RNS Modular Multiplication through Reduced Base Extensions

Karim Bigou, Arnaud Tisserand

RNS Modular Multiplication through Reduced Base Extensions

Karim Bigou[2,1] and Arnaud Tisserand[3,1]
[1]IRISA, [2]INRIA Centre Rennes - Bretagne Atlantique, [3]CNRS, University Rennes 1
6 rue Kerampont, CS 80518, F-22305 Lannion cedex, FRANCE
karim.bigou@inria.fr, arnaud.tisserand@irisa.fr

***Abstract*—The paper describes a new RNS (residue number system) modular multiplication algorithm, for finite field arithmetic over $\mathbb{F}_P$, based on a reduced number of moduli in base extensions with only $3n/2$ moduli instead of $2n$ for standard ones. Our algorithm reduces both the number of elementary modular multiplications (EMMs) and the number of stored precomputations for large asymmetric cryptographic applications such as elliptic curve cryptography or Diffie-Hellman (DH) cryptosystem. It leads to faster operations and smaller circuits.**

I. Introduction

High-performance arithmetic is required in asymmetric cryptographic applications such as RSA [1], *elliptic curve cryptography* (ECC) [2], [3] and *discrete logarithm* (DL) [4] with Diffie-Hellman [5] and Elgamal [6] cryptosystems. In these applications, operands range from a few hundred bits (160–550 in ECC) to a few thousand bits (1024–8192 in RSA and DL). Hence, reducing the computation time and silicon area is a major concern for hardware embedded systems.

The *residue number system* (RNS, [7], [8]) may be seen as a consequence of the Chinese remainder theorem (CRT) where large integers are represented by their remainders modulo small integers. There is no carry propagation between moduli, thus, some computations can be performed over all moduli in parallel or in random order [9]. An RNS base $\mathcal{B}$ consists of n moduli of w bits. RNS is increasingly used in asymmetric cryptographic research, see Tab. I. Typical numbers of moduli n range from a few moduli (for small ECC) to several dozens (for large RSA or DL).

Modular multiplication is the most costly frequent operation in cryptographic applications. In this paper, we propose a new *RNS modular multiplication algorithm* over $\mathbb{F}_P$ with a *reduced number of moduli in base extensions*. Standard base extensions algorithms use $2n$ moduli ([10], [11], [12], [13]). Our method only uses $\frac{3n}{2}$ moduli, reducing data storage and number of operations for large cryptographic parameters. Sec. II presents the notations and definitions. Sec. III recalls state-of-art methods. Sec. IV details our method and some DL and ECC applications are presented in Sec. V.

II. Notations and Definitions

- $|X|_P$ is $X \bmod P$ with P an ℓ-bit prime
- $m_{a,i}$ a w-bit pseudo-Mersenne modulo, $m_{a,i} = 2^w - h_{a,i}$ with $h_{a,i} < 2^{\lfloor w/2 \rfloor}$ and $n = \lceil \ell/w \rceil$
- $\mathcal{B}_a, \mathcal{B}_b, \mathcal{B}_c$ 3 coprime RNS bases, with $\mathcal{B}_a = (m_{a,1}, \ldots, m_{a,n_a})$ composed of n_a moduli all $m_{a,i}$ pairwise co-primes (similar definitions for $\mathcal{B}_b$ and $\mathcal{B}_c$)
- $\overrightarrow{(X)_a}$ represents X in the RNS base $\mathcal{B}_a$, abbreviated by $\overrightarrow{X_a}$ when no confusion is possible, and defined by:
$$\overrightarrow{(X)_a} = (x_{a,1}, \ldots, x_{a,n_a}) \text{ where } x_{a,i} = |X|_{m_{a,i}} \quad (1)$$
- $M_a = \prod_{i=1}^{n_a} m_{a,i}, \quad T_{a,i} = \frac{M_a}{m_{a,i}}$,
$\overrightarrow{T_a} = \left(|T_{a,1}|_{m_{a,1}}, \ldots, |T_{a,n_a}|_{m_{a,n_a}}\right)$
- EMM a w-bit elementary modular multiplication (e.g. $|x_i \cdot y_i|_m$), EMW a w-bit elementary memory word (for storage)
- $\mathcal{B}_{a|b} = (m_{a,1}, \ldots, m_{a,n_a}, m_{b,1}, \ldots, m_{b,n_b})$ is the concatenation of $\mathcal{B}_a$ and $\mathcal{B}_b$, $\overrightarrow{(X)_{a|b}}$ is X in RNS base $\mathcal{B}_{a|b}$

III. State-of-Art

A. Residue Number System (RNS)

RNS has been proposed in [7], [8] and used in asymmetric cryptography (see Tab. I). A large integer X, with $0 \leqslant X < M_a$, is converted to a unique RNS representation $\overrightarrow{X_a}$ in base $\mathcal{B}_a$ by applying definition at Eqn. (1). The reverse conversion, from RNS to a standard representation, uses the CRT formula:

$$X = |X|_{M_a} = \left| \sum_{i=1}^{n_a} \left| x_{a,i} \cdot T_{a,i}^{-1} \right|_{m_{a,i}} \times T_{a,i} \right|_{M_a} .$$

Addition/subtraction and multiplication are efficient and simple in RNS ($\diamond \in \{+, -, \times\}$). For instance in base $\mathcal{B}_a$:

$$\overrightarrow{X_a} \diamond \overrightarrow{Y_a} = \left(|x_{a,1} \diamond y_{a,1}|_{m_{a,1}}, \ldots, |x_{a,n_i} \diamond y_{a,n_i}|_{m_{a,n_a}}\right) . \quad (2)$$

Computations modulo $m_{a,j}$ are independent from those performed on other moduli $m_{a,i}$ with $i \neq j$, whereas classical binary representation requires carry propagation. Then, an RNS multiplication requires n_a independent EMMs. In base $\mathcal{B}_a$, exact division by Z can be performed through a multiplication by $\overrightarrow{\left(Z_a^{-1}\right)_a} = (|Z_a^{-1}|_{m_{a,1}}, \ldots, |Z_a^{-1}|_{m_{a,n_a}})$ when Z is co-prime to M_a. All these RNS operations are performed modulo M_a in base $\mathcal{B}_a$ (a consequence of the CRT) and similar behavior holds for bases $\mathcal{B}_b$ and $\mathcal{B}_c$.

But RNS is a *non-positional representation*, thus comparisons, general division and modular reduction are complex and costly operations in RNS (for instance, see [10]).

reference	conference/journal	year	usage	implementation	elements sizes and RNS bases ($n \times w$)
[14]	CHES	2001	RSA	ASIC 250 nm	672, 1024, 2048, 4096 (22×32), (33×32), (66×32), $(66 \times 32\ \star)$
[15]	CHES	2008	RSA ECC	GPU 8800GTS	1024 $(16 \times 32\ \star)$, 2048 $(32 \times 32\ \star)$, 224 (7×32)
[16]	IEEE TCAS I	2009	ECC	FPGA Virtex E	160, 192, 224, 256 5 bases/$\mathbb{F}_P$ size, e.g. $(30 \times \{23, 28, 30, 35\})$
[17]	CHES	2010	ECC	FPGA Stratix I & II	160 (5×34), 192 (6×33), 256 (8×33), 384 (11×35), 521 (15×35)
[18]	IEEE TC	2012	RSA	ASIC 45 nm	1024 (33×32)
[19]	Comp. J.	2012	ECC	GPU 285GTX	224 (15×16)
[20]	Arith	2013	RSA	ASIC 250 nm	1024 (33×32), 4096 (65×64)
[21]	IEEE TVLSI	2013	ECC	FPGA VirtexE Virtex 2 Pro Stratix II	160, 192, 224, 256 3×56, $(3 \times 66$ & $4 \times 50)$, 4×58, 4×66

TABLE I
STATE OF ART ON RNS IN ASYMMETRIC CRYPTOGRAPHY ($\star$ DENOTES CRT BASED RSA AND N MEANS NO IMPLEMENTATION RESULT REPORTED).

B. Base Extension

To speed up operations such as RNS modular reduction, the *base extension* (BE) has been introduced in [10]. It converts $\overrightarrow{X_a}$ from base $\mathcal{B}_a$ into $\overrightarrow{X_b}$ in base $\mathcal{B}_b$ without intermediate conversion to a classical representation. Among state-of-art BE algorithms [12], [13], [11], we only consider [12] due to its high level of parallelism (but our idea can be extended to other methods). BE Algo. 1 approximates the CRT formula $X = \sum_{i=1}^{n_a} \left(\left| x_{a,i} \cdot T_{a,i}^{-1} \right|_{m_{a,i}} T_{a,i} \right) - q\, M_a$ where

$$q = \left\lfloor \sum_{i=1}^{n_a} \frac{\left| x_{a,i} \cdot T_{a,i}^{-1} \right|_{m_{a,i}}}{m_{a,i}} \right\rfloor = \left\lfloor \sum_{i=1}^{n_a} \frac{\xi_{a,i}}{m_{a,i}} \right\rfloor .$$

In BE Algo. 1 from [12], the `trunc` function approximates the division $\xi_{a,i}/m_{a,i}$ using only a few most significant bits of $\xi_{a,i}$ and 2^w instead of $m_{a,i}$. The approximation error is denoted $\varepsilon_{\max}$. If $X_a < (1 - \sigma_0) M_a$ with $\varepsilon_{\max} < \sigma_0$ then this approximation has no influence on the result and the output of Algo. 1 is $\overrightarrow{X_b}$. In the other case, σ_0 is set to 0 and the output is either $\overrightarrow{X_b}$ or $\overrightarrow{(X + M_a)_b}$. See [12] for all details regarding the approximation. This algorithm requires $(n_a\, n_b + n_a)$ EMMs.

Algorithm 1: Base extension (BE) from [12]

```
Input: $\overrightarrow{X_a}$, $\mathcal{B}_a$, $\mathcal{B}_b$, $\sigma_0$ (fixed as a global parameter)
Precomp.: $\overrightarrow{(T_a^{-1})_a}$, $\overrightarrow{(T_a)_b}$, $\overrightarrow{(-M_a)_b}$
Output: $\overrightarrow{X_b}$
1 $\overrightarrow{\xi_a} = \overrightarrow{X_a} \times \overrightarrow{(T_a^{-1})_a}$,   $\overrightarrow{X_b} = \overrightarrow{0_b}$,   $\sigma = \sigma_0$
2 for $i = 1, \dots, n_a$ do
3     $\sigma = \sigma + \mathtt{trunc}(\xi_{a,i})$
4     $q = \lfloor \sigma \rfloor$          /* Comment: q is 0 or 1 */
5     $\sigma = \sigma - q$
6     for $j = 1, \dots, n_b$ do
7         $x_{b,j} = |x_{b,j} + \xi_{a,i} \cdot T_{a,i} + q \cdot (-M_a)|_{m_{b,j}}$
8 return $\overrightarrow{X_b}$
```

C. RNS Modular Multiplication (RNS-MM)

State-of-art RNS modular multiplication is based on the Montgomery modular multiplication [22], its RNS adaptation was initially proposed in [23] and optimized in [24], [12], [18].

Algorithm 2: RNS Montgomery Reduction from [23]

```
Input: $(\overrightarrow{X_a}, \overrightarrow{X_b})$
Precomp.: $(\overrightarrow{P_a}, \overrightarrow{P_b})$,   $\overrightarrow{(-P^{-1})_a}$,   $\overrightarrow{(M_a^{-1})_b}$
Output: $\overrightarrow{S} = \overrightarrow{\left| X |M^{-1}|_P \right|_P} + \delta \overrightarrow{P}$ in $\mathcal{B}_a$ and $\mathcal{B}_b$
        with $\delta \in \{0, 1, \dots, \alpha - 1\}$
1 $\overrightarrow{Q_a} \leftarrow \overrightarrow{X_a} \times \overrightarrow{(-P^{-1})_a}$
2 $\overrightarrow{Q_b} \leftarrow \mathrm{BE}\left(\overrightarrow{Q_a}, \mathcal{B}_a, \mathcal{B}_b\right)$
3 $\overrightarrow{R_b} \leftarrow \overrightarrow{X_b} + \overrightarrow{Q_b} \times \overrightarrow{P_b}$
4 $\overrightarrow{S_b} \leftarrow \overrightarrow{R_b} \times \overrightarrow{(M_a^{-1})_b}$
5 $\overrightarrow{S_a} \leftarrow \mathrm{BE}\left(\overrightarrow{S_b}, \mathcal{B}_b, \mathcal{B}_a\right)$
6 return $(\overrightarrow{S_a}, \overrightarrow{S_b})$
```

RNS modular reduction (RNS-MR) Algo. 2 mainly differs from the standard Montgomery reduction by the use of 2 BEs to be able to divide by $\mathcal{B}_a$. Usually, both bases have n elements ($n_a = n_b = n$). Thus, modular multiplication is computed using an RNS multiplication followed by a reduction with Algo. 2. [18] presents optimizations leading to an RNS Montgomery multiplication with $2\, n_a\, n_b + 2 n_a + 2 n_b = 2\, n^2 + 4\, n$ EMM. The output of RNS-MR is the RNS representation of $S < \alpha P$, with $\alpha \geqslant 3$, with α an adjustable parameter to perform more complex operations (e.g. $AB + CD \bmod P$).

IV. PROPOSED RNS MODULAR MULTIPLICATION ALGORITHM

The objective of our method is to, in a first time, decompose the operands into 2 sub-values, and, in a second time, use them in such a way only $3n/2$ moduli are required instead of $2n$. This leads to reduce the number of elementary computations and/or data storages in some operation patterns. In case of a square, the splitting step is performed only once. Same thing applies in case of multiplication by a constant. For instance, in the Montgomery powering ladder (see [25]), a loop with the pattern $A \leftarrow BC, D \leftarrow B^2$ is performed (the decomposition of B is reused for BC as well as for B^2).

Our RNS modular multiplication (named SPRR) is divided into 3 steps: splitting (Split), partial reduction (PR) and final reduction (R). Let us assume $|XY|_P$ is computed (or $|XYM|_P$ as for Montgomery multiplication). Split performs a specific decomposition of X and Y into 2 sub-values (see Sec. IV-A). PR uses these decompositions to compute a partially reduced value which is finally reduced using RNS Montgomery modular reduction. The complete SPRR is detailed at Sec. IV-B.

A. The Splitting Step

In this step, operands X and Y are split into their quotient/remainder by M_a using Algo. 3 such that $X = K_x M_a + R_x$ and $Y = K_y M_a + R_y$. The first BE (line 1) converts the remainder from $\mathcal{B}_a$ to the other bases (we recall $\overrightarrow{X_a} = \overrightarrow{(R_x)_a}$). Line 2 computes the quotient in bases $\mathcal{B}_b$ and $\mathcal{B}_c$. Finally, the second BE (line 3) brings back the quotient into the first base $\mathcal{B}_a$ from $\mathcal{B}_b$. Because the output of BE is approximated (see Sec. III-B), we add small additional constraints on the size of M_b and M_c (included in Prop. 1, Sec. IV-B).

To convert K_x back to $\mathcal{B}_a$ using Algo. 1, the constraints presented in Sec. III-B have to be respected to ensure an exact result. Using base $\mathcal{B}_b$ at line 3 gives $(1-\sigma_0)M_b > \frac{\alpha P}{M_a}$. The same type of constraint occurs when $\mathcal{B}_c$ is chosen instead of $\mathcal{B}_b$ in a symmetric way.

```
Algorithm 3: Proposed Splitting Step (Split)
  Input: $\overrightarrow{X_{a|b|c}}$ , $X < \alpha P$
  Precomp.: $\overrightarrow{(M_a^{-1})_{b|c}}$
  Output: $\overrightarrow{(K_x)_{a|b|c}}$ , $\overrightarrow{(R_x)_{a|b|c}}$ with
          $\overrightarrow{X_{a|b|c}} = \overrightarrow{(K_x)_{a|b|c}} \times \overrightarrow{(M_a)_{a|b|c}} + \overrightarrow{(R_x)_{a|b|c}}$
1 $\overrightarrow{(R_x)_{b|c}} \leftarrow$ BE$\left(\overrightarrow{(R_x)_a}, \mathcal{B}_a, \mathcal{B}_{b|c}\right)$
2 $\overrightarrow{(K_x)_{b|c}} \leftarrow \left(\overrightarrow{X_{b|c}} - \overrightarrow{(R_x)_{b|c}}\right) \times \overrightarrow{(M_a^{-1})_{b|c}}$
3 $\overrightarrow{(K_x)_a} \leftarrow$ BE$\left(\overrightarrow{(K_x)_b}, \mathcal{B}_b, \mathcal{B}_a\right)$
4 return $\overrightarrow{(K_x)_{a|b|c}}$ , $\overrightarrow{(R_x)_{a|b|c}}$
```

The computation performed in base $\mathcal{B}_b$ for the multiplication by $\overrightarrow{(M_a^{-1})_{b|c}}$ at line 2 of Algo. 3 can be combined with the one also in base $\mathcal{B}_b$ by $\overrightarrow{(T_b^{-1})_b}$ at line 1 of Algo. 1 (during the second BE at line 3 of Algo. 3). This combination saves n_b EMMs. As indicated in Tab. II, $\overrightarrow{(M_a^{-1}T_b^{-1})_b}$ is then precomputed.

Thus, the total cost of Split is $(n_a + n_a(n_b + n_c)) + (n_b + n_b n_a) + n_c$ EMMs. In the case $n_a = n_b = n_c = n/2$, this cost is $\frac{3}{4}n^2 + \frac{3}{2}n$ EMMs.

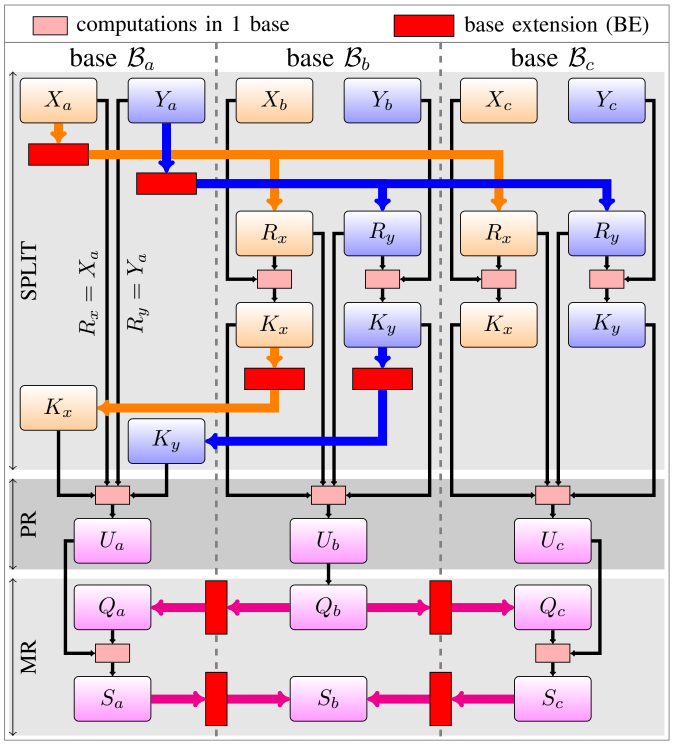

Fig. 1. Computation flow in our SPRR Algo. 4.

The Split step behaves as the decomposition in Karatsuba multiplication [26] (leading to 3 sub-products instead of 4 without decomposition). But Karatsuba method cannot be used in RNS since it is a non-positional representation.

B. SPRR Modular Multiplication Algorithm

Our method is presented in Algo. 4 and illustrated on Fig. 1.

```
Algorithm 4: Proposed Modular Multiplication (SPRR)
  Input: $\overrightarrow{X_{a|b|c}}$ , $\overrightarrow{Y_{a|b|c}}$ ,    $X, Y < \alpha P$
  Precomp.: $D = |M_a^{-1}|_P$
  Output: $\overrightarrow{V_{a|b|c}}$ , $V \equiv \left|XYM_a^{-1}M_b^{-1}\right|_P$, $V < \alpha P$
1 $\left(\overrightarrow{(K_x)_{a|b|c}}, \overrightarrow{(R_x)_{a|b|c}}\right) \leftarrow$ Split$\left(\overrightarrow{X_{a,b,c}}\right)$ ,
  $\left(\overrightarrow{(K_y)_{a|b|c}}, \overrightarrow{(R_y)_{a|b|c}}\right) \leftarrow$ Split$\left(\overrightarrow{Y_{a,b,c}}\right)$
2 $\overrightarrow{U_{a|b|c}} \leftarrow$
  PR$\left(\overrightarrow{(K_x)_{a|b|c}}, \overrightarrow{(R_x)_{a|b|c}}, \overrightarrow{(K_y)_{a|b|c}}, \overrightarrow{(R_y)_{a|b|c}}, D\right)$
3 $\overrightarrow{V_{a|b|c}} \leftarrow$ RNS-MR$\left(\overrightarrow{U_b}, \overrightarrow{U_{a|c}}\right)$
4 return $\overrightarrow{V_{a|b|c}}$
```

After the decomposition, one has:

$$|X\,Y|_P = \left|K_x K_y M_a^2 + (K_x R_y + K_y R_x)M_a + R_x R_y\right|_P . \tag{3}$$

Hypothesis 1 (H_1). *$D = |M_a^{-1}|_P$ and M_a have approximately the same size, i.e. $\exists \mu$, with μ small and such that $\mu P + 1 = M_a \times D$.*

Under hypothesis H_1, we defined U as

$$U = K_x K_y M_a + (K_x R_y + K_y R_x) + R_x R_y D + \alpha P \equiv |X\,Y D|_P\,. \quad (4)$$

If we choose $n_a = n/2$ then the sizes of M_a, R_x, K_x and D are very close to $\ell/2$ under H_1. One gets $\log_2 U \approx \frac{3}{2}\log_2 P$ under H_1. More precisely, assuming $C(\alpha,\mu) = \frac{\alpha^2}{\mu}D + 5\alpha + 4\mu M_a$, we have:

$$0 \leqslant U < C(\alpha,\mu)P + 4M_a. \quad (5)$$

This explains why U can be seen as a partially reduced value of $|XYD|_P$. Then, the computation of U is called *partial reduction* (PR). Its cost is the evaluation of U in the 3 bases: 4 multiplications and 2 multiplications by constants M_a and D in Eqn. 4. Multiplying by M_a in $\mathcal{B}_a$ is equivalent to multiply by 0, then these 6 multiplications in all bases require $4n_a + 6(n_b + n_c) = 8n$ `EMMs`. This cost can be reduced to $7n$ `EMMs` using Karatsuba's method [26] on bases $\mathcal{B}_b$ and $\mathcal{B}_c$ (there is no impact on the first base):

$$\begin{aligned} U &= K_x K_y M_a + (K_x K_y + R_x R_y - (K_x - R_x)(K_y - R_y)) \\ &\quad + R_x R_y D + \alpha P \\ &= K_x K_y (M_a + 1) + R_x R_y (D+1) \\ &\quad - (K_x - R_x)(K_y - R_y) + \alpha P. \end{aligned} \quad (6)$$

For the square operation, one multiplication in $\mathcal{B}_a$ can be saved and the cost is $6.5n$ `EMMs`. In the operation pattern (XY, XZ), the operations Split(X), $K_x(M_a+1)$ and $R_x(D+1)$ are performed only once, then its cost is $4.5n$ `EMMs`. In case of a multiplication by a constant Y, factorizations by K_x and R_x occur in Eqn. 4, leading to $3n$ `EMMs`.

The last step is the final modular reduction of U represented on $\frac{3}{2}n$ moduli using Algo. 2 (where the first base is $\mathcal{B}_b$ and the second one is $\mathcal{B}_{a|c}$ due to the choice of $\mathcal{B}_b$ in the Split see second paragraph of Sec. IV-A, this saves $(n^2/4) + (n/2)$ words of w-bit precomputations). Its cost is $2n_b(n_a + n_c) + 2(n_a + n_c) + nb = n^2 + \frac{5}{2}n$ `EMMs`. It takes into account some optimizations from [17], [18].

We have a complete proof for Proposition 1 based on some inequalities rearrangements. But due to paper length limitation, we cannot reproduce it here (ask the authors if needed).

Proposition 1 (under H_1)**.** *P is a large prime, co-prime with the 3 RNS bases $\mathcal{B}_a$, $\mathcal{B}_b$, $\mathcal{B}_c$; and M_b is selected to bring back K_x in $\mathcal{B}_a$ without approximation error in Split. If $M_b > \frac{1}{\alpha-2}(C(\alpha,\mu)+1)$, $M_c > \frac{(\alpha-2)P}{M_a}$ and $X, Y < \alpha P$ then $\overrightarrow{V_{a|b|c}}$, the output of Algo. 4, is equal to $|XYM_a^{-1}M_b^{-1}|_P$ with $V < \alpha P$.*

This result can be easily adapted to the hypothesis below.

Hypothesis 2 (H_2)**.** *$D = |-M_a^{-1}|_P$ and M_a have approximately the same size, i.e. $\exists\mu$, with μ small and such that $\mu P - 1 = M_a \times D$.*

Under hypothesis H_2, we defined U as

$$\begin{aligned} U &= K_x K_y M_a + (K_x R_y + K_y R_x) - R_x R_y D + (\alpha + M_a)P \\ &\equiv |X\,Y(-D)|_P\,. \end{aligned}$$

The added term $M_a P$ ensures U to be non negative. The only one difference in Prop. 1 using H_2 instead of H_1 is the size of M_b, now $M_b > \frac{1}{\alpha-2}(C(\alpha,\mu) + M_a + 1)$.

As in Montgomery modular multiplication, operands must be converted into $X' = |XM_aM_b|_P$ and $Y' = |YM_aM_b|_P$, then SPRR result is $|XYM_aM_b|_P$ under H_1 and H_2.

C. Parameters Selection

Condition H_1 (or H_2) is not strictly required in Prop. 1. But it strongly impacts the size of U and then the total cost. The condition H_1 restricts the choice of parameters in some cryptosystems. Up to now, we do not have a selection method to choose a base from a chosen P to satisfy H_1. Then, we cannot choose parameters proposed by NIST for ECC (which was especially designed to be efficient in the standard representation, e.g. $P_{521} = 2^{521} - 1$).

But, one can generate P such as $M_a D - 1 = \mu P$, with μ a small integer. In practice, we can select $\mu = 1$, try random values for D (of size $\approx \ell/2$) and test the primality of $M_a D - 1$. This method works to generate DL parameters and new P for ECC, optimized for the use of RNS implementation similarly to NIST primes for the binary representation.

For the final modular reduction, one can start from any of the 3 bases. Due to the precomputations savings mentioned above, base $\mathcal{B}_b$ is preferred to $\mathcal{B}_c$ and choosing $\mathcal{B}_b$ instead of $\mathcal{B}_a$ allows a wider parameter space (starting from M_a limits the number of couples (α,μ)). Starting from $\mathcal{B}_b$, operands are represented by $|XM_aM_b|_P$ (the same for Y) which is the usual RNS Montgomery representation.

In a very detailed analysis, one can notice that the size of M_b is longer than the one of M_a of two bits or more (for $\mu = 1, \alpha = 3 : M_b > (9D + 4M_a + 12)$). These few additional bits can be easily obtained (e.g. one modulo in $\mathcal{B}_a$ with $w-1$ bits and one modulo in $\mathcal{B}_b$ with $w+1$ bits). This kind of effect is neglected in the operation count (we assume multiplications of w, $w-1$ and $w+1$ bits operands all have the same cost).

V. Applications

The theoretical performances of our method (SPRR) are compared to state-of-art RNS modular multiplication (RNS-MM). We analyze the required number of elementary computations and data storage for both solutions. Only the number of elementary modular multiplications (w-bit) `EMMs` and elementary memory words `EMWs` are compared. The product `EMM` $\times$ `EMW` is also used as a common global cost metric for hardware implementation. Operations are directly written in the Montgomery representation (i.e. $|AB|_P$ is in fact $|ABM|_P$ with $M = M_aM_b$).

First, basic operations $|AB|_P$, $|A^2|_P$ and $|cst \times A|_P$ cost $2n^2 + 4n$ with RNS-MM (cst is a constant number). Using SPRR, it costs $\frac{5}{2}n^2 + \frac{25}{2}n$, $\frac{7}{4}n^2 + \frac{21}{2}n$ and $\frac{7}{4}n^2 + 7n$ respectively. Using several patterns in one larger sequence of operations, due to factorizations and reuse, the total cost can be reduced compared to the sum of the individual ones.

Split	$\overrightarrow{\left(T_a^{-1}\right)_a}$, $\overrightarrow{(T_{a,i})_{b\|c}}$, $\overrightarrow{(M_a)_{b\|c}}$,$\overrightarrow{\left(M_a^{-1}\right)_c}$, $\overrightarrow{\left(M_a^{-1}T_b^{-1}\right)_b}$, $\overrightarrow{\left(T_{b,i}\right)_a}$, $\overrightarrow{(-M_b)_a}$: $3n^2/4+3n$
PR	$\overrightarrow{D_{a\|b\|c}}$, $\overrightarrow{P_{a\|b\|c}}$: $3n$
RNS-MR	$\overrightarrow{\left(-P^{-1}T_b^{-1}\right)_b}$, $\overrightarrow{\left(T_{b,i}\right)_c}$, $\overrightarrow{(-M_b)_c}$, $\overrightarrow{\left(M_b^{-1}T_{a\|c}^{-1}\right)_{a\|c}}$, $\overrightarrow{\left(T_{a\|c,i}\right)_b}$,$\overrightarrow{\left(M_{a\|c}\right)_b}$: $3n^2/4+5n/2$
Total	$3n^2/2+17n/2$

TABLE II
MEMORY COUNT DETAILS IN w-BIT WORDS (EMWS)

The total number of precomputed memory words is $\frac{3}{2}n^2 + \frac{17}{2}n$ for SPRR (see Tab. II), and $2n^2 + 10n$ for RNS-MM (from [18]). For large fields, 25 % improvement is achieved due to the reduction from $2n$ to $3n/2$ moduli in the RNS basis. Moreover, even small RNS bases such as $n = 5$ lead to 20% memory reduction.

A. Discrete Logarithm in RNS

Exponentiations in $\mathbb{F}_P$ are common in cryptography such as in Diffie-Hellman [5] and Elgamal [6] cryptosystems. Both are based on the discrete logarithm problem in finite fields [27]. P is a 1024–3072 bits prime with $P-1 = VQ$ and Q a 160–256 bits prime. In fact this condition corresponds to H_2, choosing $\mu = 1$ and D as a multiple of Q: $P - 1 = M_aD = M_aV'Q = VQ$. Thus P can be generated under H_2 and be used for DL. Exponentiations of G are then computed with G a generator of the subgroup of order Q of $\mathbb{F}_p^*$.

Fig. 2 presents the results for both Montgomery ladder [25] and least significant bit first (LSBF) exponentiation algorithms. For this type of application, our method speeds up the computation (EMM) and reduces the global implementation cost (EMM $\times$ EMW). For instance, for the Montgomery ladder, there is a computation cost of $3.5n^2 + 19n$ EMMs for SPRR against $4n^2 + 8n$ EMMs for RNS-MM. Moreover, the product EMM$\times$EMW gives $5.25n^4 + 58.25n^3 + 161.5n^2$ for SPRR against $8n^4 + 56n^3 + 80n^2$ for RNS-MM. Common sizes of 1024 and 2048 bits correspond to 33 and 66 moduli (see Tab. I), and lead to 4 % and 9 % number of EMMs reduction respectively, and up to 30 % reduction of the global cost.

B. Elliptic Curve Cryptography Formulas in RNS

The second application uses SPRR on ECC. We compare SPRR and RNS-MM costs for a set of fast formulas from the explicit formulas database [28] with the usual parameters short Weierstrass form and Jacobian coordinates, for details see [2]. Formulas are given for the point doubling, tripling and mixed addition (point addition with a precomputed point with $Z = 1$). The detailed results are in Tab. III.

These results are graphically illustrated in Fig. 3 where mADD stands for point mixed-addition, DBL doubling and TPL tripling. The bottom sub-figure corresponds to individual curve level operations (DBL, TPL, mADD). SPRR is more

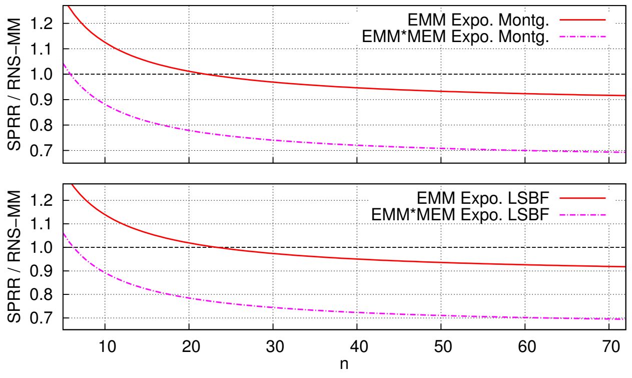

Fig. 2. Theoretical performance and cost comparison of SPRR and RNS-MM algorithms for LSBF and Montgomery ladder exponentiations.

interesting when operands are reused several times, this is why the gain is more important on DBL and TPL formulas than mADD. As an example, in doubling formulas a Split step is applied on Y_1 and Z_1 and then can be used to compute Y_1^2, Z_1^2 and $(Y_1 + Z_1)^2$ (lines 2, 4 and 10 respectively in Tab. III).

We selected 2 common curve level patterns: 2DBL+mADD and TPL+2DBL+mADD for scalar multiplication. The top sub-figure shows that considering the global cost (EMM$\times$EMW), our method is more efficient for $n > 5$. For bases with $n \geqslant 16$, one obtain more than 25 % global cost improvement using our method. Moreover, SPRR requires less EMMs than RNS-MM with $n > 16$. For instance, for the binary scalar multiplication, it leads to 4.5% and 9.5% of reduction for $n = 20$ and $n = 34$, respectively (for GPU implementations with $w = 16$ for instance, see [19]).

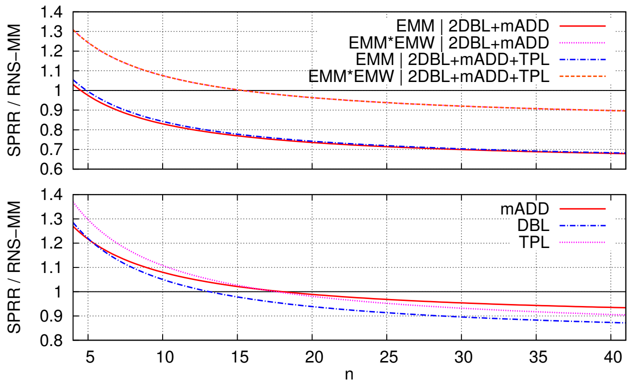

Fig. 3. Theoretical performance and cost comparison of SPRR and RNS-MM algorithms for various ECC formulas.

VI. Conclusion

A new RNS modular multiplication algorithm has been proposed. Thanks to a specific decomposition of the operands, some internal computations are shared and only use $3n/2$ moduli instead of $2n$ for standard methods. Memory requirements are reduced from 20 % to 25 % for some asymmetric cryptographic applications. For ECC and DH applications, the

Point operation	$\mathbf{P_1}+\mathbf{P_2}$ (mixed addition)	$2\,\mathbf{P_1}$	$3\,\mathbf{P_1}$
Formulas	$A_1 = Z_1^2,\, U_2 = X_2 A_1,\, S_2 = Y_2 Z_1 A_1$ $H = U_2 - X_1,\, H_2 = H^2,\, I = 4H_2$ $J = HI,\, R = 2(S_2 - Y_1),\, V = X_1 I$ $X_3 = R^2 - J - 2V$ $Y_3 = R(V - X_3) - 2Y_1 J$ $Z_3 = (Z_1 + H)^2 - A_1 - H_2$	$A = X_1^2,\, B = Y_1^2,\, C = B^2$ $D = Z_1^2,\, S = 2\left((X_1 + B)^2 - A - C\right)$ $M = 3A + aD^2,\, T = M^2 - 2S$ $X_3 = T$ $Y_3 = M(S - T) - 8C$ $Z_3 = (Y_1 + Z_1)^2 - B - D$	$A = X_1^2,\, B = Y_1^2,\, C = Z_1^2$ $D = B^2,\, M = 3A + aC^2,\, N = M^2$ $E = 6\left((X_1 + B)^2 - A - D\right) - N$ $F = E^2,\, T = 16D$ $U = (M + E)^2 - N - F - T$ $X_3 = 4(X_1 F - 4BU)$ $Y_3 = 8Y_1\left(U(T - U) - EF\right)$ $Z_3 = (Z_1 + E)^2 - C - F$
RNS-MM [EMM]	$20\,n^2 + 50n$	$20\,n^2 + 48\,n$	$28\,n^2 + 72\,n$
SPRR [EMM]	$17.5\,n^2 + 95\,n$	$16\,n^2 + 100.5\,n$	$23\,n^2 + 160\,n$
RNS-MM [EMM $\times$ EMW]	$40\,n^4 + 300\,n^3 + 500\,n^2$	$40\,n^4 + 296\,n^3 + 480\,n^2$	$56\,n^4 + 424\,n^3 + 720\,n^2$
SPRR [EMM $\times$ EMW]	$26.25\,n^4 + 291.25\,n^3 + 807.5\,n^2$	$24\,n^4 + 286.75\,n^3 + 854.25\,n^2$	$34.5\,n^4 + 435.5\,n^3 + 1360\,n^2$

TABLE III
FORMULAS FROM [28] (SHORT WEIERSTRASS, JACOBIAN COORDINATES).

number of operations can be reduced up to 10%. Hence, the total cost of an implementation using our algorithm is expected to be better than state-of-art solution for large field applications. Future prospects include more advanced parameters selection and full hardware implementation.

ACKNOWLEDGMENT

This work has been supported in part by a PhD grant from *DGA–INRIA* and PAVOIS project (ANR 12 BS02 002 01).

REFERENCES

[1] R. L. Rivest, A. Shamir, and L. Adleman, “A method for obtaining digital signatures and public-key cryptosystems,” *Communications of the ACM*, vol. 21, no. 2, pp. 120–126, Feb. 1978.

[2] D. Hankerson, A. Menezes, and S. Vanstone, *Guide to Elliptic Curve Cryptography*. Springer, 2004.

[3] H. Cohen and G. Frey, Eds., *Handbook of Elliptic and Hyperelliptic Curve Cryptography*. Chapman & Hall/CRC, 2005.

[4] A. Odlyzko, “Discrete logarithms: The past and the future,” *Designs, Codes and Cryptography*, vol. 19, no. 2/3, pp. 59–75, Mar. 2000, special Issue on Towards a Quarter-Century of Public Key Cryptography.

[5] W. Diffie and M. E. Hellman, “New directions in cryptography,” *IEEE Trans. on Information Theory*, vol. 22, no. 6, pp. 644–654, Nov. 1976.

[6] T. Elgamal, “A public key cryptosystem and a signature scheme based on discrete logarithms,” *IEEE Transactions on Information Theory*, vol. 31, no. 4, pp. 469–472, Jul. 1985.

[7] A. Svoboda and M. Valach, “Operátorové obvody (operator circuits in czech),” *Stroje na Zpracování Informací (Information Processing Machines)*, vol. 3, pp. 247–296, 1955.

[8] H. L. Garner, “The residue number system,” *IRE Transactions on Electronic Computers*, vol. EC-8, no. 2, pp. 140–147, Jun. 1959.

[9] J.-C. Bajard, L. Imbert, P.-Y. Liardet, and Y. Teglia, “Leak resistant arithmetic,” in *Proc. Cryptographic Hardware and Embedded Systems (CHES)*, ser. LNCS, vol. 3156. Springer, 2004, pp. 62–75.

[10] N. S. Szabo and R. I. Tanaka, *Residue arithmetic and its applications to computer technology*. McGraw-Hill, 1967.

[11] A. P. Shenoy and R. Kumaresan, “Fast base extension using a redundant modulus in RNS,” *IEEE Transactions on Computers*, vol. 38, no. 2, pp. 292–297, Feb. 1989.

[12] S. Kawamura, M. Koike, F. Sano, and A. Shimbo, “Cox-Rower architecture for fast parallel Montgomery multiplication,” in *Proc. 19th International Conference on the Theory and Application of Cryptographic (EUROCRYPT)*. Springer, May 2000, pp. 523–538.

[13] J.-C. Bajard, L.-S. Didier, and P. Kornerup, “Modular multiplication and base extensions in residue number systems,” in *Proc. 15th Symposium on Computer Arithmetic (ARITH)*. IEEE, Apr. 2001, pp. 59–65.

[14] H. Nozaki, M. Motoyama, A. Shimbo, and S. Kawamura, “Implementation of RSA algorithm based on RNS Montgomery multiplication,” in *Proc. Cryptographic Hardware and Embedded Systems (CHES)*, ser. LNCS, vol. 2162. Springer, May 2001, pp. 364–376.

[15] R. Szerwinski and T. Guneysu, “Exploiting the power of GPUs for asymmetric cryptography,” in *Proc. 10th International Workshop on Cryptographic Hardware and Embedded Systems (CHES)*, ser. LNCS, vol. 5154. Springer, Aug. 2008, pp. 79–99.

[16] D. M. Schinianaki, A. P. Fournaris, H. E. Michail, A. P. Kakarountas, and T. Stouraitis, “An RNS implementation of an $\mathbb{F}_p$ elliptic curve point multiplier,” *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 56, no. 6, pp. 1202–1213, Jun. 2009.

[17] N. Guillermin, “A high speed coprocessor for elliptic curve scalar multiplications over $\mathbb{F}_p$,” in *Proc. Cryptographic Hardware and Embedded Systems (CHES)*, ser. LNCS, vol. 6225. Springer, Aug. 2010, pp. 48–64.

[18] F. Gandino, F. Lamberti, G. Paravati, J.-C. Bajard, and P. Montuschi, “An algorithmic and architectural study on Montgomery exponentiation in RNS,” *IEEE Transactions on Computers*, vol. 61, no. 8, pp. 1071–1083, Aug. 2012.

[19] S. Antao, J.-C. Bajard, and L. Sousa, “RNS-based elliptic curve point multiplication for massive parallel architectures,” *The Computer Journal*, vol. 55, no. 5, pp. 629–647, May 2012.

[20] J.-C. Bajard, J. Eynard, and F. Gandino, “Fault detection in RNS Montgomery modular multiplication,” in *Proc. 21th Symposium on Computer Arithmetic (ARITH)*. IEEE, Apr. 2013, pp. 119–126.

[21] M. Esmaeildoust, D. Schinianakis, H. Javashi, T. Stouraitis, and K. Navi, “Efficient RNS implementation of elliptic curve point multiplication over GF(p),” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 21, no. 8, pp. 1545–1549, Aug. 2013.

[22] P. L. Montgomery, “Modular multiplication without trial division,” *Mathematics of Computation*, vol. 44, no. 170, pp. 519–521, Apr. 1985.

[23] K. C. Posch and R. Posch, “Modulo reduction in residue number systems,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 6, no. 5, pp. 449–454, May 1995.

[24] J.-C. Bajard, L.-S. Didier, and P. Kornerup, “An RNS Montgomery modular multiplication algorithm,” *IEEE Transactions on Computers*, vol. 47, no. 7, pp. 766–776, Jul. 1998.

[25] M. Joye and S.-M. Yen, “The Montgomery powering ladder,” in *Proc. International Workshop on Cryptographic Hardware and Embedded Systems (CHES)*. Springer, Aug. 2002, pp. 291–302.

[26] A. Karatsuba and Y. Ofman, “Multiplication of multi-digit numbers on automata,” *Doklady Akad. Nauk SSSR*, vol. 145, no. 2, pp. 293–294, 1962, translation in Soviet Physics-Doklady, 44(7), 1963, p. 595-596.

[27] N. I. of Standards and T. (NIST), “FIPS 186-4, digital signature standard (DSS),” 2013.

[28] D. J. Bernstein and T. Lange, “Explicit-formulas database,” http://www.hyperelliptic.org/EFD, 2013, september update.