Side Channel Counter-measures based on Randomized AMNS Modular Multiplication

Christophe Negre[1,2]
[1]*DALI, Université de Perpignan, France*
[2]*LIRMM, Université de Montpellier and CNRS, France*

Keywords: Exponentiation, DSA, Scalar Multiplication, ECC, Randomization, AMNS, Side Channel Analysis, Counter-measure.

Abstract: The paper presents counter-measures based on dynamic randomization against side channel analysis like differential and correlation power analysis. The building block of the proposed counter-measure is a randomization of the modular multiplication in AMNS for a prime p. We use this randomized modular multiplication to inject randomization during the whole computation in DSA exponentiation and Co-Z elliptic curve scalar multiplication. We analyze the level of randomization injected and, through implementations results, we evaluate the penalty in terms of performance of the proposed counter-measures.

1 INTRODUCTION

Modern cryptographic protocols like Digital Signature Algorithm (DSA), Elliptic Cryptography (ECC) or post-quantum SIDH (Jao and Feo, 2011) necessitate to perform hundreds of multiplications modulo a prime integer p. Such modular multiplications involve quite large integers : 256 to 500 bits for ECC and SIDH and 2000 bits to 8000 bits for DSA. Computing a multiplication modulo p consists in computing a product of integers $C = A \times B$ which produces C of size p^2, the product C is reduced to an integer R of size p by subtracting a multiple of p which clears out parts of the bits of C. Indeed, in Barrett approach (Barrett, 1987) computing $R = C - pQ$ clears out the most significant bits of C, whereas in Montgomery approach (Montgomery, 1985) computing $R = C - pQ$ clears out the least significant bit, in this latter case the output is $R/\phi \equiv AB\phi^{-1} \mod p$ where ϕ is a power 2.

Alternative number system can be used to improve such modular multiplication. Indeed, in the Adapted Modular Number System (Bajard et al., 2004) for a prime p, the elements are represented with a larger radix γ modulo p than the usual 2^w-radix for multi-precision integer representation. The initial goal of the ANMNS was to simplify carry propagation in integer multiplications and reductions. Recently in (Didier et al., 2019), it was shown that the Montgomery-like approach for modular multiplication in AMNS was competitive compared to state of the art approaches.

Cryptographic protocols can be threaten by side channel analysis when they are executed on an embedded device. Indeed when monitoring either power consumption (Kocher et al., 1999), electronic emanation or computation time, it is possible to recover part of the secret data involved in the computation. For example Differential Power Analysis (DPA) (Kocher et al., 1999) or Correlation Power Analysis (CPA) (Brier et al., 2004) guess some secret bits, and they check if this guess leads to data correlated to leaked out power consumption. The main strategy to counteract these attacks consists in randomizing the data involved in cryptographic computation, which reduces the correlation between the secret data and the power consumption or electronic emanation. The main methods for randomizing data (i.e. the integer modulo p or the exponent in DSA) consists in masking data with additive mask (Tunstall and Joye, 2010; Clavier et al., 2010) or multiplicative mask, but this induces additional computations and penalty in terms of performance. In 2016, the method proposed in (Lesavourey et al., 2016) combines Montgomery and Barrett modular multiplication to induces multiplicative mask of the form 2^t with random t. The interest of this approach is that it is almost free of computation, and it produces a mask which randomly changes during the whole computation.

Negre, C.
Side Channel Counter-measures based on Randomized AMNS Modular Multiplication.
DOI: 10.5220/0010599706110619
In *Proceedings of the 18th International Conference on Security and Cryptography (SECRYPT 2021)*, pages 611-619
ISBN: 978-989-758-524-1

Contributions. In this paper we extend the approach presented in (Lesavourey et al., 2016) to the case of modular arithmetic in AMNS for a prime p. We provide a randomized version of both Montgomery-like (resp. Barrett-like) multiplication in AMNS producing multiplicative mask $\phi^{-1}\gamma^{-s}$ (resp. γ^{-s}) for a random s. Afterwards we present modular exponentiation for DSA and scalar multiplication on elliptic curve both using the proposed randomized multiplication to produce multiplicative random mask during the whole computation. We evaluate the level of randomization produced by the proposed approach and also present implementation results.

Organization of the Paper. In Section 2 we review Barrett-like and Montgomery-like modular multiplication in an AMNS. In Section 3 we present a strategy to randomized Barrett-like and Montgomery-like modular multiplication in AMNS. In Section 4 and 5 we adapt the Montgomery ladder for modular exponentation and scalar multiplication in order to use the proposed randomized AMNS multiplications. Finally, in Section 6, we give some concluding remarks.

2 ARITHMETIC IN ADAPTED MODULAR NUMBER SYSTEM

In this section we review the Adapted Modular Number System and related algorithms for multiplication modulo a prime integer p.

2.1 Definition

Arithmetic of integers are generally based on radix representation : in radix β an integer A is expressed as $A=\sum_{i=0}^{n-1} a_i\beta^i$ where $0\le a_i<\beta$. On computers the radix β is generally chosen as $\beta=2^w$ where w is the word size of the computer. In (Bajard et al., 2004) the authors introduced the Adapted Modular Number System (AMNS) to represent integers modulo p (cf. Definition 1). This system somehow extends the radix representation to a larger set of radix $\gamma\in\{0,1,\ldots,p-1\}$.

Definition 1 (AMNS (Bajard et al., 2004)). *An Adapted Modular Number System $\mathcal{B}=(p,n,\rho,\gamma,\lambda)$ is such that*

i) *p is prime integer.*

ii) *n is the number of coefficients of the system.*

iii) ρ *the upper bound of the absolute value of the coefficients.*

iv) γ *is the radix of the system and* λ *is a small integer such that*
$$\gamma^n=\lambda \bmod p. \tag{1}$$

v) *Any integer A modulo p can be written as*
$$A\equiv\sum_{i=0}^{n-1} a_i\gamma^i \mod p \text{ with } |a_i|<\rho.$$

The elements of an AMNS are seen as degree $n-1$ polynomials $A(X)=\sum_{i=0}^{n-1} a_iX^i$ in X with coefficients smaller that ρ. To get their integer expression we have to evaluate $A(X)$ at γ modulo p.

2.2 AMNS Lattice and Short Polynomial

Given an AMNS $\mathcal{B}=(p,n,\gamma,\lambda)$ the authors in (Nègre and Plantard, 2008) define the following rank n lattice
$$\begin{array}{rcl}\mathcal{L}_{p,n,\rho,\gamma,\lambda} & = & \{V(X)\in\mathbb{Z}[X] \text{ s.t. } \deg V(X)<n \\ & & \text{and } V(\gamma)\equiv 0 \mod p\}\end{array}.$$

A lattice can be seen as integer linear combinations of vector in $\mathbb{Z}^n$. Below we provide a basis of the lattice $\mathcal{L}_{p,n,\rho,\gamma,\lambda}$:
$$\mathbf{B}=\begin{pmatrix} p & 0 & 0 & 0 & \ldots & 0\\ -\gamma & 1 & 0 & 0 & \ldots & 0\\ -\gamma^2 & 0 & 1 & 0 & \ldots & 0\\ \vdots & & & \ddots & & \vdots\\ -\gamma^{n-2} & 0 & 0 & \ldots & 1 & 0\\ -\gamma^{n-1} & 0 & 0 & \ldots & 0 & 1\end{pmatrix}\begin{array}{l}\leftarrow p\\ \leftarrow X-\gamma\\ \leftarrow X^2-\gamma^2\\ \vdots\\ \leftarrow X^{n-2}-\gamma^{n-2}\\ \leftarrow X^{n-1}-\gamma^{n-1}\end{array}$$

The above basis tells us that the volume of the lattice is
$$\det(\mathbf{B})=p.$$

With Minkowsky's inequality, the authors (Nègre and Plantard, 2008) then obtained a short non-zero vector (or polynomial) in $\mathcal{L}_{p,n,\rho,\gamma,\lambda}$ by applying a reduction algorithm such as LLL or BKZ:
$$M=m_0+m_1X+\cdots+m_{n-1}X^{n-1} \tag{2}$$
such that $\|M\|_\infty\cong\sqrt[n]{\det(\mathcal{L}_{p,n,\gamma,\lambda})}=\sqrt[n]{p}$ and $M(\gamma)=0$ mod p.

2.3 Montgomery-like Multiplication in AMNS

The condition *iv)* on γ in Definition 1 is meant to ease the multiplication modulo p in the AMNS. Indeed, let us call $E(X)$ the polynomial $X^n-\lambda$: this means that, from condition *iv)* in Definition 1, γ is a root of the polynomial $E(X)$ in $\mathbb{Z}/p\mathbb{Z}$. As described in (Bajard et al., 2004) a multiplication of two elements in

AMNS consists of a polynomial multiplication modulo $E(X) = X^n - \lambda$

$$C(X) = A(X) \times B(X) \mod E(X)$$

and a reduction of the coefficients. Since $\|A\|\infty, \|B\|_\infty < \rho$, the coefficients of C lie in the interval $] - n\rho^2\lambda, n\rho^2\lambda[$, they must be reduced such that they have absolute value smaller than ρ.

A first method to reduce the coefficients was proposed in (Nègre and Plantard, 2008), this approach uses the short polynomial $M(X)$ of (2) which satisfies $M(\gamma) = 0 \mod p$ and $\|M\|_\infty$ is small. This method is depicted in Algorithm 1: it computes Q such that the lower parts of the coefficient $(C + Q \times M) \bmod E$ are all zero (or equivalently are equal to 0 modulo $\phi = 2^k$). But adding $Q \times M$ modulo E does not change the value modulo p since $M(\gamma) = 0 \mod p$ and $E(\gamma) = 0 \bmod p$. At the end the polynomial

$$R = (C + Q \times M \bmod E)/\phi$$

evaluated at γ leads to

$$R(\gamma) = C(\gamma)\phi^{-1} \bmod p = A(\gamma) \times B(\gamma)\phi^{-1} \bmod p.$$

Algorithm 1: AMNS_MonMul.

```
Require: $A, B \in \mathcal{B} = AMNS(p, n, \gamma, \lambda, \rho)$ with $E = X^n - \lambda$ $M$ such that $M(\gamma) \equiv 0 \pmod{p}$ an integer $\phi$ and $M' = -M^{-1} \bmod (E, \phi)$
Ensure: $R$ such that $R(\gamma) = A(\gamma)B(\gamma)\phi^{-1} \bmod p$
  1: $C \leftarrow A \times B \bmod E$
  2: $Q \leftarrow C \times M' \bmod (E, \phi)$
  3: $R \leftarrow (C + Q \times M \bmod E)/\phi$
```

The authors in (Nègre and Plantard, 2008) showed that the above algorithm outputs R in the AMNS (i.e. with $\|R\|_\infty < \rho$) under the following condition

$$\rho > 2|\lambda|n\sigma \quad \text{and} \quad \phi > 2|\lambda|n\rho$$

2.4 Barrett-like Multiplication in AMNS

A second approach to perform the reduction of the coefficients in an AMNS multiplication was proposed in (Plantard, 2005). This method adapts the Barrett method (Barrett, 1987) to the case of multiplication in an AMNS. This approach use the short polynomial M defined in (2) to reduce the upper part of the coefficients of the following polynomial:

$$C = A(X) \times B(X) \mod E(X).$$

The method of (Plantard, 2005) is shown in Algorithm 2, this method computes a polynomial Q such that in $C - ((Q \times M) \mod E)$ the most significant bits of the coefficients are set to zero. In the sequel we will assume $\beta = 2$, indeed, in this case a division by power of β is just a right shift.

Algorithm 2: AMNS_BarMul.

```
Require: $A(X), B(X)$ two elements in an AMNS $(p, n, \rho, \gamma, \lambda)$, a radix $\beta$, a polynomial $M$ such that $M(\gamma) = 0 \mod p$ and $V = \lfloor (M^{-1} \mod E) \times \beta^{2k} \rceil$
Ensure: $R$ such that $R(\gamma) = A(\gamma) \times B(\gamma) \mod p$.
  1: $C \leftarrow (A \times B) \mod E$
  2: $U \leftarrow \lfloor C/\beta^{k-1} \rceil$
  3: $W \leftarrow (U \times V) \mod E$
  4: $Q \leftarrow \lfloor W/\beta^{k+1} \rceil$
  5: $R \leftarrow C - ((Q \times M) \mod E)$
  6: return $(R)$
```

If we assume

$$\rho > 2n^2\lambda^2\|M\|_\infty \text{ and } \|A\|_\infty, \|B\|_\infty < \rho$$

then, the polynomial R output by Algorithm 2 satisfies $\|R\|_\infty < \rho$ and $R(\gamma) = A(\gamma) \times B(\gamma) \mod p$.

3 RANDOMIZED MODULAR MULTIPLICATION IN AMNS

In this section we present a randomization of modular multiplication in an AMNS$(p, n, \rho, \gamma, \lambda)$. We will use it later to randomise modular exponentiation and scalar multiplication. For the remaining of the paper, we assume that:

$$\lambda = 2. \tag{3}$$

3.1 Randomized Polynomial Multiplication Modulo E

We propose to change Step 1 in AMNS_MonMul and AMNS_BarMul with

$$C \leftarrow 2 \times (A \times B)/X^s \bmod E. \tag{4}$$

for $s \in \{0, \ldots, n-1\}$. Let us first see how to compute C in (4). We consider the product $U(X) = A(X) \times B(X)$, then we rewrite $2 \times U(X)$ as follows:

$$2 \times U(X) = \underbrace{(\sum_{i=0}^{s-1} 2u_iX^i)}_{U_0} + \underbrace{(\sum_{i=s}^{n+s-1} 2u_iX^i)}_{U_1} + \underbrace{(\sum_{i=n+s}^{2n-1} 2u_iX^i)}_{U_2}.$$

Then using (3), we have $2 \equiv X^n \mod E(X)$, we can replace each 2 with X^n in U_0 and we can also replace each X^n with 2 in U_2. We get:

$$\begin{aligned} 2 \times U(X) \quad &\equiv (\textstyle\sum_{i=s}^{n-1}(2u_i + 4u_{i+n})X^i) \\ &\quad + (\textstyle\sum_{i=n}^{n+s-1}(2u_i + u_{i-n})X^i) \mod E \end{aligned}$$

Which leads to the following

$$(2U(X))X^{-s} \bmod E = (\textstyle\sum_{i=0}^{n-s-1}(2u_{i+s} + 4u_{i+n+s})X^i) + (\textstyle\sum_{i=n-s}^{n-1}(2u_{i+s} + u_{i+s-n})X^i) \tag{5}$$

Complexity of Randomized Multiplication. The costs of non-randomized and randomized multiplication are show in Table 1. The shifts are due to the multiplications by 2 or 4 in a reduction modulo E.

Table 1: Complexity of randomized and non-randomized multiplication mod E.

Operation	# mul.	# add.	# shifts
$A \times B \bmod E$	n^2	n^2	n
$(A \times B)/X^s \bmod E$	n^2	n^2	$2n - s$

We can notice that a randomized multiplication has a complexity close to a non-randomized multiplication: we just have a penalty of $n - s$ shifts.

3.2 Randomized AMNS-Montgomery and AMNS-Barret Multiplication

We can use this randomized multiplication modulo $E(X)$ to randomize AMNS_MonMul multiplication. To reach this goal we replace the first step of AMNS_MonMul with $C \leftarrow (A \times B)/X^s \bmod E$. We show in Algorithm 3 the resulting randomized AMNS_MonMul. The proposed modification changes the output of the algorithm. The output of Rd_AMNS_MonMul is:

$$\begin{aligned} R &= C + Q \times M \bmod E \\ &= ((A \times B) + W \times E)/X^s + Q \times M + W' \times E \end{aligned}$$

where in the last expression $W(X)$ and $W'(X)$ are due to the reduction by $E(X)$. If we evaluate the above expression of R at γ the terms $Q \times M$, $W \times E$, and $W' \times E$ vanish since $M(\gamma) = 0$ and $E(\gamma) = 0$. This leads to the following:

$$R(\gamma) = A(\gamma)B(\gamma)\phi^{-1}\gamma^{-s} \mod p.$$

Algorithm 3: Rd_AMNS_MonMul.

```
Require: $A, B \in \mathcal{B} = AMNS(p, n, \gamma, \lambda, \rho)$ with $E = X^n - \lambda$ and $\lambda = 2$, $s$ a randomizing integer, $M$ a
    polynomial such that $M(\gamma) \equiv 0 \pmod{p}$, an integer $\phi$ and $M' = -M^{-1} \bmod (E, \phi)$
Ensure: $R$ such that $R(\gamma) = A(\gamma)B(\gamma)\phi^{-1}\gamma^{-s} \bmod p$
 1: $C \leftarrow (A \times B)/X^s \bmod E$
 2: $Q \leftarrow C \times M' \bmod (E, \phi)$
 3: $R \leftarrow (C + Q \times M \bmod E)/\phi$
 4: return $R$
```

We can do the exact same modification in AMNS_BarMul. The only change is on the output of the algorithm which in this case produce a polynomial $R(X)$ satisfying:

$$R(\gamma) = A(\gamma)B(\gamma)\gamma^{-s} \bmod p$$

The resultatin algorithms are shown in Algorithm 3 and 4.

Algorithm 4: Rd_AMNS_BarMul.

```
Require: $A, B \in \mathcal{B} = AMNS(p, n, \gamma, \lambda, \rho)$ with $E = X^n - \lambda$ and $\lambda = 2$, $s$ a randomizing integer, $M$ such
    that $M(\gamma) = 0 \mod p$, $V = \lfloor (M^{-1} \mod E) \times \beta^{2k} \rceil$.
Ensure: $R$ such that $R(\gamma) = A(\gamma) \times B(\gamma)\gamma^{-s} \mod p$
 1: $C \leftarrow (A \times B)/X^s \mod E$
 2: $U \leftarrow \lfloor C/\beta^{k-1} \rceil$
 3: $W \leftarrow (U \times V) \mod E$
 4: $Q \leftarrow \lfloor W/\beta^{k+1} \rceil$
 5: $R \leftarrow C - ((Q \times M) \mod E)$
 6: return $R$
```

3.3 Implementation Results

We implemented in C Algorithm 3 and 4 along with non-randomized Algorithm 1 and 2.We used the following strategies for large and small fields:

- Small fields: $\mathbb{F}_p$ with p of bit-length 256 and 500 bits. AMNS elements are stored in an arrays of n 64-bit word integers. Polynomial multiplication is done using schoolbook method using 64-bits integer multiplication instruction of the processor.
- Larger fields: $\mathbb{F}_p$ with p of bit-length 2048 bits, 3096 bits. AMNS elements are stored in arrays of n 128-bit word integers in order to keep n small. We implemented multiplication of unsigned 128 bits integers through several 64-bit instructions. This approach reduces the efficiency of Barrett multiplication compared to Montgomery multiplication since it involves more signed 128 integer multiplications which are, in this case, less efficient.

We compiled our C code with gcc 9.3.0, and run it on an Ubuntu 20.04 and an Intel Westmere processor. The timings are averages of 2000 multiplications with randomized input.

The above timing results show that for larger fields, the penalty due to signed 128 bits multiplication render non-randomized and randomized Barret AMNS multiplication significantly slower. On small fields one can notice that the randomization on AMNS_BarMul and AMNS_MonMul reduce slightly

Table 2: Timings of AMNS multiplication.

Field and AMNS				Algorithm	#CC
$\log_2(p)$	ρ	n	λ		
3040	2^{110}	30	2	AMNS_MonMul	75527
				Rd_AMNS_MonMul	74930
				AMNS_BarMul	107429
				Rd_AMNS_BarMul	107625
2020	2^{109}	20	2	AMNS_MonMul	33372
				Rd_AMNS_MonMul	34334
				AMNS_BarMul	47661
				Rd_AMNS_BarMul	47624
510	2^{53}	10	2	AMNS_MonMul	1041
				Rd_AMNS_MonMul	1176
				AMNS_BarMul	1507
				Rd_AMNS_BarMul	1632
256	2^{57}	5	2	AMNS_MonMul	207
				Rd_AMNS_MonMul	230
				AMNS_BarMul	201
				Rd_AMNS_BarMul	228

their efficiency compared to non-randomized counter parts.

4 RANDOMIZED DSA EXPONENTIATION

We consider in this section the modular exponentiation involved in Digital Signature Algorithm (DSA (NIST.FIPS.186.4, 2012)). We present a randomized exponentiation based on the randomized Montgomery and Barrett multiplications in AMNS introduced in Subsection 3.2.

4.1 Background on DSA and Side Channel Analysis

DSA security is based on the difficulty of the discrete logarithm problem. Given a prime p, and an element G of order q in the finite field $\mathbb{F}_p$, then, computing the discrete logarithm of $R \in \langle G \rangle$ in base G consists in finding the exponent E such that $R = G^E \mod p$. For a security level larger than 128-bit the prime p has a bit-length is larger than 2048 bits and q has a bit-length larger than 256 bits.

The main computation in DSA is an exponentiation modulo p. Specifically, we have to compute:

$$R = G^E \mod p \qquad (6)$$

where G has order $q|(p-1)$ and $E \in [0, q-1]$. The basic approach to compute the modular exponentiation in (6) consists in a sequence of squares and multiplications in order to reconstruct from the most significant bits to the least significant bits the exponent $E = (e_{\ell-1}, \ldots, e_0)_2$ of R (cf. Algorithm 5).

Algorithm 5: Square-and-multiply.

```
**Require:** $G \in \mathbb{F}_p$ and $E = (e_{\ell-1}, \ldots, e_0)_2$ a positive integer.
**Ensure:** $R$ such that $R = G^E \mod p$
  $R \leftarrow 1$
  **for** $i = 0$ **to** $\ell$ **do**
    $R \leftarrow R^2 \times G^{e_i} \mod p$
  **return** $R$
```

Side Channel Analysis. Sensitive computation on an embedded device can be threaten by side channel analysis. Indeed, such attacks use either power consumption, electromagnetic emanation or computation time to recover part of the secret data involved in the computation. An example of such attacks is the simple power analysis (SPA) on Square-and-multiply exponentiation: assuming that multiplication consume more power than a square we can identify on the power trace the loop iteration involving a multiplication, which are loop iteration corresponding to $e_i = 1$.

The basic protection against SPA consists in rendering the sequence of squares and multiplications of the exponentiation independent to the bits of the exponent. A popular approach to reach this goal is the Montgomery ladder (Algorithm 6) which uses two intermediate variables R_0 corresponding to R in the Square-and-multiply exponentiation and R_1 always satisfying $R_1 = R_0 \times G \mod p$. At each loop iteration we have a multiplication followed by a square, which then does not leak out the corresponding bit e_i of E.

Algorithm 6: Montgomery-ladder (Joye and Yen, 2002).

```
**Require:** An base $G\mathbb{F}_p$ and an exponent $E = (e_{\ell-1}, \ldots, e_0)_2$
**Ensure:** $R_0 = G^E \mod N$
1: $R_0 \leftarrow 1, R_1 \leftarrow G$
2: **for** $i$ **from** 0 **to** $\ell - 1$ **do**
3:     $R_{1-e_i} \leftarrow R_0 \times R_1 \mod p$
4:     $R_{e_i} \leftarrow R_{e_i}^2 \mod p$
5: **return** $R_0$
```

There are more powerful attacks like the Differential Power Analysis (DPA) (Kocher et al., 1999) which guesses a bit of the exponent and correctly predict power consumption of a loop iteration. Or the Correlation Power Analysis (CPA) (Brier et al., 2004) which recovers a bit of exponent by correlating power consumption of two consecutive loop iterations. To counteract these attacks the main strategy consists in randomizing the data involved in the exponentiation (the exponent E and intermediate variables R_0 and R_1).

Specifically, one strategy to randomize the data is the *base blinding:*. This strategy was proposed in (Coron, 1999) and consists in multiplying G with a random α, assuming that $\beta = \alpha^E \mod p$ is precomputed. Then we have

$$G' = G \times \alpha \bmod p \Rightarrow G'^E(\beta^{-1}) \bmod p = G^E \bmod p$$

In (Lesavourey et al., 2016) the authors propose to randomly update the multiplicative mask α with the Montgomery factor induced by Montgomery multiplication. Their randomization is limited by the fact that this random mask should be equal to 1 at the end of the exponentiation and then is reduced to a small set of values.

4.2 Randomized Montgomery Ladder for DSA

Our approach consists in running the Montgomery ladder with input given in an AMNS $(p,n,\rho,\gamma,\lambda)$. At each loop iteration we pick two random bits t_i and s_i and then the two modular multiplications are computed as follows:

- If $t_i = 1$ we apply Rd_AMNS_MonMul with randomizing parameter s_i, in this case the output has a multiplicative mask equal to $\phi^{-1} \times \gamma^{-s_i}$.
- If $t_i = 0$ we apply Rd_AMNS_BarMul with randomizing parameter s_i, in this case the output has a multiplicative mask equal to γ^{-s_i}.

The resulting randomized Montgomery ladder is shown in Algorithm 7.

Algorithm 7: Randomized Montgomery Ladder.

```
Require: $G$ of order $q$ in $\mathbb{F}_p$ where $q$ of bit-length $\ell$, an exponent $E = (e_{\ell-1},..,e_1e_0)_2$, $w$ the bit-length of the computer words, an AMNS $(p,n,\rho,\gamma,\lambda)$ with $\lambda = 2$ and precomputed data $u \leftarrow \lfloor (p-1)/2^\ell \rfloor, v \leftarrow (p-1) \bmod 2^\ell$ and $\beta = \gamma^{-u} \mod p$.
Ensure: $R = G^E \mod p$
 1: $T \leftarrow Random(0,...,\lfloor (v/(nw)) \rfloor)$
 2: $S \leftarrow v - nwT$
 3: $R_0(X) \leftarrow AMNS(1 \times \beta \mod p)$
 4: $R_1(X) \leftarrow AMNS(G \mod p)$
 5: for $i = \ell - 1$ to 0 do
 6:   if $t_i = 1$ then
 7:     $R_{e_i} \leftarrow$ Rd_AMNS_MonMul$(R_{e_i}, R_{1-e_i}, s_i)$
 8:     $R_{1-e_i} \leftarrow$ Rd_AMNS_MonMul$(R_{e_i}, R_{e_i}, s_i)$
 9:   else
10:     $R_{e_i} \leftarrow$ Rd_AMNS_BarMul$(R_{e_i}, R_{1-e_i}, s_i)$
11:     $R_{1-e_i} \leftarrow$ Rd_AMNS_BarMul$(R_{e_i}, R_{e_i}, s_i)$
12: $R \leftarrow R_0(\gamma) \mod p$
13: return $R$
```

At each iteration of the above algorithm R_0 R_1 are multiplied by a factor which can be 1, ϕ^{-1}, γ^{-1} or $\phi^{-1}\gamma^{-1}$. These multiplications contribute to randomly modify a multiplicative mask on R_0 and R_1 providing a protection against side channel analyses like DPA or CPA. But this mask should be equal to 1 at the end of the exponentiation in order to have R_0 equal to the correct output $G^E \mod p$. Steps 1 and 2 of the algorithm set the necessary conditions on the random bits in t_i and s_i which ensure that the random mask induced by the factors $\phi^{-t_i}\gamma^{-s_i}$ and β is equal to 1 at the end. This is shown in the following lemma.

Lemma 1. *Algorithm 7 correctly outputs $R = G^E \bmod p$.*

Proof. Let us first unroll the algorithm to understand how the random mask evolves during the exponentiation:

- After loop $i = \ell - 1$ we have
$$R_0 = \beta^2 G^{e_{\ell-1}}(\gamma^{-1})^{s_{\ell-1}}(\phi^{-1})^{t_{\ell-1}},$$
$$R_1 = \beta^2 G^{e_{\ell-1}+1}(\gamma^{-1})^{s_{\ell-1}}(\phi^{-1})^{t_{\ell-1}}.$$
- After loop $i = \ell - 2$ whe have
$$R_0 = \beta^4 G^{2e_{\ell-1}+e_{\ell-2}}(\gamma^{-1})^{2s_{\ell-1}+s_{\ell-2}}(\phi^{-1})^{2t_{\ell-1}+t_{\ell-2}},$$
$$R_1 = \beta^4 G^{2e_{\ell-1}+e_{\ell-2}+1}(\gamma^{-1})^{2s_{\ell-1}+s_{\ell-2}}(\phi^{-1})^{2t_{\ell-1}+t_{\ell-2}}.$$
- ...
- After loop $i = 0$, we have

$$R_0 = \beta^{2^\ell} G^E (\gamma^{-1})^S (\phi^{-1})^T, \ R_1 = \beta^{2^\ell} G^{E+1}(\gamma^{-1})^S(\phi^{-1})^T.$$

We consider the last multiplicative mask of R_0

$$\beta^{2^\ell}(\gamma^{-1})^S(\phi^{-1})^T$$

we use the fact that $\beta = \gamma^{-u}$ and $\phi = 2^w$ and $2 = \gamma^n \bmod p$ which leads to

$$\beta^{2^\ell}(\gamma^{-1})^S(\phi^{-1})^T = 2^{-2^\ell u}\gamma^{-S}2^{-wT}$$
$$= \gamma^{-(2^\ell u + S + nwT)} = \gamma^{-(2^\ell u + v)} = \gamma^{-(p-1)} = 1$$

□

Level of Randomization. At the beginning of the Montgomery ladder there are no random mask on R_0 and R_1 (β is a public value), but the random mask is growing by one bits after each iteration. This means that the level of randomization injected at the end is 2^ℓ which correspond to the size of the random data T. This is a larger level of dynamic randomization than the one of (Lesavourey et al., 2016).

4.3 Implementation Results

We implemented in C the randomized and non-randomized form of Montgomery ladder using AMNS modular multiplication. We considered DSA exponentiation for the two cryptographic size 2048 bits and 3096 bits for p. We also considered (non-DSA) exponentiation on small field from 256 bits and 500 bits since for these size AMNS Barrett multiplications is as efficient as their AMNS Montgomery counter-parts. The resulting timing results are reported in Table 3.

We notice that for large fields, the use of slow AMNS Barrett multiplication render the proposed approach not efficient. For smaller fields, particularly for field of size 256-bits the randomized approach is competitive. This means that if we could improve signed 128-bit multiplication we could get randomized exponentiation on large field with smaller penalty.

Table 3: Timings of exponentiation.

Field and AMNS					Algorithm	#CC
$\log_2(p)$	$\log_2(q)$	ρ	n	λ		
3040	300	2^{110}	30	2	Mont. Ladder	46036405
					Rand. Mont. Ladder	57285659
2020	256	2^{109}	20	2	Mont. Ladder	15911757
					Rand. Mont. Ladder	21920264
510	510	2^{53}	10	2	Mont. Ladder	1211894
					Rand. Mont. Ladder	1662200
256	256	2^{57}	5	2	Mont. Ladder	106877
					Rand. Mont. Ladder	116832

5 RANDOMIZED SCALAR MULTIPLICATION

We present in this section a strategy to dynamically randomize data in scalar multiplication on an elliptic curve $E(\mathbb{F}_p)$. First, we provide the necessary background on elliptic curve and related algorithms.

5.1 Background on Elliptic Curve

An elliptic curve $E(\mathbb{F}_p)$ is the set of points $(x,y) \in \mathbb{F}_p^2$, along with a point at infinity O, which satisfy an equation of the form:

$$y^2 = x^3 + ax + b \text{ where } a,b \in \mathbb{F}_p$$

with $\Delta = -16(4a^3 + 27b^2) \neq 0$. There is a additive group law on $E(\mathbb{F}_p)$ which is derived from the chord and tangent rules: a line crossing two points P,Q on the curve intersects the curve on a third point R', then $R = P + Q$ is defined as the x-axis symmetric of R'. The coordinates of $R = (x_3, y_3)$ can be computed with a few operations in $\mathbb{F}_p$ from the coordinates of $P = (x_1, y_1)$ and $Q = (x_2, y_2)$

$$\begin{cases} x_3 = \lambda - x_1 - x_2 \\ y_3 = y_1 - \lambda(x_3 - x_1) \end{cases} \text{ with } \lambda = \begin{cases} \frac{y_1 - y_2}{x_1 - x_2} \text{ if } P \neq Q \\ \frac{3x_1^2 + a}{2y_1} \text{ if } P = Q \end{cases}$$

To improve the efficiency of these operations, a various set of projective coordinate system was used in order to avoid costly inversions. Among them there is the Jacobian coordinates (X,Y,Z) which corresponds to affine coordinates $(x,y) = (X/Z^2, Y/Z^3)$.

Definition 2. *Two points given in Jacobian coordinates are equivalent* $(X,Y,Z) \sim (X',Y',Z')$ *if there is* $\beta \in \mathbb{F}_p$ *such that*

$$(X',Y',Z') = (X\beta^2, Y\beta^3, Z\beta)$$

these two Jacobian coordinates correspond to the same affine point $(x,y) = (X/Z^2, Y/Z^3) = (X'/Z'^2, Y'/Z'^3)$.

The use of Jacobian coordinates avoids inversion in $\mathbb{F}_p$ but, in counterpart, this increases the number of multiplications per point operation. In the literature, several improvements were proposed to simplify these Jacobian formula. We will focus here on the co-Z formula for addition which was first proposed in (Méloni, 2007) and then extended in (Goundar et al., 2011) to a few other co-Z point operations. Co-Z point formula take as input two points in Jacobian coordinates sharing the same Z coordinate. In (Méloni, 2007) they show that this simplifies Jacobian addition and leads to the formula shown in Algorithm 8. The last operations in Step 8 of Algorithm 8 are meant to update the input P such that it shares the same Z coordinate as $R = P + Q$.

Algorithm 8: Co-Z addition with update (ZADDU) (Méloni, 2007).

```
Require: $P = (X_1, Y_1, Z)$ and $Q = (X_2, Y_2, Z)$
Ensure: $(R, P')$ such that $R = P + Q$ and $P' \sim P$
 1: $C \leftarrow (X_1 - X_2)^2$
 2: $W_1 \leftarrow X_1 C$, $W_2 \leftarrow X_2 C$
 3: $D \leftarrow (Y_1 - Y_2)^2$, $A_1 \leftarrow Y_1(W_1 \leftarrow W_2)$
 4: $X_3 \leftarrow D - W_1 - W_2$
 5: $Y_3 \leftarrow (Y_1 - Y_2)(W_1 - X_3) - A_1$
 6: $Z_3 \leftarrow Z(X_1 - X_2)$
 7: $X_1 \leftarrow W_1$, $Y_1 \leftarrow A_1$, $Z_1 \leftarrow Z_3$
 8: $R \leftarrow (X_3, Y_3, Z_3), P' \leftarrow (X_1, Y_1, Z_1)$
 9: return $R, P'$
```

In (Goundar et al., 2011) the authors adapt the ZADDU approach to the computation of $P + Q$ and $P - Q$ with shared Z. They use the fact that most computation for $P + Q$ and $P - Q$ are the same, unless for

$P-Q$ we have to negate the Y coordinate of Q. This leads to ZADDC algorithm ouputing $P+Q$ and $P-Q$ with shared Z.

In (Goundar et al., 2011) the authors take advantage of ZADDU and ZADDC to get a variant of the Montgomery ladder for scalar multiplication. Indeed, the two operations $R_{1-b} \leftarrow R_b + R_{1-b}$ and $R_b \leftarrow 2R_b$ in the Montgomery ladder can be done as follows:

$$\begin{aligned}(R_{1-b}, R_b) &\leftarrow ZADDC(R_b, R_{1-b}) \\ &= (R_b + R_{1-b}, R_b - R_{1-b}), \\ (R_b, R_{1-b}) &\leftarrow ZADDU(R_{1-b}, R_b) \\ &= (R_b + R_{1-b} + R_b - R_{1-b}, R_b - R_{1-b}) \\ &= (2R_b, R_b + R_{1-b}).\end{aligned}$$

5.2 Randomized Scalar Multiplication on Elliptic Curve

Jacobian coordinates can be used to randomly mask a point. Indeed, given a point $P = (X, Y, Z)$, we randomly pick $\beta \in \mathbb{F}_p$ then we compute $P' = (X\beta^2, Y\beta^3, Z\beta)$ which are equivalent Jacobian coordinates of the affine point $(X/Z^2, Y/Z^3)$.

We propose to use Rand_AMNS_BarMul to dynamicly randomize the coordinates of the points R_0 and R_1 during the scalar multiplication. We first focus on randomizing the ZADDU curve operation. Given a randomizing integer t, we perform all multiplications involved in ZADDU with Rd_AMNS_BarMul with parameter $s = t$, only, the square in Step 3 is computed with parameter $s = 2t$. This approach is shown in Algorithm 9.

Algorithm 9: Randomized Co-Z addition with update (Rand_ZADDU).

```
Require: $P' = (X'_1, Y'_1, Z')$ and $Q' = (X'_2, Y'_2, Z')$ with coordinates in an AMNS $(p, n, \rho, \gamma, \lambda)$ with $\lambda = 2$.
Ensure: $(R', P')$ such that $R = P + Q$ and $P' \cong P$
 1: $C' \leftarrow Rd\_AMNS\_BarMul((X'_1 - X'_2), (X'_1 - X'_2), t)$
 2: $W'_1 \leftarrow Rd\_AMNS\_BarMul(X'_1, C', t)$,
 3: $W'_2 \leftarrow Rd\_AMNS\_BarMul(X'_2, C', t)$
 4: $D' \leftarrow Rd\_AMNS\_BarMul((Y'_1 - Y'_2), (Y'_1 - Y'_2), 2t)$
 5: $A'_1 \leftarrow Rd\_AMNS\_BarMul(Y'_1, (W'_1 - W'_2), t)$
 6: $X'_3 \leftarrow D' - W'_1 - W'_2$
 7: $Y'_3 \leftarrow Rd\_AMNS\_BarMul(Y'_1 - Y'_2), (W'_1 - X'_3), t) - A'_1$
 8: $Z'_3 \leftarrow Rd\_AMNS\_BarMul(Z', (X'_1 - X'_2), t)$
 9: $X''_1 \leftarrow W'_1$, $Y''_1 \leftarrow A'_1$, $Z''_1 \leftarrow Z'_3$
10: $R' \leftarrow (X'_3, Y'_3, Z'_3), P'' \leftarrow (X'_1, Y'_1, Z'_1)$
11: return $R', P''$
```

Let us show that the two points R' and P'' output by Algorithm 7 have Jacobian coordinates equivalent to the points R and P output by ZADDU. We assume that the input points P' and Q' are equivalent to P and Q. Which means for P' that there exists β such that

$$Z' = Z\beta, X'_1 = X_1\beta^2 \text{ and } Y'_1 = Y_1\beta^3$$

and the same β applies for the equivalence between Q' and Q. Now, one can notice that:

$$\begin{aligned} C' &= (X'_1 - X'_2)^2\gamma^{-t} = C\beta^4\gamma^{-t}, \\ W'_1 &= X'_1 C'\gamma^{-t} = X_1 C\beta^6\gamma^{-2t} = W_1\beta^6\gamma^{-2t} \\ W'_2 &= X'_2 C'\gamma^{-t} = X_2 C\beta^6\gamma^{-2t} = W_2\beta^6\gamma^{-2t} \\ D' &= (Y'_1 - Y'_2)^2\gamma^{-2t} = D\beta^6\gamma^{-2t} \\ A'_1 &= (Y'_1 - Y'_2)(W'_1 - W'_2)\gamma^{-t} = A_1\beta^9\gamma^{-3t} \end{aligned}$$

Then we can express the coordinates of R' in terms of the ones of R:

$$\begin{aligned} X'_3 &= D\beta^6\gamma^{-2t} - W_1\beta^6\gamma^{-2t} - W_2\beta^6\gamma^{-2t} \\ &= X_3\beta^6\gamma^{-2t}, \\ Y'_3 &= (Y'_1 - Y'_2)(W'_1 - X'_3)\gamma^{-t} - A'_1 \\ &= (Y_1 - Y_2)(W_1 - X_3)\beta^9\gamma^{-3t} - A_1\beta^9\gamma^{-3t}, \\ &= Y_3\beta^9\gamma^{-3t} \\ Z'_3 &= Z'(X'_1 - X'_2)\gamma^{-t} = Z_3\beta^3\gamma^{-t}, \end{aligned}$$

But the above equations show that $R' \sim R$ with $\beta' = \beta^3\gamma^{-t}$. Similarly for P'' we have

$$\begin{aligned} X''_1 &= W'_1 = W_1\beta^6\gamma^{-2t} \\ Y''_1 &= A'_1 = A_1\beta^9\gamma^{3t} \end{aligned}$$

which means that the Jacobian coordinates P'' are equivalent to the ones of P' output by ZADDU with $\beta' = \beta^3\gamma^{-t}$.

The same strategy can be applied to the conjugate addition ZADDC leading to the same Jacobian coordinates factor $\beta^3\gamma^{-t}$. Now using these Rand_ZADDU and Rand_ZADDC we can randomize the co-Z Montgomery ladder as shown in Algorithm 10.

Algorithm 10: Randomized co-Z Montgomery ladder.

```
Require: $P = (x_P, y_P) \in E(\mathbb{F}_p)$ and $e = (e_{\ell-1}, \ldots, e_0) \in \mathbb{N}$ with $e_{\ell-1} = 1$.
Ensure: $Q = eP$
1: $(R_1, R_0) \leftarrow DBLU(P)$
2: $(t_{2\ell-1}, \ldots, t_0)_3 \leftarrow Random(3^{2\ell})$
3: for $i = \ell - 1$ to 0 do
4:     $b \leftarrow k_i$
5:     $(R_{1-b}, R_b) \leftarrow Rand\_ZADDC(R_b, R_{1-b}, t_{2i+1})$
6:     $(R_b, R_{1-b}) \leftarrow Rand\_ZADDU(R_{1-b}, R_b, t_{2i})$
7: return $Jac2aff(R_0)$
```

From the analysis on Rand_ZADDU and Rand_ZADDC we know that they output points equivalent to non-randomized ZADDU and ZADDC, which means that the randomized Co-Z Montgomery ladder correctly output the point $R = eP$.

Level of Randomization. At each loop iteration the random multiplicative mask β evolves as $\beta^3 \times \gamma^{-t_i}$ for Rand_ZADDC and Rand_ZADDU. Since t_i is in $\{0, 1, 2\}$, this sequence of operations for $i = \ell - 1, \ldots 0$

consists in a cube and multiply exponentiation of γ. Which means that in loop i, the random factor is $\beta = \gamma^{-T_i}$ for some T_i of size $\sim 3^{2(\ell-i)}$ and at the end we have $\beta = \gamma^{-T}$ for $t \sim 3^{2\ell}$.

In other words, the level of randomization is low during the first loops of the algorithm but it grows quickly and it is really large at the end. The lack of randomization at the beginning can be overcome by picking an random β and compute an equivalent Jacobian coordinates R'_0 and R'_1 with factor β at just after Step 1 in Algorithm 9.

5.3 Implementation Results

Our implementation are done in C using our code for randomized and non-randomized AMNS multiplication presented in Subsection 3.3. The timings of randomized scalar multiplication are reported in Table 4. For field size 510 bits, the proposed randomization is significantly slower, but we don't know what is the reason for that. But for field size 256 bits the proposed randomization is competitive with the non-randomized version.

Table 4: Timings of scalar multiplication.

Field and AMNS					Algorithm	#CC
$\log_2(p)$	ρ	n	λ			
510	510	2^{53}	10	2	co-Z Mont. Ladder	8891506
					Rd. co-Z Mont. Ladder	12683841
256	256	2^{57}	5	2	co-Z Mont. Ladder	939424
					Rd. co-Z Mont. Ladder	885270

6 CONCLUSION

In this paper we considered randomization for DSA exponentiation and elliptic curve scalar multiplication. Our randomization take advantage of the modular multiplication in AMNS. We then presented a randomized AMNS multiplication using modified polynomial reduction and random choice between Barrett and Montgomery multiplication. This leads to a randomizing factor $\phi^{-t}\gamma^{-s}$ for some $t \in \{0,1\}$ and $s \in \{0,\ldots,n-1\}$. We then presented randomized DSA exponentiation and co-Z elliptic curve scalar multiplication using these modified AMNS multiplications. This improves the level of randomization, with, in the best case, a limited loss of performance.

REFERENCES

Bajard, J., Imbert, L., and Plantard, T. (2004). Modular Number Systems: Beyond the Mersenne Family. In *SAC 2004*, volume 3357 of *LNCS*, pages 159–169. Springer.

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.

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.

Clavier, C., Feix, B., Gagnerot, G., Roussellet, M., and Verneuil, V. (2010). Horizontal Correlation Analysis on Exponentiation. In *ICICS 2010*, volume 6476 of *LNCS*, pages 46–61. Springer.

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

Didier, L.-S., Dosso, F.-Y., Mrabet, N. E., Marrez, J., and Véron, P. (2019). Randomization of arithmetic over polynomial modular number system. In *ARITH 2019*, pages 199–206. IEEE.

Goundar, R. R., Joye, M., Miyaji, A., Rivain, M., and Venelli, A. (2011). Scalar multiplication on Weierstraß elliptic curves from Co-Z arithmetic. *J. Cryptogr. Eng.*, 1(2):161–176.

Jao, D. and Feo, L. D. (2011). Towards Quantum-Resistant Cryptosystems from Supersingular Elliptic Curve Isogenies. In *Post-Quantum Cryptography 2011*, volume 7071 of *LNCS*, pages 19–34. Springer.

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

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.

Lesavourey, A., Nègre, C., and Plantard, T. (2016). Efficient Randomized Regular Modular Exponentiation using Combined Montgomery and Barrett Multiplications. In *SECRYPT 2016*, pages 368–375. SciTePress.

Méloni, N. (2007). New Point Addition Formulae for ECC Applications. In *WAIFI 2007*, volume 4547 of *LNCS*, pages 189–201. Springer.

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

Nègre, C. and Plantard, T. (2008). Efficient Modular Arithmetic in Adapted Modular Number System Using Lagrange Representation. In *ACISP 2008*, volume 5107 of *LNCS*, pages 463–477. Springer.

NIST.FIPS.186.4 (2012). Digital Signature Standad (DSS). Standard, NIST.

Plantard, T. (2005). *Arithmétique modulaire pour la cryptographie*. PhD thesis, Montpellier 2 University, France.

Tunstall, M. and Joye, M. (2010). Coordinate blinding over large prime fields. In *CHES 2010*, volume 6225 of *LNCS*, pages 443–455. Springer.