Efficient and secure modular operations using the Polynomial Modular Number System (Part 1)

Laurent-Stéphane Didier[1], **Fangan Yssouf Dosso**[1], Nadia El Mrabet[2], Jérémy Marrez[3], Pascal Véron[1]

[1] IMATH, University of Toulon
[2]École des mines de Saint-Étienne, Gardanne
[3]LIP6, Sorbonne University

Workshop on Randomness and Arithmetics for Cryptography on Hardware
Roscoff, April 19 2019

Introduction

About the PMNS (Polynomial Modular Number System):

- Goal: Perform efficiently and safely modular arithmetic operations on big integers.
- Main feature: Uses polynomial representation for its elements.

Motivations:

- Construction of PMNS for any (prime) integer.
- Study the efficiency of these PMNS.
- Use PMNS as tool against (some) side channel attacks.

Plan

Definition: MNS (Modular Number System)

Let p be an integer.

Definition

A MNS for p is defined by a tuple $\mathcal{B} = (p,\ n,\ \gamma,\ \rho)$ such that for every integer $0 \leqslant y < p$, there exists a polynomial $V(X) = v_0 + v_1.X + \cdots + v_{n-1}.X^{n-1}$ which satisfies:

- $|v_i| < \rho$
- $y \equiv V(\gamma) \pmod{p}$

where $0 < \gamma < p$ and $\rho \approx \sqrt[n]{p}$

Example of MNS

0	1	2	3	4
0	1	$-X^2$	$1-X^2$	$-1+X+X^2$

5	6	7	8	9	10
$X+X^2$	$-1+X$	X	$1+X$	$-X-1$	$-X$

11	12	13	14	15	16
$-X+1$	$-X-X^2$	$1-X-X^2$	$-1+X^2$	X^2	-1

Table: The elements of $\mathbb{Z}/17\mathbb{Z}$ in $\mathcal{B} = (p, n, \gamma, \rho) = (17, 3, 7, 2)$.

Arithmetic operations

Main operations:

- **Addition:** a simple polynomial addition.
  But, result infinity norm can be greater than ρ. **(1)**

- **Multiplication:** a simple polynomial multiplication.
  But, result infinity norm can be greater than ρ **(1)**
  and result degree can be greater than $n - 1$. **(2)**

- In case 1, an **internal reduction** must be done.
- In case 2, an **external reduction** must be done.

The Polynomial Modular Number Systems (PMNS)

Introduced to perform the internal and external reductions efficiently.

Let p be an integer.

Definition

A PMNS for p is defined by a tuple $\mathcal{B} = (p, n, \gamma, \rho, E)$ such that:

- (p, n, γ, ρ) is a MNS,
- E is a monic polynomial such that:
  - $\deg(E) = n$,
  - $E(\gamma) \equiv 0 \pmod p$,
  - $\|E\|_\infty$ is small.

Arithmetic operation: the external reduction

Let $\mathcal{B} = (p,\ n,\ \gamma,\ \rho,\ E)$ be a PMNS and $A, B \in \mathcal{B}$.

Let $C = A.B$ be a polynomial, then $\deg(C) < 2n - 1$.

Goal: Compute a polynomial R such that: $R(\gamma) \equiv C(\gamma) \pmod{p}$ and $\deg(R) < n$.

How it works

- There exists $Q \in \mathbb{Z}[X]$ and $R \in \mathbb{Z}[X]$ such that:
  $C = Q.E + R$, where $\deg(R) < n$.
  As $E(\gamma) \equiv 0 \pmod{p}$, $R(\gamma) \equiv C(\gamma) \pmod{p}$.
- **External reduction:** $R = C \pmod{E}$

Arithmetic operation: the internal reduction

Let $\mathcal{B} = (p,\ n,\ \gamma,\ \rho,\ E)$ be a PMNS.
Let $C \in \mathbb{Z}[X]$ be a polynomial such that deg(C)$< n$.

Goal: Compute a polynomial R such that: $R(\gamma) \equiv C(\gamma) \pmod{p}$ and $R \in \mathcal{B}$.

Can be done in several ways.
When p can't be chosen freely, the best proposal is a Montgomery-like method; (by C. Nègre and T. Plantard).

The internal reduction: a Montgomery-like method

Let $\mathcal{B} = (p,\ n,\ \gamma,\ \rho,\ E)$ be a PMNS.
It requires two polynomials M and M' such that: $M \in \mathcal{B}$, $M(\gamma) \equiv 0 \pmod{p}$ and $M' = -M^{-1} \bmod (E, \phi)$, with $\phi \in \mathbb{N} \setminus \{0\}$.

Algorithm: RedCoeff

- 1: **Input:** a polynomial V, such that: $deg(V) < n$
- 2: **Ensure:** $S(\gamma) = V(\gamma)\phi^{-1} \bmod p$
- 3: $Q \leftarrow V \times M' \bmod (E,\ \phi)$
- 4: $T \leftarrow Q \times M \bmod E$
- 5: $S \leftarrow (V + T)/\phi$ # exact divisions
- 6: **return** S

For optimal efficiency, ϕ should be taken as power of two.

About the parameters M and M'

- The polynomial M' is such that $M' = -M^{-1} \bmod (E, \phi)$, with $\phi \in \mathbb{N} \setminus \{0\}$. So, $M^{-1} \bmod (E, \phi)$ must exist.

- In 2012, Nadia El Mrabet and Nicolas Gama showed how to generate the polynomial M such that $M^{-1} \bmod (E, \phi)$, with $E = X^n + 1$ and ϕ as a power of two.

- Recently (in 2018), Laurent-Stephane Didier, Pascal Véron and Yssouf Dosso showed how to generate the polynomial M such that $M^{-1} \bmod (E, \phi)$, with $E = X^n - \lambda$ ($\lambda \in \mathbb{Z} \setminus \{0\}$) and ϕ as a power of two.

Some advantages of the PMNS

- High parallelization capability, because elements are polynomials.
- No carry propagation to deal with, because elements coefficients are independent.
- There is no conditional branching.

Additional works on PMNS

PMNS can be an interesting alternative to the usual number system. Example of ratios for cryptographic size integers (implementation in C without parallelization):

(p size, n)	(192, 4)	(224, 4)	(256, 5)	(384, 7)	(521, 10)
ratio 1	**0.86**	**0.57**	**0.98**	**0.98**	**0.95**
ratio 2	0.10	0.08	0.14	0.19	0.25
ratio 3	0.21	0.16	0.30	0.43	0.56
ratio 4	0.36	0.23	0.45	0.61	0.69

Table: Relative performances of PMNS vs GNU MP and OpenSSL, for modular multiplication

ratio 1: PMNS/OpenSSL Montgomery modular mult.
ratio 2: PMNS/OpenSSL default modular mult.
ratio 3: PMNS/GNU MP mult. + modular reduction.
ratio 4: PMNS/GNU MP mult. + modular reduction, using low level functions.

Randomisation using the PMNS

Let $p > 0$ be a (prime) integer.

Main idea: provide many distinct representations for each element in $\mathbb{Z}/p\mathbb{Z}$.

Two types of randomisation:

- The external randomisation: uses the existence of many PMNS for given an integer.
- The internal randomisation: uses the redundancy in the PMNS.

The external randomisation

It is a randomisation from PMNS to PMNS.

We showed that it is always possible to generate many PMNS, given a prime p.

How it works:

1. Generate a set Ω of PMNS for the required modulus.
2. Each time a protocol using that modulus is executed, randomly select a PMNS in Ω to perform arithmetic operations.

We call this the **external randomisation**.

The internal randomisation

It is a randomisation inside the PMNS.

Goals:

- Randomise conversion process in the PMNS.
- Randomise the modular multiplication in the PMNS.

We call this the **internal randomisation**.

General idea:

We introduce a parameter $z \in \mathbb{N}$.

Let $\mathcal{H} = \{Z \in \mathbb{Z}[X]$, such that: $deg(Z) < n$ and $\|Z\|_\infty \leqslant z\}$.
We have: $\#\mathcal{H} = (2\,z + 1)^n$.

We generate the PMNS $\mathcal{B} = (p,\ n,\ \gamma,\ \rho,\ E)$ such that:

- Given $x \in \mathbb{Z}/p\mathbb{Z}$, each element $Z_i \in \mathcal{H}$ allows to compute a representation $A_i \in \mathcal{B}$ of x.
- If $Z_i \neq Z_j$, then $A_i \neq A_j$.

So, each element in $\mathbb{Z}/p\mathbb{Z}$ has at least $\#\mathcal{H}$ **distinct** representations in $\mathcal{B}$.

Requirements

Let $\mathcal{B} = (p,\ n,\ \gamma,\ \rho,\ E)$ be a PMNS and $A \in \mathcal{B}$.

For the internal randomisation to work, three requirements have to be met:

- The randomisation must not modify $A(\gamma) \pmod{p}$.
- Randomised operations should output result in $\mathcal{B}$.
- If $Z_i \neq Z_j$, then randomisations using Z_i and Z_j should output different representations; i.e: guarantee that there is no collision.

Randomisation of the conversion process: the algorithm

For consistency, a conversion to Montgomery domain is done. We need to precompute representations $P_i(X)$ of $(\rho^i \phi^2)$ in $\mathcal{B}$.

Algorithm: RandConv

- 1: **Input:** $a \in \mathbb{Z}/p\mathbb{Z}$
- 2: **Ensure:** $A \equiv (a.\phi)_{\mathcal{B}}$
- 3: $Z \leftarrow$ **RandPoly**(z) # randomly generate an element of $\mathcal{H}$
- 4: $t = (a_{n-1}, ..., a_0)_\rho$ # radix-ρ decomposition of a
- 5: $U \leftarrow \sum_{i=0}^{n-1} t_i \, P_i$
- 6: $V \leftarrow U + ((\phi + 1)Z \times M) \mod E$ # $V(\gamma) \equiv U(\gamma) \pmod{p}$
- 7: $A \leftarrow$ **RedCoeff**(V)
- 8: **return** A

Randomisation of the conversion process

Conditions on ρ and ϕ for the three requirements to be met:

$$\rho \geqslant 2.n.s.\|M\|_\infty.\left(1 + z + \tfrac{z}{\phi}\right) \quad \text{and} \quad \phi \geqslant 2.n.s.\rho$$

Without randomisation, we need:

$$\rho \geqslant 2.n.s.\|M\|_\infty \quad \text{and} \quad \phi \geqslant 2.n.s.\rho$$

The factor s is due to reductions modulo E. It can be easy computed once E is known.

Randomisation of the multiplication: the algorithm

One input is randomised so that all the operations are randomised too.

Algorithm: RandMult

- 1: **Input:** $A \in \mathcal{B}$ and $B \in \mathcal{B}$
- 2: **Ensure:** $R(\gamma) = A(\gamma)B(\gamma)\phi^{-1} \mod p$
- 3: $Z \leftarrow$ **RandPoly**(z) # randomly generate an element of $\mathcal{H}$
- 4: $J \leftarrow Z \times M \mod E$
- 5: $B' \leftarrow B + J$
- 6: $C \leftarrow (A \times B') \mod E$
- 7: $Q \leftarrow (C \times M') \mod (E, \phi)$
- 8: $R' \leftarrow C + (Q \times M) \mod E$
- 9: $R \leftarrow R'/\phi + 2 \times J$
- 10: **return** R

Randomisation of the multiplication

Conditions on ρ and ϕ for the three requirements to be met:

$$\rho \geqslant 2.n.s.\|M\|_\infty.(2z+1) \quad \text{and} \quad \phi \geqslant 2.n.s.\rho.max\left(z, \tfrac{5}{4}\right)$$

Allow to randomise both the conversion and the multiplication.

Remarks:

- Without randomisation, we need:

$$\rho \geqslant 2.n.s.\|M\|_\infty \quad \text{and} \quad \phi \geqslant 2.n.s.\rho$$

- For randomised conversion only, we need:

$$\rho \geqslant 2.n.s.\|M\|_\infty.\left(1+z+\tfrac{z}{\phi}\right) \quad \text{and} \quad \phi \geqslant 2.n.s.\rho$$

Cost evaluation: theoretical costs

In table below, we compare the non-randomised Montgomery-like modular multiplication to the randomised one.
We assume: $\phi = 2^j$, $\rho = 2^w$, $E(X) = X^n - \lambda$ with $\lambda = \pm 2^u$.

Mult. Method	Montgomery-like
Polynomial Mult.	$n^2\mathcal{M} + (2n^2 - 4n + 2)\mathcal{A}$
External reduct.	$2(n-1)\mathcal{A} + (n-1)\mathcal{S}_l^u$
Internal reduct.	$2n^2\mathcal{M} + (3n^2 - n)\mathcal{A} + n\mathcal{S}_r^j$
Total	$3n^2\mathcal{M} + (5n^2 - 3n)\mathcal{A} + (n-1)\mathcal{S}_l^u + n\mathcal{S}_r^j$
Mult. Method	**Randomised Montgomery-like**
Polynomial Mult.	$2n^2\mathcal{M} + (3n^2 - 4n + 2)\mathcal{A} + \mathcal{R}$
External reduct.	$2(n-1)\mathcal{A} + (n-1)\mathcal{S}_l^u$
Internal reduct.	$2n^2\mathcal{M} + 3n^2\mathcal{A} + n(\mathcal{S}_r^j + \mathcal{S}_l^1)$
Total	$4n^2\mathcal{M} + (6n^2 - 2n)\mathcal{A} + (n-1)\mathcal{S}_l^u + n(\mathcal{S}_l^1 + \mathcal{S}_r^j) + \mathcal{R}$

$\mathcal{M}$ and $\mathcal{A}$ respectively denote the multiplication and the sum of two w-bits integers. $\mathcal{R}$ is the cost of one call to the `RandPoly` function. $\mathcal{S}_l^i$ and $\mathcal{S}_r^i$ are respectively a left shift and a right shift of i bits.

Conclusion

We have shown that:

- For any (prime) integer, it is possible to generate many PMNS.
- The PMNS can be an interesting alternative to classical methods like Montgomery modular multiplication.
- The PMNS can be used to randomise modular operations.

Some perspectives:

- Implement PMNS using its high parallelization capability.
- For side channel attacks, make a deeper study to establish the relevance of these proposals with regard to existing countermeasures.

Thank you for your attention.

Questions ?

References

1. Mrabet, N.E., Gama, N.: Efficient multiplication over extension fields. In: WAIFI. Lecture Notes in Computer Science, vol. 7369, pp. 136–151. Springer (2012)
2. Nègre, C., Plantard, T.: Efficient modular arithmetic in adapted modular number system using lagrange representation. In: Information Security and Privacy, 13th Australasian Conference, ACISP 2008, Wollongong, Australia. pp. 463–477 (2008)
3. Plantard, T.: Arithmétique modulaire pour la cryptographie. Ph.D. thesis, Montpellier 2 University, France (2005)