Efficient Randomized Regular Modular Exponentiation using Combined Montgomery and Barrett Multiplications

Andrea Lesavourey, Christophe Negre, Thomas Plantard

Efficient Randomized Regular Modular Exponentiation using Combined Montgomery and Barrett Multiplications

Andrea Lesavourey[1], Christophe Negre[1] and Thomas Plantard[2]
[1] *DALI (UPVD) and LIRMM (Univ. of Montpellier, CNRS), Perpignan, France*
[2] *CCISR, SCIT, University of Wollongong, Wollongong, Australia*
andrea.lesavourey@outlook.fr, christophe.negre@univ-perp.fr,thomaspl@uow.edu.au

Keywords:
RSA, modular exponentiation, Barrett, Montgomery, differential power analysis, correlation power analysis, randomization

Abstract:
Cryptographic operations performed on an embedded device are vulnerable to side channel analysis and particularly to differential and correlation power analysis. The basic protection against such attacks is to randomize the data all along the cryptographic computations. In this paper we present a modular multiplication algorithm which can be used for randomization. We show that we can use it to randomize the modular exponentiation of the RSA cryptosystem. The proposed randomization is free of computation and induces a level of randomization from 2^{10} to 2^{15} for practical RSA modulus size.

1 Introduction

Modern digital communications are intensively encrypted and authenticated to ensure a good level of confidentiality and security. Public key encryption and signature is a concept initiated in 1976 by Diffie and Hellman. This concept was realized by Rivest Shamir and Adlemann who proposed the RSA cryptosystem in (Rivest et al., 1978). This RSA cryptosystem is nowadays the most used public key scheme for electronic signature and remote authentication.

The basic operation in RSA protocols is an exponentiation modulo a integer N which is of size 2048-4096 bits. This exponentiation is generally computed through a sequence of a few thousands squarings and multiplications modulo N using the Square-and-multiplication exponentiation scheme. Unfortunately, a naive implementation of this algorithm on an embedded device could be threaten by side channel analysis. These attacks monitor either power consumption (Kocher et al., 1999), electromagnetic emanation (Mangard, 2003) or computation time (Kocher, 1996) in order to extract the secret exponent.

The kind of attacks we will consider here are the simple power analysis (SPA) (Kocher et al., 1999), the differential and correlation power analysis (Kocher et al., 1999; Brier et al., 2004). The SPA can be easily defeated by using a regular algorithm for the exponentiation like the Montgomery-ladder (Joye and Yen, 2002) or the Square-and-multiply-always algorithm (Coron, 1999). To counteract the differential and correlation power analysis it is necessary to randomize the data and the computations all along the exponentiation.

In this paper we study a new method to randomize modular exponentiation. This approach is based on a modular multiplication algorithm which randomly combines the two main methods for modular multiplication: Montgomery (Montgomery, 1985) and Barrett multiplications (Barrett, 1987). The advantage of this proposed randomization is that it is free of computation. We then present a modified Montgomery-ladder and a modified Square-and-multiply-always algorithms for modular exponentiation which uses this randomized modular multiplication. For these two proposed randomized exponentiations we study the level of randomization obtained.

The remainder of the paper is organized as follows. In Section 2 we review modular exponenti-

ation and side channel analysis. In Section 3 we review the methods of Montgomery and Barrett for modular multiplication and we present a combined version of these two methods. In Section 4 we study two randomized exponentiations based on the combined Montgomery and Barrett multiplication. Finally, in Section 5, we give some concluding remarks and some perspectives.

2 Exponentiation and side channel analysis

The basic operation in RSA protocols is the modular exponentiation: given an RSA modulus N, an exponent E and $X \in \{0, \ldots, N-1\}$ we have to compute

$$Y = X^E \mod N.$$

Generally, the most sensitive data is the exponent E. This exponentiation can be efficiently computed with the so-called Square-and-multiply algorithm which consists in a sequence of squarings followed by a multiplication by X when the bit e_i of E is equal to 1. This approach is detailed in Algorithm 1.

Algorithm 1 Left-to-right Square-and-multiply

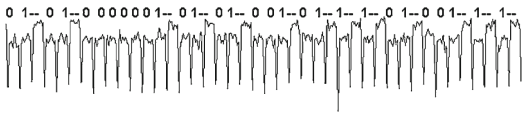
Require: An RSA modulus N, an integer $X \in \{0, \ldots, N-1\}$ and an exponent $E = (e_{\ell-1}, \ldots, e_0)_2$
Ensure: $R_0 = X^E \mod N$
1: $R_0 \leftarrow 1$
2: **for** i **from** $\ell - 1$ **to** 0 **do**
3: $\quad R_0 \leftarrow R_0^2 \mod N$
4: $\quad$ **if** $e_i = 1$ **then**
5: $\quad\quad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

2.1 Side channel analysis of RSA exponentiation

When an RSA exponentiation is computed on an embedded device it is under the threat of side-channel analysis. Such attacks monitor either the power consumption, the electromagnetic emanation or the computation time in order to extract the secret data. We review the simple power analysis (SPA) and differential/correlation power analysis (Kocher et al., 1999; Brier et al., 2004):

- **Simple power analysis.** The simple power analysis (Kocher et al., 1999) threatens implementations based on the Square-and-multiply exponentiation. Indeed, if the squaring of a data has a different power trace than a multiplication, the eavesdropper can read on the power trace the sequence of squarings and multiplications which were computed. Then, since a multiplication is done only when $e_i = 1$, the attacker can easily deduce the sequence of bits of the exponent. Figure 1 illustrates this fact.

Figure 1 – SPA attack on the Square-and-multiply exponentiation from (Kocher et al., 2011)



Counter-measures. SPA can be easily defeated by using a regular algorithm for the computation of the exponentiation. For example the Square-and-multiply-always approach proposed in (Coron, 1999) is a variant of the Square-and-multiply (Algorithm 1) which performs a dummy multiplication when the bit $e_i = 0$.

Another popular approach is the Montgomery-ladder which is shown in Algorithm 2. This approach is a variant of Algorithm 1 where a second variable R_1 is used and always satisfies $R_1 = R_0 \times X \mod N$. In the Montgomery-ladder, when $e_i = 1$, the two instructions $R_0 \leftarrow R_0^2$ and $R_0 \leftarrow R_0 \times X$ are replaced by a single multiplication $R_0 \leftarrow R_0 \times R_1$. Steps 5 and 8 are also included to maintain the relation between R_0 and R_1.

The power trace of the execution of the Square-and-multiply-always and Montgomery-ladder algorithms consists in a regular sequence of squarings followed by a multiplication: this does not leak anymore the bits of the exponent E.

- **Differential and correlation power analysis.** A more advanced attack called differential power analysis was initiated in (Kocher et al., 1999). This attack was later extended in (Brier et al., 2004) as a correlation power analysis (CPA).

This attack extracts hidden information in the power trace to identify the exact sequence

Algorithm 2 Montgomery-ladder (Joye and Yen, 2002)

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

of operations. The principle is to follow the data computed in the exponentiation algorithm: if we know the first bits $e_0, e_1, \ldots, e_i$ and the message X we can calculate $X_i = X^{E_i} \mod N$ where $E_i = (e_i, \ldots, e_0)_2$. Next we guess the next bit $e_{i+1} = 0$ or 1 and then compute the corresponding next value $X_{i+1,0} = X_i^2 \mod N$ or $X_{i+1,1} = X_i^2 \times X \mod N$ in the next iteration. The power trace is correlated to either $X_{i+1,0}$ or $X_{i+1,1}$, so the authors in (Kocher et al., 1999) use a differential of multiple power traces to accurately decide which value of e_{i+1} is the correct guess. The authors in (Brier et al., 2004) evaluate the covariance between the Hamming weight of data $X_{i+1,0}$ or $X_{i+1,1}$ and the power traces to decide which guess is correct.

Counter-measures. The basic approach to counteract differential and correlation power analysis is to inject randomization in the exponentiation. Specifically, the three main strategies to randomize the data are the following:

- Exponent blinding: this strategy given in (Coron, 1999) add to E a random multiple of $\phi(N) = (p-1)(q-1)$

$$E' = E + \beta \times \phi(N)$$

Then exponentiation $X^{E'} \mod N = X^E \mod N$ is the same. This leads to unpredictable values taken by R_0 during the exponentiation.
- Message blinding: The idea is to mask X and thus makes it impossible to predict anything regarding the power trace related to X. We choose a random value ρ for which we know $\rho^{E'}$ and we compute

$$X' = X \times \rho^{E'} \mod N$$

Then we compute $X'^E \mod N = X^E \times \rho^{E'E} \mod N = X^E \times \rho \mod N$ and then we divide by ρ to get $X^E \mod N$.
- Randomization of modular multiplication: the main approach is the use of randomized expression modulo N, instead of X we use $X' \equiv X \mod N$ with $X' \in \{0, \ldots, 2^w N\}$. The value X' can then be randomly generated as follows:

$$X' \leftarrow X' + \beta N,$$

with β random in $\{0, \ldots, 2^w\}$. The drawback of this approach is that each modular multiplication becomes more expensive, resulting in a penalty in space requirement, computation time and power consumption.

In the sequel we will present an alternative approach for the randomization in modular exponentiation in order to get a cheaper countermeasure against differential and correlation power analysis.

3 Modular multiplication

In this section we first review the two main methods, Montgomery (Montgomery, 1985) and Barrett (Barrett, 1987), for the computation of a multiplication modulo an integer N as the ones used in RSA cryptosystems. These methods differ in the way the product is reduced: the Montgomery method reduces the product by clearing the least significant bits and the method of Barrett reduces the product by clearing the most significant bits. We will then present a combination of these two approaches which can be used to randomize a multiplication modulo N.

3.1 Montgomery multiplication

Let N be an n-bit modulus which is stored on t computer words with $t = \lceil n/w \rceil$ and each computer word containing w bits. Let X be a t-word integer in $[0, N]$ and Y be an s-word integer such that $s \leq t$. Usually we have $s = t$ when we want to compute the product of two elements modulo N. But in the sequel we will need this version of the Montgomery multiplication with a multiplicand of word size $s < t$.

The word level form of Montgomery multiplication computes $R = X \times Y \times 2^{-sw} \mod N$ as follows: it sequentially multiply X by the s words of Y and add it to Z, and reduce Z by clearing its w least significant bits. For each word reduction: it computes Q of bit length w such that $Z+Q\times N$ have its w least significant bits equal to 0, it finally computes the exact division $(Z+Q\times N)/2^w$ which is equal to $Z\times 2^{-w} \mod N$. The main advantage of the Montgomery approach for modular multiplication is that it avoids a costly integer divisions. This approach is shown in Algorithm 3.

Algorithm 3 MontMul

Require: $2^{n-1} < N < 2^n$ the modulus of bit-length n, $X = (X_{t-1}, \ldots, X_0)_{2^w}$ an integer in $[0, N]$ and $Y = (Y_{s-1}, \ldots, Y_0)_{2^w}$ an integer in $[0, \ldots, 2^{ws}[$ and $N' = -N^{-1} \mod 2^w$
Ensure: $R = X \times Y \times 2^{-sw} \mod N$
1: **for** $i = 0$ **to** $s - 1$ **do**
2: $\quad Z \leftarrow Z + X \times Y_i$
3: $\quad Q \leftarrow N' \times Z \mod 2^w$
4: $\quad R \leftarrow (Z + Q \times N)/2^w$
5: **end for**
6: **if** $R \geq N$ **then**
7: $\quad R \leftarrow R - N$
8: **end if**
9: **return** R

3.2 Barrett multiplication

We consider a modulus N of bit length n and word length t. Let X be an integer of word length t and Y be an integer of word length $s \leq t$. Usually we have $s = t$ but in the sequel we will need this version of Barrett multiplication with $s < t$.

The approach of Barrett to perform the modular multiplication $(X \times Y) \mod N$ is the following: it sequentially multiply X the words of Y, add the result to Z, and reduce Z modulo N by clearing the most significant bits. For each word reduction it approximates the quotient Q of the division of Z by N and then computes the remainder $R = Z - QN$. This approach is shown in Algorithm 4.

$\hat{Q}$ in Algorithm 4 is a good approximation of the quotient $Q = \lfloor \frac{Z}{N} \rfloor$. The authors of (Knezevic et al., 2009) showed that

$$Q \geq \hat{Q} \geq Q - 1.$$

This means that Step 7 in Algorithm 4 returns the correct element $R = X \times Y \mod N$.

Algorithm 4 BarrettMul

Require: N a modulus of bit length n and word length t, $X = (X_{t-1}, \ldots, X_0)_{2^w}$ an integer in $[0, N]$ and $Y = (Y_{s-1}, \ldots, Y_0)_{2^w}$ an integer in $[0, \ldots, 2^s[$ and $N' = \lfloor 2^{n+w+3}/N \rfloor$
Ensure: $R = X \times Y \mod N$
1: **for** $i = s - 1$ **to** 0 **do**
2: $\quad Z \leftarrow Z \times 2^w + X \times Y_i$
3: $\quad \hat{Q} \leftarrow \lfloor \lfloor Z/2^{n-2} \rfloor N'/2^{w+5} \rfloor$
4: $\quad R \leftarrow Z - \hat{Q}N$
5: **end for**
6: **if** $R \geq N$ **then**
7: $\quad R \leftarrow R - N$
8: **end if**
9: **return** R

3.3 Combined Montgomery and Barrett multiplication

Now, we present our first contribution. We consider two n-bit integers X and Y stored on $t = \lceil n/w \rceil$ words. We combine Montgomery and Barrett multiplication in order to compute

$$R = X \times Y \times 2^{-sw} \mod N$$

with an arbitrary $s \in [0, t]$. For this, we split the integer X into two parts $X = X_0 + 2^{sw} X_1$. We then perform

$$\begin{aligned} Z_0 &= \text{MontMul}(Y, X_0) = Y \times X_0 \times 2^{-sw} \mod N, \\ Z_1 &= \text{BarrettMul}(Y, X_1) = Y \times X_1 \mod N. \end{aligned}$$

We then obtain the required result $R = X \times Y \times 2^{-sw} \mod N$ as follows

$$\begin{aligned} Z_0 + Z_1 &= Y \times X_0 \times 2^{-sw} + Y \times X_1 \mod N \\ &= Y \times (X_0 + 2^s X_1) 2^{-sw} \mod N \\ &= X \times Y \times 2^{-sw} \mod N. \end{aligned}$$

Algorithm 5 gives the details of this approach.

3.4 Complexity comparison

We use the word level Montgomery and Barrett algorithms reviewed in Subsection 3.1 and 3.2. We study the complexity in terms of word additions (Add) and word multiplications (Mul). The complexity of Barret and Montgomery multiplications (without the final subtraction by N) for a t-word X and an s-word Y are as follows

- Barret: $\#Mul = s(t+1)$ and $\#Add = s(3t+2)$.
- Montgomery: $\#Mul = s(t+1)$ and $\#Add = s(3t+2)$.

Algorithm 5 CombMontBarrettMul

Require: $2^{n-1} < N < 2^n$ the modulus of bit-length n and word length t, $X = (X_{t-1}, \ldots, X_0)_{2^w}$ and $Y = Y_{n-1}, \ldots, Y_0)_{2^w}$ two integers in $[0, N[$ and a split $s \in [0, t]$
Ensure: $Z = X \times Y \times 2^{-sw} \mod N$
1: Split. $X = X_0 + 2^{sw} X_1$
2: $Z_0 \leftarrow \text{MontMul}(Y, X_0)$
3: $Z_1 \leftarrow \text{BarrettMul}(Y, X_1)$
4: $Z \leftarrow Z_0 + Z_1$
5: **if** $Z > N$ **then**
6: $\quad Z \leftarrow Z - N$
7: **end if**
8: **return** Z

This leads to the following complexity of the CombinedMontMul algorithm (including the final subtraction by N):

$$\begin{aligned} \#Mul &= s(t+1) + (t-s)(t+1) = t^2 + t \\ \#Add &= s(3t+2) + (t-s)(3t+2) + t \\ &= 3t^2 + 3t + t \end{aligned}$$

This leads to the complexity shown in Table 1 for the multiplication of the t words integers X and Y with the three approaches: Montgomery, Barrett and combined Montgomery Barrett. These complexity includes the final subtraction to get $Z < N$.

Table 1 – Complexity comparison of Montgomery, Barrett and combined Montgomery Barrett

	#Add	#Mul	Total
Montgomery	$t^2 + t$	$3t^2 + 2t + t$	$4t^2 + 3t$
Barrett	$t^2 + t$	$3t^2 + 2t + t$	$4t^2 + 3t$
Combined Montgomery and Barret	$t^2 + t$	$3t^2 + t$	$4t^2 + 3t$

Since the complexity of the word level form of CombMontBarretMul is the same as Montgomery and Barrett, we can take advantage of the algorithm to randomize modular exponentiation. We will study this strategy in the following section.

4 Randomized exponentiation

We present in this section two randomized exponentiations which uses CombMontBarretMul for modular multiplications.

4.1 Randomized Montgomery-ladder

We first consider the Montgomery-ladder (Algorithm 2). We randomize this exponentiation by using CombMontBarrettMul for each modular multiplication. Before each multiplication the splitting s for CombMontBarretMul is randomly generated. The effect is that in the Montgomery-ladder exponentiation we have $\widetilde{R_0} = R_0 2^{\gamma w} \mod N$ and $\widetilde{R_1} = R_1 2^{\gamma w} \mod N$ such that the integer γ evolves randomly. The integer s is randomly chosen in order to keep γ in the interval $[t/3, 2t/3]$. This approach is shown in Algorithm 6.

Algorithm 6 Randomized-Montgomery-ladder

Require: An RSA modulus N of bit length n, an integer $X \in \{0, \ldots, N-1\}$ and $E = (e_{\ell-1}, \ldots, e_0)_2$
Ensure: $R_0 = X^E \mod N$
1: $s_\ell \leftarrow \text{rand}(t/3, 2t/3)$
2: $\gamma_\ell \leftarrow s_\ell w$
3: $\widetilde{R}_0 \leftarrow 2^{s_\ell w}$
4: $\widetilde{R}_1 \leftarrow \text{BarrettMul}(X, 2^{s_\ell w})$
5: **for** i **from** $\ell - 1$ **to** 0 **do**
6: $\quad \gamma_i \leftarrow \text{rand}(t/3, 2t/3)$
7: $\quad s_i \leftarrow 2\gamma_{i+1} - \gamma_i$
8: $\quad$ **if** $e_i = 1$ **then**
9: $\quad\quad \widetilde{R}_0 \leftarrow \text{CombMontBarrettMul}(\widetilde{R}_0, \widetilde{R}_1, s_i)$
10: $\quad\quad \widetilde{R}_1 \leftarrow \text{CombMontBarrettMul}(\widetilde{R}_1, \widetilde{R}_1, s_i)$
11: $\quad$ **else**
12: $\quad\quad \widetilde{R}_1 \leftarrow \text{CombMontBarrettMul}(\widetilde{R}_0, \widetilde{R}_1, s_i)$
13: $\quad\quad \widetilde{R}_0 \leftarrow \text{CombMontBarrettMul}(\widetilde{R}_0, \widetilde{R}_0, s_i)$
14: $\quad$ **end if**
15: **end for**
16: $\widetilde{R}_0 \leftarrow \text{CombMontBarrettMul}(R_0, 1, \gamma_0)$
17: **return** $\widetilde{R}_0$

Validity of Algorithm 6. Let $\widetilde{R}_0^{(i)}$ and $\widetilde{R}_1^{(i)}$ be the values of $\widetilde{R}_0$ and $\widetilde{R}_1$ after the i-th iteration. The integer γ_i in Algorithm 6 satisfies $\widetilde{R}_0^{(i)} = R_0 \times 2^{\gamma_i w} \mod N$ and $\widetilde{R}_1^{(i)} = R_1^{(i)} \times 2^{\gamma_i w} \mod N$ where $R_0^{(i)}$ and $R_1^{(i)}$ are the values of R_0 and R_1 after the i-th loop of the non-randomized Montgomery-ladder exponentiation (i.e. Algorithm 2). When $e_i = 1$ we have

$$\begin{aligned} \widetilde{R}_0^{(i)} &= (R_0^{(i+1)} \times 2^{\gamma_{i+1} w}) \\ &\quad \times (R_1^{(i+1)} \times 2^{\gamma_{i+1} w}) \times 2^{-s_i w} \mod N \\ &= R_0^{(i+1)} \times R_1^{(i+1)} \times 2^{2\gamma_{i+1} w - s_i w} \mod N \\ &= R_0^{(i)} \times 2^{2\gamma_{i+1} w - s_i w} \mod N \end{aligned}$$

which means that the instruction $s_i = 2\gamma_{i+1} - \gamma_i$ in Step 7 is correct. Let us now check that s_i is always in $[0, t]$. We have $\gamma_{i+1}, \gamma_i \in [t/3, 2t/3]$ then $2t/3 \leq 2\gamma_{i+1} \leq 4t/3$ and then

$$0 \leq 2\gamma_{i+1} - \gamma_i \leq 3t/3$$

which implies that $s_i = 2\gamma_{i+1} - \gamma_i$ is in $[0, t]$.

The main advantage of the proposed randomization is that it is free of computation. But its main drawback is that the level of randomization is not so important since we have, at each loop iteration, only $t/3$ possible values for $\widetilde{R}_0$ and $\widetilde{R}_1$. In practice this means that we have to combine the proposed randomization with the classical ones reviewed in Section 2. When these strategies are combined to the one proposed here we can obtain any level of randomization with a reduced cost.

4.2 Randomized Right-to-left Square-and-multiply-always

Now, we extend the randomization technique presented in Subsection 4.1, to the Right-to-left Square-and-multiply-always exponentiation. This algorithm is reviewed in Algorithm 7. Our goal is to get a regular exponentiation with a large level of randomization.

Algorithm 7 Right-to-left Square-and-multiply-always

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

We propose to randomize Algorithm 7 as follows:

- We randomize Z with the strategy of Subsection 4.1. We have $\widetilde{Z}_i = Z^{(i)} \times 2^{\gamma_i w} \mod N$ such that γ_i is in $[t/3, 2t/3]$ all along the exponentiation.
- We randomize R_0 and R_1 as follows: before each multiplication $R_0 \times Z \mod N$ and $R_1 \times Z \mod N$ we randomly choose $s'_i \in [0, t]$. Then we perform the multiplications $R_0 \times Z \mod N$ or $R_1 \times Z \mod N$ with CombMontBarrettMul with split s'_i.

This randomized version of Algorithm 7 is shown in Algorithm 8.

Algorithm 8 Randomized Right-to-left Square-and-multiply-always

Require: An RSA modulus N of bit length n, an integer $X \in \{0, \ldots, N-1\}$ and $E = (e_{\ell-1}, \ldots, e_0)_2$
Ensure: $R_0 = X^E \mod N$
1: $s_{-1} \leftarrow \text{rand}(t/3, 2t/3)$
2: $\widetilde{R}_0 \leftarrow 1, \widetilde{R}_1 \leftarrow 1, \widetilde{Z} \leftarrow Barrett(X, 2^{s_{-1}w})$
3: $\gamma_{-1} \leftarrow s_{-1}, \gamma'_{-1} \leftarrow 0$
4: **for** i **from** 0 **to** $\ell - 1$ **do**
5: $s'_i \leftarrow \text{rand}(0, t)$
6: **if** $e_i = 0$ **then**
7: $\widetilde{R_0} \leftarrow \text{CombMontBarrettMul}(\widetilde{R_0}, \widetilde{Z}, s'_i)$
8: $\gamma'_i \leftarrow \gamma'_{i-1} + (\gamma_{i-1} - s'_i)$
9: **else**
10: $\widetilde{R_1} \leftarrow \text{CombMontBarrettMul}(\widetilde{R_1}, \widetilde{Z}, s'_i)$
11: $\gamma'_i \leftarrow \gamma'_{i-1}$
12: **end if**
13: $\gamma_i \leftarrow \text{rand}(t/3, 2t/3)$
14: $s_i \leftarrow 2\gamma_{i-1} - \gamma_i$
15: $\widetilde{Z} \leftarrow \text{CombMontBarrettMul}(\widetilde{Z}, \widetilde{Z}, s)$
16: **end for**
17: **return** $\widetilde{R}_1$

Validity of Algorithm 8. The way γ and $\widetilde{Z}$ evolve in Algorithm 8 is well known since it was studied in Subsection 4.1. Consequently, we have $\widetilde{Z}^{(i)} = Z^{(i)} \times 2^{\gamma_i w} \mod N$ where $Z^{(i)}$ is the value of Z after the i-th loop in the non-randomized Algorithm 7 and γ_i is a random element in $[t/3, 2t/3]$.

Now we consider $\widetilde{R}_1^{(i)}$ and γ'_i. We have:

$$\widetilde{R}_1^{(i)} = R_1^{(i)} \times 2^{\gamma'_i w} \mod N.$$

The value γ'_{i+1} is expressed in terms of γ'_i, γ_i and s'_{i+1} as follows:

- If $e_{i+1} = 1$ we have

$$\begin{aligned}\widetilde{R}_1^{(i+1)} &= \widetilde{R}_1^{(i)} \times \widetilde{Z} \times 2^{-s'_{i+1}w} \mod N \\ &= R_1^{(i)} \times 2^{\gamma'_i w} \times Z \times 2^{\gamma_{i-1}w} \\ &\quad \times 2^{-s'_{i+1}w} \mod N \\ &= R_1^{(i+1)} \times 2^{\gamma'_i w + \gamma_i w - s'_{i+1}w} \mod N\end{aligned}$$

which means that $\gamma'_{i+1} = \gamma'_i + \gamma_i - s'_{i+1}$.

- If $e_{i+1} = 0$ we have

$$\begin{aligned} \widetilde{R}_1^{(i+1)} &= \widetilde{R}_1^{(i)} \\ &= R_1^{(i+1)} \times 2^{\gamma'_i} \mod N \end{aligned}$$

which gives $\gamma'_{i+1} = \gamma'_i$.

So if we set

$$\delta_{i+1} = \begin{cases} \gamma_i - s'_{i+1} & \text{if } e_{i+1} = 1 \\ 0 & \text{if } e_{i+1} = 0 \end{cases}$$

we have $\gamma'_{i+1} = \gamma'_i + \delta_{i+1}$, which means that γ'_i consists in random walk of step sizes δ_i for $i = 1, \ldots, \ell - 1$. The absolute value of these step sizes δ_i are bounded by $2t/3$: indeed since s'_i always satisfies $0 \leq s'_i \leq t$ we always have

$$-2t/3 = t/3 - t \leq \underbrace{\gamma_{i-1} - s'_i}_{\delta_i} \leq 2t/3 - 0 = 2t/3.$$

Such random walk can get away from 0 as i increases. This is interesting since this enlarges the possible values for $\widetilde{R}_1$, i.e., this enlarges the level of randomization.

But, on the other hand, this random walk induces a problem we need to tackle: the final value $\gamma'_{\ell-1}$ can be a quite large integer and this might render difficult to extract $R_1^{(\ell)}$ from $\widetilde{R}_1^{(\ell)}$.

We propose a strategy which controls the way γ'_i evolves, in order to have the final value $\gamma'_{\ell-1} = 0$. The following lemma provides the conditions to reach this goal.

Lemma 1. *Let E be an ℓ-bit exponent and let h_E be his hamming weight. We denote $\{i_0, \ldots, i_{h_E}\}$ the set of indexes i such that $e_i \neq 0$. In Algorithm 8 we can choose the integers s'_i such that:*

$$\begin{aligned} \delta_i &> 0 \text{ for the first } h_E/2 \text{ bits } e_i \neq 0, \\ \delta_i &< 0 \text{ for the last } h_E/2 \text{ bits } e_i \neq 0, \end{aligned}$$

and such that we have

$$0 \leq \gamma'_{i_j} < (h_E - j)2t/3 \text{ for } j = 1, \ldots, h_E$$

for all $i_j \in \{i_0, \ldots, i_{h_E}\}$.

Proof. There are the following two phases in the evolution of γ_i:

- *Ascending phase.* This phase corresponds to loops i_j for $j = 1, \ldots, h_E/2$ where chose s'_{i_j} such that $\delta_{i_j} \in [0, 2t/3]$. Consequently, after loop i_j we have made j such step sizes and then the exponent of $\widetilde{R}_1^{(i_j)}$ satisfies:

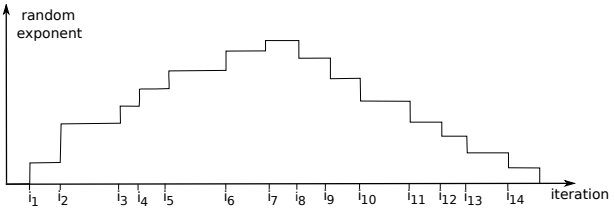
$$0 \leq \gamma'_{i_j} \leq j2t/3 \leq (h_E - j)2t/3.$$

- *Descending phase.* This phase corresponds to loops i_j for $j = h_E/2 + 1, \ldots, h_E$. If we have $\gamma'_{i_j} \leq (h_E - j)2t/3$ then there exists a step size $\delta_{i_j} \in [-2t/3, 0]$ which yields $\gamma'_{i_{j+1}} < (h_E - j)2t/3 - t/3$. Such δ_{i_j} can be obtained by choosing properly s'_{i_j} and $\gamma_{i_j - 1}$

□

Figure 2 illustrates the way γ'_i increase and then decrease towards 0.

Figure 2 – Evolution of the exponent γ based on Lemma 1



Level of randomization. We now focus on the level of randomization injected in $\widetilde{R}_1^{(i)}$ in Algorithm 8 under the restriction of Lemma 1. The number of possible values for $\widetilde{R}_1^{(i)}$ depends on i and on the Hamming weight of E. The following lemma establishes the average case of this level of randomization.

Lemma 2. *Let $\widetilde{R}_1^{(i)} = R_1^{(i)} 2^{\gamma'_i} \mod N$ be the value of $\widetilde{R}_1$ at the end of the i-th loop in Algorithm 8. Then γ'_i satisfies the following:*

i) For $i < \ell/2$ we have, in average, $\gamma'_i \in [0, (2t/3) \times (i/2)]$

ii) For $i > \ell/2$ we have, in average, $\gamma'_i \in [0, (2t/3) \times (\ell - i)/2]$

Proof. We prove each case i) and ii) separately:

i) Let α_i be the number of $e_j \neq 0$ with $j \leq i$. Then the maximal value for γ'_i which can be reached at loop i is $(2t/3) \times \alpha_i$ which corresponds to a walk of α_i steps of size $2t/3$. In average, we have $\alpha_i = i/2$ such non-zero e_j with $j \leq i$ and this leads to the assertion i) of the lemma.

ii) For $i > \ell/2$ we consider that we make backward steps starting from the end where we start at $\gamma'_\ell = 0$. We make $\ell - i$ steps backwards, and only $(\ell - i)/2$ of them have step size $\neq 0$, the maximal distance walked is $(2t/3) \times (\ell - i)/2$ and this leads to assertion ii) of the lemma.

□

Table 2 – Level of randomization for practical sizes of N, $w = 32$ and several loop iterations

	Loop iterations i					
	50	100	500	1000	1500	2000
RSA 4096	2^{10}	2^{11}	$2^{13.4}$	$2^{14.4}$	2^{15}	$2^{15.4}$
RSA 3072	2^{10}	2^{11}	$2^{13.4}$	$2^{14.4}$	2^{15}	
RSA 2048	2^{10}	2^{11}	$2^{13.4}$	$2^{14.4}$		

The previous lemma shows that the level of randomization for $\widetilde{R}_1^{(i)}$ becomes large as soon as we advance in the exponentiation. Table 2 shows the level of randomization obtained for different size of RSA modulus and different values for i.

5 Conclusion

In this paper we focused on counter-measures based on randomization of RSA exponentiation against side channel analysis. We proposed to perform modular multiplications with a combination of Montgomery and Barrett multiplications. This algorithm provides a way to randomize modular multiplication by setting the splitting value s at random. We then provided a modified version of two regular algorithm, i.e., Montgomery-ladder and Square-and-multiply-always exponentiation. We analyzed the algorithms which showed that the proposed approach for randomization is interesting since it does not induce any penalty in terms of performance.

References

Barrett, P. (1987). Implementing the Rivest Shamir and Adleman Public Key Encryption Algorithm on a Standard Digital Signal Processor. In *CRYPTO '86*, pages 311–323. Springer-Verlag.

Brier, E., Clavier, C., and Olivier, F. (2004). Correlation Power Analysis with a Leakage Model. In *CHES 2004*, volume 3156 of *LNCS*, pages 16–29. Springer.

Coron, J.-S. (1999). Resistance against Differential Power Analysis for Elliptic Curve Cryptosystems. In *CHES*, pages 292–302.

Diffie, W. and Hellman, M. (1976). New directions in cryptography. *IEEE Trans. Information Theory*, 22(6):644–654.

Joye, M. and Yen, S. (2002). The Montgomery Powering Ladder. In *CHES 2002*, volume 2523 of *LNCS*, pages 291–302. Springer.

Knezevic, M., Vercauteren, F., and Verbauwhede, I. (2009). Speeding Up Barrett and Montgomery Modular Multiplications.

Kocher, P. (1996). 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.

Kocher, P., Jaffe, J., Jun, B., and Rohatgi, P. (2011). Introduction to differential power analysis. *J. Cryptographic Engineering*, 1(1):5–27.

Kocher, P. C., Jaffe, J., and Jun, B. (1999). Differential Power Analysis. In *Advances in Cryptology, CRYPTO'99*, volume 1666 of *LNCS*, pages 388–397. Springer.

Mangard, S. (2003). Exploiting Radiated Emissions - EM Attacks on Cryptographic ICs. In *Austrochip 2003, Linz, Austria, October 1st*, pages 13–16.

Montgomery, P. (1985). Modular Multiplication Without Trial Division. *Math. Computation*, 44:519–521.

Rivest, R., Shamir, A., and Adleman, L. (1978). A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21:120–126.