Study of Modular Inversion in RNS

Jean Claude Bajard, Nicolas Meloni, Thomas Plantard
LIRMM, UMR CNRS 5506, Montpellier, France

ABSTRACT

Residue Numbers System have some features useful in implementations of cryptographic protocols. The main property of RNS is the distribution of the evaluation on large values over its small residues, allowing parallelization. This last property implies that we can randomize the distribution of the bases elements. Hence, the resulting arithmetic is leak resistant, i.e. robust against side channel attacks. One drawback of RNS is that modular inversion is not obvious. Thus, RNS is well suited for RSA but not really for ECC. We analyze in this paper the features of the modular inversion in RNS over $GF(P)$. We propose a RNS Extended Euclidean Algorithm which uses a quotient approximation module.

Keywords: Residue Number Systems, Modular Inversion, Cryptography

1. INTRODUCTION

1.1. Context

Since the proposal of cryptography over elliptic curves by Miller and Koblitz[4, 6] , the arithmetic over finite field enables efficient hardware or software implementations.

The Elliptic Curves Cryptosystems ECC deals with the structure of group defined by the points of an elliptic curve over a finite field. The basic operation is the "addition" of two points of the curve, which is done by determining the intersection with the curve of the line defined by those two points. In affine coordinates over $GF(P)$, this operation is composed of two products, one square (two for doubling a point) and one inversion. Now if we consider homogeneous coordinates to avoid the inversion, we need: twelve products and one square (7 products and 5 squares for doubling). Hence, if the cost of an inversion is lower than that of six products we can assume that affine coordinates offer the best features for an implementation.

In the constext of side chanel attack it was shown[2, 3] that residue numbers systems offer the opportunity to randomize the bases and thus provide a leak resistant arithmetic. This approach gave good results for cryptosystems like RSA which is based on modular exponentiation and on modular multiplication. But, considering ECC, the drawback of the RNS arithmetic is the lack of an inversion. This problem is quite difficult, and rarely approached in the literature. Hence, we propose in this paper to deal with this RNS operation and give one RNS version of the Euclidean algorithm.

1.2. Residue Number Systems

The residue number systems (RNS) are very old.[5, 8, 9] The aim of this system is to represent a number by its residues over a set of coprime numbers $(m_1, ..., m_n)$, called RNS basis. For example, a number A has a residue a_i modulo m_i for each i from 1 to n. Thus the vector $(a_1, ..., a_n)$ will be called the RNS representation of A. We denote a corresponding digit-vector with $\overline{A}$, and $M = \prod_{n}^{i=1} m_i$. The Chinese Remainders Theorem assumes that each vector $\overline{A}$ defines one and only one value A such $0 \leq A < M$. Moreover, the following explicit formula enables to recover an integer from its RNS representation (we note that α is an integer smaller than n).

$$A = \left| \sum_{i=1}^{n} \left| a_i M_i^{-1} \right|_{m_i} M_i \right|_M = \sum_{i=1}^{n} \left| a_i M_i^{-1} \right|_{m_i} M_i - \alpha M \tag{1}$$

We can remark that this representation is not redundant: one vector defines one value.

E-mail: bajard,meloni,plantard@lirmm.fr, LIRMM, 61 rue ADA, Montpellier, 34000 France

Example: We consider here the set of coprimes $\mathcal{B} = (3, 7, 13, 19)$ as a RNS basis. Thus we can represent all the numbers from 0 to $M = 5187$

$$\begin{array}{llll} X = 147 & & Y = 31 & \\ \overline{X} = & (0,0,4,14) & \overline{Y} = & (1,3,5,12) \end{array}$$

The addition and the multiplication become:

$$\begin{array}{llllll} \overline{X} + \overline{Y} & = (& \|0+1\|_3 , & \|0+3\|_7 , & \|4+5\|_{13} , & \|14+12\|_{19} \quad) \\ & = (& 1 , & 3 , & 9 , & 7 \quad) \\ & = & 178 & & & \end{array}$$

$$\begin{array}{llllll} \overline{X} \times \overline{Y} & = (& \|0 \times 1\|_3 , & \|0 \times 3\|_7 , & \|4 \times 5\|_{13} , & \|14 \times 12\|_{19} \quad) \\ & = (& 0 , & 0 , & 7 , & 16 \quad) \\ & = & 4557 & & & \end{array}$$

Hence, the main advantage of this representation is due to the property of distributing the evaluation over the residues which are small values. We remark that it is fully parallel, i.e. there is no propagation of data through the moduli. The drawback of the RNS is that comparisons and overflow detections are hard. In RNS the operation are made modulo M.

The features of RNS are interesting for cryptography.[2] Some efficient modular multiplications have been proposed in the literature[1, 7] in particular for RSA.

Actually we could not find any RNS inversion algorithms for finite fields $GF(P)$, which could be useful for ECC. So we will focus our study on this operation in RNS .

2. EXTENDED EUCLIDEAN ALGORITHM IN RNS

In this section, we present the classical Extended Euclidean Algorithm (EEA).[5] Then we propose an RNS version of it, with points discussed in detail in the next section.

2.1. The classical algorithm

The Extended Euclidean Algorithm (EEA) is the classical Euclidean algorithm for computing the greatest common divisor (gcd) where we add some variables which satify an invariant equality coming from the Bezout identity.

```
Algorithm 1: ExtEucl($A, P$)
  Data: $A < P$ and coprime;
  Result: $A^{-1} \bmod P$;
  $(U_1, U_2, U_3) \leftarrow (0, 1, P)$;
  $(V_1, V_2, V_3) \leftarrow (1, 0, A)$;
  while $V_3 > 0$ do
      $q \longleftarrow \left\lfloor \frac{U_3}{V_3} \right\rfloor$ ;
      $(U_1, U_2, U_3) \longleftarrow (U_1, U_2, U_3) - q(V_1, V_2, V_3)$ ;
      $(U_1, U_2, U_3) \longleftrightarrow (V_1, V_2, V_3)$ ;
  end
  return $U_1$
```

In this algorithm the variables at each step satisfy:

$$U_1 A + U_2 P = U_3 \tag{2}$$

$$V_1 A + V_2 P = V_3 \tag{3}$$

We can verify that at each step V_3 decreases, since it is replaced by a remainder of Euclidean division by V_3. Then, when $V_3 = 0$, the Euclidean algorithm assumes that U_3 contains the gcd of A and P. When P is prime, this gcd is 1 and $U_1 A \equiv 1 \pmod{P}$. We could find a negative value for U_1, in this case we have to add P to obtain a positive representation of the inverse of A in $GF(P)$. The sign of U_1 is due to the number of exchanges in the algorithm, if this number is odd then U_1 is positive else it is negative..

Now we will consider this algorithm for numbers in RNS representations.

2.2. The RNS version

We note $\overline{A} = (a_1, ..., a_n)$ the RNS representation of A. The RNS Extended Euclidean Algorithm will merge the division to the gcd algorithm. In fact, since the quotient is estimated, we are not sure that the remainder is smaller than the divisor. So, some additional iterations could be necessary before the exchange.

Algorithm 2: `ExtEuclRNS`$(\overline{A}, \overline{P})$

```
  Data: $A < P < \frac{M}{4}$ with $P$ a prime number;
  Result: $\overline{A^{-1}}$ mod $P$;
  $(\overline{U_1}, \overline{U_3}) \leftarrow (\overline{0}, \overline{P})$;
  $(\overline{V_1}, \overline{V_3}) \leftarrow (\overline{1}, \overline{A})$;
  $t_u \leftarrow 0$ ; $t_v \leftarrow 0$; $\overline{2^{t_u}} \leftarrow (\overline{1})$; $\overline{2^{t_v}} \leftarrow (\overline{1})$;
  NormalSup$(\overline{U_3}, t_u, \widetilde{u}, \overline{2^{t_u}})$;
  NormalSup$(\overline{V_3}, t_v, \widetilde{v}, \overline{2^{t_v}})$;
  counter $\leftarrow 0$ ;
1 while $\overline{V_3} \neq 0$ do
      $(\overline{Q}, \widetilde{q}) \longleftarrow$ Estim$(\widetilde{u}, \widetilde{v}, t_u, t_v, \overline{2^{t_u}}, \overline{2^{t_v}})$ ;
2     if $\widetilde{q} > 0$ then
          $(\overline{U_1}, \overline{U_3}) \longleftarrow (\overline{U_1}, \overline{U_3}) - \overline{Q}(\overline{V_1}, \overline{V_3})$ ;
          NormalSup$(\overline{U_3}, t_u, \widetilde{u}, \overline{2^{t_u}})$;
      else
3         if Test$(\overline{U_3}, \overline{V_3})$ then
              $(\overline{U_1}, \overline{U_3}) \longleftarrow (\overline{U_1}, \overline{U_3}) - (\overline{V_1}, \overline{V_3})$ ;
              NormalSup$(\overline{U_3}, t_u, \widetilde{u}, \overline{2^{t_u}})$;
          end
4         $(\overline{U_1}, \overline{U_3}, t_u, \overline{2^{t_u}}) \longleftrightarrow (\overline{V_1}, \overline{V_3}, t_v, \overline{2^{t_v}})$ ;
          counter $\leftarrow$ counter $+ 1$ ;
      end
  end
  if counter is even then
      $\overline{U_1} \longleftarrow \overline{P} + \overline{U_1}$
  end
  return $\overline{U_1}$
```

The main loop 1 of this algorithm is the one of the Euclidean algorithm. At each iteration, to perform the division we estimate the quotient. If the obtained value is a positive non zero number, step 2, then we can reduce the value of U_3 and update all the associated variables, with a call of `NormalSup`. If the approximation is a zero value then, step 6, we compare U_3 to V_3. If U_3 is the greater, the quotient is , in this case, equal to 1, and $U_3 - V_3$ is smaller than V_3. We update the variables associated to U_3 by calling `NormalSup`. Then, we exchange the U values with the V values.

The convergence of the algorithm is due to the estimation of $(\widetilde{Q}, \widetilde{q})$ such that if $\widetilde{q} = 0$ then $U_3 < V_3$ or $U_3 - V_3 < V_3$. Thus we maintain at the end of each pass in the while loop, two facts: first that $U_3 > V_3$, and secondly, at least U_3 and/or V_3 are decreasing.

3. ESTIMATION AND COMPARISON

3.1. Approximation of U

In the Algorithm 2, we calculate an approximation of the quotient $\frac{U}{V}$ by estimating the values of $\frac{U}{M}$ and $\frac{V}{M}$. The main idea comes from the following equality :

$$U = \sum_{i=1}^{n} \left|u_i \times M_i^{-1}\right|_{m_i} \times M_i - \alpha M = \sum_{i=1}^{n} \frac{\lambda_i}{m_i} \times M - \alpha M \tag{4}$$

where α is an integer and $\lambda_i = \left|u_i \times M_i^{-1}\right|_{m_i}$. Now let E and F be such that $\sum_{i=1}^{n} \frac{\lambda_i}{m_i} = E + F$ where E is an integer and $0 \le F < 1$. Then from the equation (4) we have

$$\frac{U}{M} = \sum_{i=1}^{n} \frac{\lambda_i}{m_i} - \alpha = (E - \alpha) + F \tag{5}$$

Noticing that $(E - \alpha)$ is an integer and as $0 < \frac{U}{M} < 1$, we finally get that $E = \alpha$ and so, $F = \frac{U}{M}$. So, in order to evaluate $\frac{U}{M}$ we calculate an estimate of $\sum_{i=1}^{n} \frac{\lambda_i}{m_i}$ and then we take its fractional part.

Now let us assume that $n < 2^{l-2}$ and that $\frac{M}{8} < U \times 2^{t_u} < \frac{M}{2}$. Then the following algorithm returns an integer $\widetilde{u}$ which satifies :

$$\frac{U \times 2^{t_u}}{M} \times 2^{l-1} \le \widetilde{u} < \frac{U \times 2^{t_u}}{M} \times 2^{l-1} + n \tag{6}$$

Algorithm 3: ApproxSup($\overline{U}, \overline{2^{t_u}}$)

```
Precomputed: $\beta_i = \left\lceil \frac{2^{2l}}{m_i} \right\rceil$ ; $\overline{\mu}$ with $\mu_i = |M_i|_{m_i}^{-1}$; for $i = 1, ..., n$
$\overline{\lambda} \leftarrow \overline{U} \times \overline{\mu} \times \overline{2^{t_u}}$;
$\widetilde{u} \leftarrow 0$ ;
for $i \leftarrow 1$ to $n$ do
    $\widetilde{u} \leftarrow \widetilde{u} + \beta_i \times \lambda_i \bmod 2^{2l}$ ;
end
$\widetilde{u} \leftarrow \widetilde{u}$ div $2^l + 1$ ;
return $\widetilde{u}$
```

Proof : To simplify the proof, we consider the case where $\frac{M}{8} < U < \frac{M}{2}$, which corresponds to $t_u = 0$. It is easy to see that if $t_u \neq 0$ we can consider $U' = U \times 2^{t_u}$ instead of U.

We have $U = (u_1, ..., u_n)_{RNS}$ and for each $i \in \{1..n\}$, $\beta_i = \left\lceil \frac{2^{2l-1}}{m_i} \right\rceil$ and $0 \le \lambda_i \le 2^l$. Let $i \in \{1..n\}$:

$$\frac{2^{2l-1}}{m_i} \quad \le \beta_i \quad \le \frac{2^{2l-1}}{m_i} + 1 \tag{7}$$

$$\sum_{i=1}^{n} \lambda_i \times \frac{2^{2l-1}}{m_i} \quad \le \sum_{i=1}^{n} \lambda_i \times \beta_i \quad \le \sum_{i=1}^{n} \lambda_i \times \frac{2^{2l-1}}{m_i} + \sum_{i=1}^{n} \lambda_i \tag{8}$$

Now, let set $\sum_{i=1}^{n} \frac{2^{2l-1}}{m_i} = (E + \frac{U}{M}) \times 2^{2l-1}$ where E is an integer. Since, $\frac{U}{M} < \frac{1}{2}$, we have

$$(E + \frac{U}{M}) \times 2^{2l-1} \leq \sum_{i=1}^{n} \lambda_i \times \beta_i < (E + \frac{U}{M}) \times 2^{2l-1} + \sum_{i=1}^{n} \lambda_i \tag{9}$$

$$(E + \frac{U}{M}) \times 2^{2l-1} \leq \sum_{i=1}^{n} \lambda_i \times \beta_i < E \times 2^{2l-1} + 2^{2l-2} + n2^l \tag{10}$$

So, since $n < 2^{l-2}$, we get $2^{2l-2} + n2^l < 2^{2l-1}$. Thus

$$\left| \sum_{i=1}^{n} \lambda_i \times \beta_i \right|_{2^{2l-1}} = \sum_{i=1}^{n} \lambda_i \times \beta_i - E \times 2^{2l-1} \tag{11}$$

and finally

$$\frac{U}{M} \times 2^{2l-1} \leq \left| \sum_{i=1}^{n} \lambda_i \times \beta_i \right|_{2^{2l-1}} \leq \frac{U}{M} \times 2^{2l-1} + n2^l \tag{12}$$

A final division by 2^l gives the expected result.

3.2. Normalization

Thanks to the ApproxSup algorithm, we are able to estimate the first bits of $\frac{U \times 2^{t_u}}{M}$. To have as many significant bits as possible, we use the following normalization algorithm.

It consists in evaluating $t_u, \widetilde{u}, \overline{2^{t_u}}$ such that:

$$\frac{M}{8} < U \times 2^{t_u} < \frac{M}{2}$$

where $\widetilde{u}$ correspond to the result of ApproxSup.

Algorithm 4: NormalSup($\overline{U}, t_u, \widetilde{u}, \overline{2^{t_u}}$)

```
$\widetilde{u} \leftarrow$ ApproxSup($\overline{U}, \overline{2^{t_u}}$);
while $\widetilde{u} < 2^{e+2}$ do
    $t_u \leftarrow t_u + l - e - 4$;
    $\overline{2^{t_u}} \leftarrow 2^{l-e-4} \times \overline{2^{t_u}}$;
    $\widetilde{u} \leftarrow$ ApproxSup($\overline{U}, \overline{2^{t_u}}$);
end
$c \leftarrow 0$;
while $\widetilde{u} < 2^{l-3}$ do
    $t_u \leftarrow t_u + 1$;
    $\widetilde{u} \leftarrow \widetilde{u} << 1$;
    $c \leftarrow c + 1$;
end
if $c > 0$ then
    $\overline{2^{t_u}} \leftarrow 2^c \times \overline{2^{t_u}}$;
    $\widetilde{u} \leftarrow$ ApproxSup($\overline{U}, \overline{2^{t_u}}$);
end
```

Proof : We assume that $0 < U \times 2^{t_u} < \frac{M}{2}$ and we note $e = \left\lceil \frac{\log n}{\log 2} \right\rceil$.

First, let us suppose that $\widetilde{u} < 2^{e+2}$, then

$$\begin{aligned}\frac{U}{M} \times 2^{t_u} \times 2^{l-1} &\leq \widetilde{u} \\ U \times 2^{t_u} \times 2^{l-1} &< M \times 2^{e+2} \\ U \times 2^{t_u} \times 2^{l-e-4} &< \frac{M}{2}\end{aligned}$$

So, at the end of each iteration of the first while loop, as t_u is updated to $t_u + l - e - 4$, we still have $U \times 2^{t_u} < \frac{M}{2}$. At the end, we get $2^{e+2} \leq \widetilde{u} < 2^{l-2}$ and

$$\frac{U}{M} \times 2^{t_u} \times 2^{l-1} \leq \widetilde{u} < \frac{U}{M} \times 2^{t_u} \times 2^{l-1} + 2^e \tag{13}$$

We consider the second loop which construct c such that: $2^{l-3} \leq \widetilde{u} \times 2^c < 2^{l-2}$.

$$2^{l-3} < \widetilde{u} \times 2^c < \frac{U}{M} \times 2^c \times 2^{t_u} \times 2^{l-1} + 2^e \times 2^c$$

$$2^{-3} < \frac{U}{M} \times 2^c \times 2^{t_u} \times 2^{-1} + 2^e \times 2^c \times 2^{-l}$$

As $\widetilde{u} \times 2^c \geq 2^{e+2} \times 2^c$, we have $e + c + 2 \leq l - 2$.

Then

$$2^{-3} < \frac{U}{M} \times 2^c \times 2^{t_u} \times 2^{-1} + 2^{-4}$$

Finally, we get,

$$2^{-4} = 2^{-3} - 2^{-4} < \frac{U}{M} \times 2^c \times 2^{t_u} \times 2^{-1}$$

or,

$$2^{-3} < \frac{U}{M} \times 2^c \times 2^{t_u}$$

Similarly, we have

$$\frac{U}{M} \times 2^{t_u} \times 2^c \times 2^{l-1} \leq \widetilde{u} \times 2^c \leq 2^{l-2}$$

which gives,

$$\frac{U}{M} \times 2^{t_u} \times 2^c \leq 2^{-1}$$

So, we get , with $t_u = t_u + c$,

$$\frac{1}{8} < \frac{U}{M} \times 2^{t_u} < \frac{1}{2} \tag{14}$$

3.3. Estimation of the quotient

The procedure Estim computes an approximation of the quotient $\left\lfloor \frac{U}{V} \right\rfloor$ such that the returned value is zero only if $\left\lfloor \frac{U}{V} \right\rfloor < 2$. In other cases, when $t_v - t_u - (l-2) \geq 0$, the returned value $\widetilde{q}$ is such that $2^{l-5} < \widetilde{q} < 2^l$. This assumes that the division is as efficient as a division in radix 2^{l-4}.

Algorithm 5: Estim$(\widetilde{u}, \widetilde{v}, t_u, t_v, \overline{2^{t_u}}, \overline{2^{t_v}})$

```
if $t_v - t_u - (l-2) \geq 0$ then
    $s \longleftarrow l-2$
else
    $s \longleftarrow t_v - t_u$
end
$\widetilde{q} \longleftarrow \frac{(\widetilde{u}-n)2^s}{\widetilde{v}}$;
$\widetilde{Q} \longleftarrow \widetilde{q} \times \overline{2^{t_v-t_u-(s)}}$ ;
return $(\widetilde{Q}, \widetilde{q})$;
```

Proof. The proof of this algorithm is in two part. First we show that $2^{s-3} < \widetilde{q} < 2^{s+2}$. For this we consider that:

$$\frac{M}{8} < U \times 2^{t_u} < \frac{M}{2} \text{ and } \frac{M}{8} < V \times 2^{t_v} < \frac{M}{2} \tag{15}$$

Then we obtain,

$$2^{-2} < \frac{U}{V} \times 2^{t_u - t_v} < 2^2 \tag{16}$$

Now we try to bound $\widetilde{q}$. For this, we use the equation defined by ApproxSup where we note $\widetilde{u} = \widetilde{u} - n$ to simplify the notations.

$$\frac{U}{M} \times 2^{t_u+l-1} - n \leq \widetilde{u} < \frac{U}{M} \times 2^{t_u+l-1} \tag{17}$$

$$\frac{V}{M} \times 2^{t_v+l-1} \leq \widetilde{v} < \frac{V}{M} \times 2^{t_v+l-1} + n \tag{18}$$

Thus we obtain for the quotient:

$$\frac{U \times 2^{t_u+l-1} - nM}{V \times 2^{t_v+l-1} + nM} \leq \frac{\widetilde{u}}{\widetilde{v}} \leq \frac{U}{V} 2^{t_u - t_v} \tag{19}$$

$$\frac{U - nM \times 2^{-(t_u+l-1)}}{V + nM \times 2^{-(t_v+l-1)}} 2^{t_u - t_v} \leq \frac{\widetilde{u}}{\widetilde{v}} \leq \frac{U}{V} 2^{t_u - t_v} \tag{20}$$

Now we use that : $\frac{M}{8} < U \times 2^{t_u} < \frac{M}{2}$ and $\frac{M}{8} < V \times 2^{t_v} < \frac{M}{2}$ As, for each i, $2^{l-1} < m_i < 2^l$ and $n < 2^e$, the equation (20) becomes:

$$\frac{1 - 2^{e+3-(l-1)}}{1 + 2^{e+3-(l-1)}} \times \frac{U}{V} \times 2^{t_u - t_v} \leq \frac{\widetilde{u}}{\widetilde{v}} \leq \frac{U}{V} \times 2^{t_u - t_v} \tag{21}$$

We know that $\frac{1}{1+2^{e+3-(l-1)}} < 1 - 2^{e+3-(l-1)}$ when $e + 3 - (l-1) \leq -2$, in other words, when $l \geq e + 6$.
Hence, the equation is:

$$(1 - 2^{e+3-(l-1)})^2 \times \frac{U}{V} \times 2^{t_u - t_v} \leq \frac{\widetilde{u}}{\widetilde{v}} \leq \frac{U}{V} \times 2^{t_u - t_v} \tag{22}$$

But, if $l \geq e + 6$, then $(1 - 2^{e+3-(l-1)})^2 > \frac{1}{2}$. We deduce that:

$$\left\lfloor \frac{U}{V} \times 2^{t_u - t_v + s - 1} \right\rfloor \leq \widetilde{q} \leq \left\lfloor \frac{U}{V} \times 2^{t_u - t_v + s} \right\rfloor \tag{23}$$

And,

$$\left\lfloor 2^{-1} \times \frac{U}{V} \right\rfloor \leq \widetilde{Q} \leq \left\lfloor \frac{U}{V} \right\rfloor \tag{24}$$

If $s = l - 2$ and $l \geq e + 6$, we obtain $2^{l-5} \leq \widetilde{q} \leq 2^{l}$.

If $s = t_v - t_u$, then $\widetilde{Q} = \widetilde{q}$. In this case, $\widetilde{q}$ can be equal to zero when $(1 - 2^{e+3-(l-1)})^2 \times \frac{U}{V} < 1$, in other words $\frac{U}{V} < (1 - 2^{e+3-(l-1)})^{-2}$ which is lower than 2 since $e + 3 - (l - 1) \leq -3$ that is to say $l > e + 7$. Hence, if $l > e + 7$, we assume that `Estim` gives a zero value only when $U < 2 \times V$. □

3.4. Sign of a difference

In the RNS Euclidean Algorithm when $\widetilde{q}$, the approximation of the quotient given by `Estim` is zero, we know that $U < 2V$. Thus, if $U \geq V$ we assume that $0 \leq U - V < V$. So we must compare U and V. For this we propose to evaluate the difference $D = U - V$. In fact we know that we will use this comparison when the returned approximated quotient is zero i.e., when $-V \leq D < V$. If $D < 0$ then $U < V$ else $0 \leq U - V < V$.

Now, when we evaluate the difference D in RNS, $\overline{D} = \overline{U} - \overline{V}$, the returned value can be $\overline{D} = D$ if $U \geq V$ or $\overline{D} = M + D$ if D is negative. We know that $0 < V < P < \frac{1}{4}M$, which are the preconditions of our algorithm. So, we deduce that if D is negative then $\overline{D} = M + D > \frac{1}{2}M$ and if D is positive then the returned value $\overline{D} = D < \frac{1}{4}M$.

We note $\widetilde{d}$ the `ApproxSup` of the RNS returned value of $\overline{D}$,

$$\frac{D}{M} 2^{l-1} < \widetilde{d} < \frac{D}{M} 2^{l-1} + n \tag{25}$$

If $\widetilde{d} \geq 2^{l-2}$ then $\overline{D} = M + D$, else $\overline{D} = D$ and $\widetilde{d} < 2^{l-3} + n$. As, $2^{l-3} + n < 2^{l-2}$ for $n < 2^{l-3}$, we don't have any overlaps.

But, `ApproxSup` approximate values lower than $\frac{M}{2}$. So, we will divide D by 2 to verify this condition. In this case we will compare $\widetilde{d}$ to 2^{l-3} . When D is even, the division by two can be obtained by multiplying with the inverse of 2 modulo M if M is odd. To find the parity, we need one m_i to be even, indeed the parity of D is the one of its residue modulo this m_i. Thus, the procedure `Red` computes $\frac{D}{2}$ or $\frac{D-1}{2}$ if D is odd, by multiplying with the inverse of 2 for all the odd moduli, and dividing by two for the even one.

The function `Test` is given by the algorithm below :

```
Algorithm 6: Test ($\overline{U}, \overline{V}$)
Data: $0 \leq U, V < P < \frac{1}{4}M$
Result: $B$ boolean value, true if $U \geq V$, else false
$\overline{D} \leftarrow \overline{U} - \overline{V}$;
$\overline{D} \leftarrow$ Red($\overline{D}$);
$\widetilde{d} \leftarrow$ ApproxSup($\overline{D}, \overline{1}$);
if $\widetilde{d} \geq 2^{l-3}$ then
    $B \leftarrow False$;
else
    $B \leftarrow True$;
end
return $B$
```

4. CONCLUSION

The main features of the RNS Extended Euclidean Algorithm, proposed in this paper, are that all the operators are l bits integer operators, and that, except the quotient, all the variables are updated in RNS. The main cost in time of an iteration of the algorithm 2 is one product and one addition modulo an m_i, an estimation of the quotient and a normalization, a call to Test occurs only if the approximated quotient is zero. The number of integer operators needed is equal to the number of bits of the prime number P divided by l the size of the integers which is equivalent to the size needed for classical big numbers arithmetic. In futur work we will study the exact complexity of this RNS inversion and identify the worst and average cases. We assume that `ApproxSup` can be logarithmic in time.

REFERENCES

1. J.-C. Bajard and L. Imbert. A full rns implementation of rsa. *IEEE Transactions on Computers*, 53(6):769–774, 2004.
2. J.-C. Bajard, L. Imbert, P.LY. Liardet, and Y. Teglia. Leak resistant arithmetic. *CHES 2004*, pages 59–65, Boston MA, USA, 2004. LNCS Kluwer.
3. M. Ciet, M. Neve, E. Peeters, and J.-J. Quisquater. Parallel FPGA implementation of RSA with residue number systems – can side-channel threats be avoided? In *46th IEEE International Midwest Symposium on Circuits and Systems (MWSCAS-2003)*, Cairo, Egypt, December 2003.
4. N. Koblitz. Elliptic curve cryptosystems. *Mathematics of Computation*, 48(177):203–209, January 1987.
5. Donald Knuth. *Seminumerical Algorithms*, volume 2 of *The Art of Computer Programming.* Addison-Wesley, 2 edition, 1981.
6. V. S. Miller. Uses of elliptic curves in cryptography. In H. C. Williams, editor, *Advances in Cryptology – CRYPTO '85*, volume 218, pages 417–428. Springer-Verlag, 1986.
7. K. C. Posch and R. Posch. Modulo reduction in residue number systems. *IEEE Transaction on Parallel and Distributed Systems*, 6(5):449–454, 1995
8. N. S. Szabo and R. I. Tanaka. *Residue Arithmetic and its Applications to Computer Technology.* McGraw-Hill, 1967.
9. H. L. Garner. The residue number system. *IRE Transactions on Electronic Computers*, EL-8(6):140–147, June 1959.