1) = \mathbb{P}(\lambda_j^{(1)} = -1) = 1/8$ which are larger than $\mathbb{P}(\lambda_j^{(1)} = d) = 0$ for $|d| > 1$.
- We assume that the assertion is true for $i-1$ and we prove it for i. We first prove for $d > 0$ and $d + 1 \leq i$ that $\mathbb{P}(\lambda_j^{(i)} = d) \geq \mathbb{P}(\lambda_j^{(i)} = d + 1)$:

$$\begin{aligned}\mathbb{P}(\lambda_j^{(i)} = d) &= \tfrac{1}{8}\mathbb{P}(\lambda_j^{(i-1)} = d+1) + \tfrac{3}{4}\mathbb{P}(\lambda_j^{(i-1)} = d) + \tfrac{1}{8}\mathbb{P}(\lambda_j^{(i-1)} = d-1)\\ &\geq \tfrac{1}{8}\mathbb{P}(\lambda_j^{(i-1)} = d+2) + \tfrac{3}{4}\mathbb{P}(\lambda_j^{(i-1)} = d+1) + \tfrac{1}{8}\mathbb{P}(\lambda_j^{(i-1)} = d)\\ &= \mathbb{P}(\lambda_j^{(i)} = d+1)\end{aligned}$$

- The case $d > 0$ and $d + 1 > i$ is trivial.
- The proof for $\mathbb{P}(\lambda_j^{(i)} = 0) \geq \mathbb{P}(\lambda_j^{(i)} = 1)$, i.e. for $d = 0$, is slightly different, but very close the case $d > 0$. We then skip it.
- For $d < 0$ it is symmetric to $d > 0$.

B Proof of Proposition 1

The fact that each γ_j have to satisfy $|\gamma_j| \leq i$ is direct since there are i loop iterations and in each loop iteration we can increase or decrease each γ_j by at most 1. We can further assume without loss of generality that $i = \|\Gamma\|_\infty$. We prove the proposition by induction on t and on i.

We first deal with $t = 1$ and $i = 1$:

- If $t = 1$. We have $\Gamma = (\gamma_1, \gamma_2)$ and by assumption this vector satisfies $\gamma_1 + \gamma_2 = 0$. Without loss of generality we assume that $\gamma_1 \geq 0$ and $\gamma_2 \leq 0$. We have also that $|\gamma_1| = |\gamma_2| = \|\Gamma\|_\infty$. We can easily see that Γ can be reached after $\|\Gamma\|_\infty$ loop iteration, where we increase by 1 at each loop iteration $\gamma_1^{(k)}$ and decrease by 1 at each loop turn $\gamma_2^{(k)}$.
- If $i = 1$, we have $\|\Gamma\|_\infty \leq 1$ and $\sum_{j=0}^{2t} \gamma_j = 0$, but in this case the sequence with only one element $\Gamma^{(1)} = \Gamma$ is a good sequence.

Now we assume that the lemma is true up do t and i and we prove it for $t + 1$ and $i+1$. We assume that $\|\Gamma\|_\infty = i+1$ and $\Gamma = (\gamma_1, \ldots, \gamma_{2t}, \gamma_{2t+1}, \gamma_{2t+2})$ such that

- γ_{2t+2} is ≥ 0 and $\gamma_{2t+2} = \|\Gamma\|_\infty$,
- γ_{2t+1} is the smallest coefficient among $\gamma_j, j = 1, \ldots, 2t + 2$,

We deal with a special case: γ_{2t+2} is the only $\gamma_j \geq 0$. We have

$$\gamma_{2t+2} = -\sum_{j=1}^{2t+1} \gamma_j$$

we can easily define a sequence $\Gamma^{(k)}, k = 0, \ldots, i+1$ reaching Γ as follows: at each loop we increase $\gamma_{2t+2}^{(k)}$ and we decrease one $\gamma_j^{(k)}$ such that $\gamma_j^{(k)} \neq \gamma_j$.

The other cases: there exists $j \neq 2t+2$ such that $\gamma_j \geq 0$. For the sake of simplicity, we assume that this γ_j is γ_{2t}. In other words, without loss of generality, we can assume that $\Gamma = (\gamma_1, \ldots, \gamma_{2t}, \gamma_{2t+1}, \gamma_{2t+2})$ is such that

- γ_{2t+2} is ≥ 0 and $\gamma_{2t+2} = \|\Gamma\|_\infty$,
- γ_{2t+1} is the smallest coefficient among $\gamma_j, j = 1, \ldots, 2t+2$,
- γ_{2t} is the smallest coefficient ≥ 0.

We split the proof into the two following cases:

- *Case $|\gamma_{2t}| \leq |\gamma_{2t+1}|$.* In this case we use an induction on t. We set $\Gamma' = (\gamma_1, \ldots, \gamma_{2t-1}, \gamma_{2t} + \beta)$ where $\beta = |\gamma_{2t+2}| - |\gamma_{2t+1}|$. Then Γ' is a vector of size $2t$ such that $\|\Gamma'\|_\infty \leq i+1$ since

  $$0 \leq \gamma_{2t} + \beta \leq |\gamma_{2t+1}| + |\gamma_{2t+2}| - |\gamma_{2t+1}| \leq \|\Gamma\|_\infty = i+1.$$

  So by induction hypothesis on t there exists a sequence reaching Γ'

  $$\Gamma'^{(0)}, \Gamma'^{(1)}, \ldots, \Gamma'^{(i+1)} = \Gamma'$$

  We define $\delta_j'^{(k-1)} = \gamma_j'^{(k)} - \gamma_j'^{(k-1)}$ which lies in $\{-1, 0, 1\}$. Now since $\gamma_{2t}'^{(k)}$ reaches γ'_{2t} there are at least $s = \gamma_{2t}'^{(k)} = \gamma_{2t} + \beta$ superscripts k where $\delta_{2t}^{(k)} = 1$. Let $k_1, \ldots, k_s$ such superscripts. We can now define $\delta_j^{(k)}$ for Γ as follows:

  $$\begin{aligned}
  \delta_j^{(k)} &= \delta_j'^{(k)} \text{ for } j = 1, \ldots, 2t-1 \text{ and } k = 0, \ldots, i, \\
  \delta_{2t}^{(k)} &= \begin{cases} 0 \text{ if } k \in \{k_1, \ldots, k_\beta\}, \\ \delta_{2t}'^{(k)} \text{ otherwise.} \end{cases} \\
  \delta_{2t+1}^{(k)} &= \begin{cases} 0 \text{ if } k \in \{k_1, \ldots, k_\beta\}, \\ -1 \text{ otherwise.} \end{cases} \\
  \delta_{2t+2}^{(k)} &= 1.
  \end{aligned}$$

  In other word for $k \in \{k_1, \ldots, k_\beta\}$ we moved the 1 of $\delta_{2t}'^{(k)}$ to $\delta_{2t+2}^{(k)}$. And for the other superscripts we set $\delta_{2t+2}^{(k)} = 1$ and $\delta_{2t+1}^{(k)} = -1$. Then one can see that for each k we have $\sigma(\Delta^{(k)}) = 0$ and also:

  $$\begin{aligned}
  \textstyle\sum_{k=0}^{i} \delta_j^{(k)} &= \gamma_j \text{ for } j = 1, \ldots, 2t-1 \text{ and } k = 1, \ldots, i+1 \\
  \textstyle\sum_{k=0}^{i} \delta_{2t}^{(k)} &= (\textstyle\sum_{k=1}^{i} \delta_{2t}'^{(k)}) - \beta = (\gamma_{2t} + \beta) - \beta = \gamma_{2t} \\
  \textstyle\sum_{k=0}^{i} \delta_{2t+1}^{(k)} &= -(i+1-\beta) = \gamma_{2t+1} \\
  \textstyle\sum_{k=0}^{i} \delta_{2t+2}^{(k)} &= i+1 = \gamma_{2t+2}
  \end{aligned}$$

  which means that the sequence $\Gamma^{(k)}, k = 0, \ldots, i+1$ reaches Γ.

– *Case* $|\gamma_{2t}| > |\gamma_{2t+1}|$. This means we have all $\gamma_j \neq 0$ (since γ_{2t} is the smallest among $\gamma_j \geq 0$). Moreover any $\gamma_j \geq 0$ satisfies $\gamma_j > |\gamma_s|$ for all $\gamma_s < 0$. In this case we have

$$\#\{j \text{ s.t. } \gamma_j > 0\} < \#\{j \text{ s.t. } \gamma_j < 0\}$$

otherwise it would contradict the fact that $\sigma(\Gamma) = 0$. Let $j_1, \ldots, j_r$ be the subscripts such that $\gamma_j > 0$ and let $j'_1, \ldots, j'_r$ be r subscripts in $\{j \text{ s.t. } \gamma_j < 0\}$. We define $\Delta^{(i)}$ by

$$\begin{aligned} \delta_j^{(i)} &= 1 \text{ for } j \in \{j_1, \ldots, j_r\}, \\ \delta_j^{(i)} &= -1 \text{ for } j \in \{j'_1, \ldots, j'_r\}, \\ \delta_j^{(i)} &= 0 \text{ otherwise.} \end{aligned}$$

Then $\Gamma' = \Gamma - \Delta^{(i)}$ satisfies $\|\Gamma'\|_\infty \leq i$ and then, by induction on i, Γ' can be reached by a sequence of length i. If we complete this sequence reaching Γ' with $\Gamma^{(i+1)} = \Gamma$ we obtain a sequence reaching Γ of length $i + 1$.

This ends the proof of Proposition 1.