Generating Residue Number System Bases

Jean-Claude Bajard, Kazuhide Fukushima, Shinsaku Kiyomoto, Thomas Plantard, Arnaud Sipasseuth, Willy Susilo

Generating Residue Number System Bases

Jean-Claude Bajard*, Kazuhide Fukushima †, Shinsaku Kiyomoto †,
Thomas Plantard ‡, Arnaud Sipasseuth†‡ and Willy Susilo ‡

* Sorbonne Université, CNRS, INRIA, Institut de Mathématiques de Jussieu-Paris Rive Gauche, Ouragan, F-75005 Paris, France.
† Institute of Cybersecurity and Cryptology, University of Wollongong, Australia
‡ Information Security Laboratory of KDDI Research, Inc, Japan
Email: jean.bajard@inria.fr, ka-fukushima@kddi-research.jp, kiyomoto@kddi-research.jp, thomaspl@uow.edu.au, ar-sipasseuth@kddi-research.jp, wsusilo@uow.edu.au

***Abstract*—Residue number systems provide efficient techniques for speeding up calculations and/or protecting against side channel attacks when used in the context of cryptographic engineering. One of the interests of such systems is their scalability, as the existence of large bases for some specialized systems is often an open question. In this paper, we present highly optimized methods for generating large bases for residue number systems and, in some cases, the largest possible bases. We show their efficiency by demonstrating their improvement over the state-of-the-art bases reported in the literature. This work make it possible to address the problem of the scalability issue of finding new bases for a specific system that arises whenever a parameter changes, and possibly open new application avenues.**

***Index Terms*—Residue Number Systems**

I. Introduction

Scientific Context: The residue number system (RNS) became popular in computer science in the late 50s [1], [2]. It is a direct application of the Chinese Remainder Theorem to represent integers [3], [4]. The main interest of this approach is to deal with the residue of a value's modulo as a set of pairwise coprime numbers representing the RNS base. In theoretical works, the authors of [5], [6] use RNS to transfer approximately l-bit calculations to $\frac{l}{\log(l)}$ operations on $n \approx \log(l)$-bit operators. An intensive use of such systems was first made in signal processing [7] with small bases of approximately four elements [8], [9]. Then, in the late 90s after the emergence of public key cryptography[10], [11] involving modular arithmetic with large numbers, RNS found a new area of interest where it could show its efficiency and robustnes. For these cases, RNS bases need many elements the size of one of the machine operators [12], [13], [14].

There is a rich literature on RNS for cryptography focused on conversion algorithms and their integration into the cryptographic protocols (using different architectures such as field-programmable gate array (FPGA)) [15], [16], [17], [18] or fault detections [19], [20] where the choice of the RNS base is crucial. A frequent challenge is encountered with implementations on small operator devices, which restricts the size of the elements of the RNS bases and thus their numbers[21], [22]. Thus, a question remains: What is the maximum number of elements we can reach for a certain operator size?

In [22], the authors propose a brute-force approach of post-filtering with a search of the maximal clique in the pairwise coprime graph. Since this problem is known to be NP-complete [23], [24], they suggest a heuristic method for massive sizes to obtain a solution close to the optimal one.

Main results: We present two new filtering methods to efficiently reduce the size of the graph in order to apply a maximal clique algorithm. These approaches improve both theoretically and practically upon those of [22] and consequently improve the maximum known sizes of RNS bases for several state-of-the-art algorithms, thus expanding all of their application fields.

Organization of the paper: We first introduce some background and notations on RNS and then present two new filtering methods. Then, exhibit new bases for most relevant cases of moduli families used in state of the art RNS optimization.

II. Background

A. RNS base

Definition 1 (RNS)**.**
A set of pairwise coprime integers $\mathcal{M} = \{m_1, .., m_d\} \in (\mathbb{N}^*)^d$ is an RNS base of set size d and product size $M = \prod m_i$. For $x \in \mathbb{Z}$, we denote $\langle x \rangle_{\mathcal{M}} = \langle x_1, ..., x_d \rangle = \langle x \mod m_1, ..., x \mod m_d \rangle$ and call the set of x_i the residues of x in $\mathcal{M}$ (the RNS base of product size M).

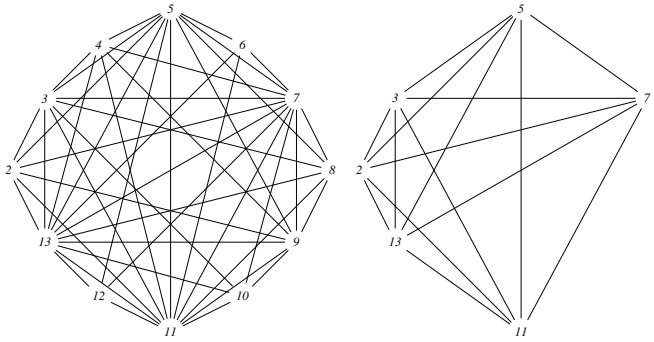
We tend to use the same notation for both $\mathcal{M}$ and M. Thus, for each $x \in \mathbb{Z}$, there is a unique representative $x_M = x \mod M$ where $0 \leq x_M < M$ and the Chinese remainder theorem (CRT) tells us that there is a trivial isomorphism from $\prod_{i=1}^{d} \mathbb{Z}_{m_i}$ to $\mathbb{Z}_M$. In particular, we denote **CRT** the function such that given any RNS base of product size M, $\mathbf{CRT}(\langle x \rangle_M) = x_M$. Note that when we search a "largest" RNS base, we tend to maximize d rather than M. The reason is simple: as hardware often have limited capabilities on the word size n and thus is often bounded as a power of 2 (i.e

2^n), the emphasis to increase M is often set on increasing d to circumvent the issue. In practice, we would like to have the same size for all residues, to even the load between all computation units. For example, given $M = 42$, we could obtain a base of set size $d = 2$ and moduli $\{6, 7\}$ or a base of set size $d = 3$ and moduli $\{2, 3, 7\}$. The first one is preferred over the second one as all moduli have the same bit size b, despite having a lower set size. The preference is then to have the following: $d \times 2^n \approx M$ with fixed n, where we can assume n is fixed in a way that optimizes hardware usage.

B. Maximum RNS base via maximum clique algorithms

In [22], methods were proposed to find a maximum size RNS base within an interval of consecutive numbers. Their approach was based on the resolution of a maximum clique problem, where every number is represented as a node and every edge between two nodes represent a coprimality relation.

Example 1. *Let $S = [2, 13]$ be a set of 12 consecutive numbers. On the left is the graph representing every number of the set connected as described above, and on the right is a maximum clique of the left graph, i.e., a subgraph of the largest possible size where every node is connected.*



Note that, while the solution is not unique (for example, node 2 can be replaced by node 4 or 8 and 3 by 9), a clique, maximal or not, will always represent a set of pairwise coprime numbers in this model.

Without reusing the exact same notations as in [22] and given a known interval $I = [I_{\min}, I_{\max}]$, we denote the following sets of pairwise coprime numbers:

- E_1 a set of prime numbers and powers of a single prime number within I.
- E_2 a maximum size set of products pq of two distinct primes such that $pq \in I$ such that no power of p, q is within E_1.
- E_k a set of integers of k prime factors (powers counted as multiple) that are not within E_1.

Do note that in the work of [22], $I_{\min}$ and $I_{\max}$ are close enough to make E_1 unique: in particular, [22] only considered the case of pseudo-mersenne numbers within a same bit size. In most RNS applications, $I_{\min}$ and $I_{\max}$ have the same bit size, thus ensuring the unicity of E_1. The work of [22] proved that there exists a maximum size (in terms of the number d of moduli within I) RNS base which include E_1 and a corresponding version of E_2 (note that the contrapositive is false). Thus, after constructing E_1 and one version of E_2, one only needs to brute-force search the remaining numbers of a set S, where S in [22] is initially $I \backslash \{E_1 \cup E_2\}$. In [22], this brute-force approach was proposed to be tackled using a maximum clique algorithm: we need to find a subgraph where every node is connected (i.e, a clique) given a graph, and no other clique should have more nodes than this subgraph.

The problem is that the maximum clique problem is known to be NP-complete, which was also pointed out in [22]. Thus, for any reasonably large set, the problem is known to be computationally unfeasible realistically. To simplify the computations, [22] also propose to prune successively after E_1, after E_2, E_3, E_4 and so on. This method does not provide the optimal graph but an approximate of the optimal solution.

Our methods, while not proven to be able to avoid the resolution of an NP-complete problem, are sure to provide an optimal solution. They do simplify the computations by essentially giving a graph of a much lower size as an input to the maximum clique algorithm. Interestingly, in many of our applications, our methods transform our original input into a graph already corresponding to a clique, thus avoiding the necessity to call a maximum clique algorithm in the first place.

III. New Filtering

In this section, we improve the theoretical work of [22] in constructing the maximum RNS base given an integer set by proposing two new filtering methods to find the largest RNS base. Similarly to the filtering method of [22] prior to the use of a graph algorithm, our methods are also proven to generate a maximum set size RNS base since they rely on the same proven assumption previously used: picking one optimal factor at a time. Basically, both algorithms use the same principle as stated in [22]: in every possible RNS base, any prime factor can be used at most once. Thus, any choice of a prime has the consequence of restricting the remaining possibilities. Both algorithms do not completely discard the utility of a clique algorithm. However, during our tests, the call for a clique algorithm was never necessary. Note that those algorithm by themselves have only one purpose: to obtain a large subset of pairwise coprime numbers from an entry set. The resulting RNS base therefore depends entirely on the entry set given as a parameter. If one wishes to create an RNS base with moduli of a specific shape, then the entry set should be composed of numbers having this specific shape (see the pseudo-Mersenne discussion in Sec IV, the Montgomery-friendly discussion in Sec VII, etc.).

A. The first method: recomposing by factors

Constructing coprime moduli sets within an interval using their prime decomposition is the initial idea of [22]. After constructing E_1 and E_2, their initial method resorts to the call of a clique algorithm over all remaining possible numbers.

Here, the first method we present is a direct upgrade over their initial method: we thin the leftover primes, thus lowering the possible combinations. As our terminology is slightly different than the one of [22], we present below our method to compute E_1 with algorithm 1, keeping information about the primes not being used for E_1 as those are necessary for the construction of E_2.

Algorithm 1 FirstStep, build E_1 and the initial set of primes

Input: $I_{\min}, I_{\max}$ the interval range
Output: E_1 a base part and P the unused remaining primes.

$E_1 \leftarrow \{\}, P \leftarrow \{\}$
for $a \in [I_{\min}, I_{\max}]$ **do**
  if a is prime **then** $E_1 \leftarrow E_1 \cup \{a\}$
for ($a \in [2, \lfloor\sqrt{I_{\max}}\rfloor]$) and ($a$ is prime) **do**
  if $\exists k > 1$ s.t $a^k \in [I_{\min}, I_{\max}]$ **then**
    $E_1 \leftarrow E_1 \cup \{a^k\}$
  else
    $P \leftarrow P \cup \{a\}$
return E_1,P

We then construct E_2, but we wish to keep information on the primes we pick for E_2 and those we will not pick. This is done by using algorithm 2. Let us call U the set of the leftover primes for I after the construction of E_2: by construction, this set is unique, as it represents the primes that cannot be included in any version of $E_1 \cup E_2$. Let s be the lowest value among U, and let k the smallest value such that $s^k > I_{\max}$. We define $R = \{p \in U | s^2 p < I_{\max}\}$.

Algorithm 2 SecondStep, build E_2 and thin down primes

Input: $I_{\min}, I_{\max}$ the interval range, and P a set of primes
Output: E_2 a base part and R a set of leftover primes.

$E_2 \leftarrow \{\}, R \leftarrow \{\}$
for $a \in P$ **do**
  $b \leftarrow$ **NextPrime**($\lfloor \frac{I_{\min}}{a} \rfloor$) ▷ Get smallest larger prime
  $x \leftarrow a \times b$
  if $x \in [I_{\min}, I_{\max}]$ **then**
    $E_2 \leftarrow E_2 \cup \{x\}$
    $P \leftarrow P \backslash a$
  else
    $s \leftarrow a^2$ ▷ Smallest unused prime squared
    Break the loop
for $a \in P$ **do**
  $P \leftarrow P \backslash a$
  $b \leftarrow$ **NextPrime**($\lfloor \frac{I_{\min}}{a} \rfloor$)
  $x \leftarrow a \times b$
  if $x \in [I_{\min}, I_{\max}]$ **then**
    $E_2 \leftarrow E_2 \cup \{x\}$
  else if $s \times a \leq I_{\max}$ **then**
    $R \leftarrow R \cup \{a\}$
return E_2,R

Now, instead of looking for combinations among U, we only need to look at combinations over R of at most k elements. The number of combinations to try is then of at most $|R|^k$ to obtain the final input to the clique algorithm.

The reason why searching over combinations of R is sufficient is since the powers p^k of a prime p are already included in E_1 and the product of two distinct primes p, q in E_2, and the only possible factorizations left are the ones that include at least 3 primes with at least 2 distinct primes, such as p^2q. Since s is the smallest prime and s^2p becomes the smallest possible product divisible by p, any possible valid integer left within I must exclusively use factors within R. This final step of the filtering process is done by algorithm 3 below.

Algorithm 3 ThirdStep, a recursive procedure

Input: $I_{\min}, I_{\max}$ the interval range, R a list of leftover primes, K the currently saved product, and RES a list of saved candidates.
Output: RES contains the remaining clique candidates.

if $K \leq I_{\max}$ **then**
  if $K \geq I_{\min}$ **then**
    $\text{RES} \leftarrow \text{RES} \cup K$
  else if $|R| \geq 1$ **then**
    $p \leftarrow R[1]$
    $R_p \leftarrow R \backslash p$ ▷ Test all lists without p
    while $K \leq I_{\max}$ **do**
      RES $\leftarrow$ **ThirdStep**($I_{\min}, I_{\max}$,R_p,K,RES)
      $K \leftarrow K \times p$ ▷ Test all powers of p
return RES

Note this does not theoretically guarantee a set of pairwise coprime integers since $a = p \times q \times z$ and $b = p \times x \times w$ could be valid integers within I. While this scenario has never occurred in our tests, we do not have any proof showing its impossibility. Thus, in theory, a maximum clique algorithm could still be needed.

The entire process is thus the concatenation of the three above algorithms, described in algorithm 4.

Algorithm 4 FactorFilter

Input: $I_{\min}, I_{\max}$ the interval range.
Output: C a set of pairwise coprime numbers, RES potential additions to grow C

$C_1, P \leftarrow$ **FirstStep**($I_{\min}, I_{\max}$)
$C_2, R \leftarrow$ **SecondStep**($I_{\min}, I_{\max}, P$)
$C \leftarrow C_1 \cup C_2$
RES$\leftarrow \{\}$
RES$\leftarrow$ **ThirdStep**($I_{\min}, I_{\max}, R, 1,$ RES)
return C, RES ▷ If RES is empty, then C is maximal

Do note that in the algorithm 4, if R is empty, then C is a maximum clique and **ThirdStep** would actually do nothing.

B. The second method: a more generic filter

The second method does not rely on any properties on the set it is used on: this makes this particular filtering process the only other choice for any set of integers that is not an interval of consecutive numbers and is also best suited for small sets. It tries to build a set by looking at the factorization of the members of the set and thinning candidates within this set. We will refer to it as *generic filtering*. To the best of our knowledge, there is no generic method for generating a RNS base of maximal size from a generic set of moduli.

The algorithm is simple: only pick up numbers that are coprime with every other number (i.e., as a substitute for E_1) and numbers that admit only one common divisor with others (i.e., as a substitute for E_2), and thinning the entry set every time a number is picked by discarding every integer that is not coprime with the chosen integer. Repeat until the entry set is empty or cannot decrease. The full process is described by algorithm 5. Note that this filter can also be applied after the previous filter since the output is also a set. After this filtering algorithm, to further increase the set size of B, we would need to call a maximum clique algorithm over M_r, the leftover of M: however, in our experiments, M_r is mostly empty, leaving B as the final base.

Algorithm 5 Generic Filtering

Input: $M = \{m_i\}$ a set of integers
Output: $B \subset M$ a RNS base, and leftovers M_r

$B \leftarrow \{\}$
$M_r \leftarrow M$
while M_r decreases in size **do**
  for $m \in M_r$ **do**
    $f \leftarrow m$ ▷ Initial common divisors
    for $m' \in M_r \backslash m$ **do**
      $d \leftarrow \gcd(m, m')$
      if $d \neq 1$ **then** ▷ Test divisor unicity
        $f \leftarrow \gcd(d, f)$
        if $f = 1$ **then** Break ▷ $\#divisors \geq 2$
    if $f \neq 1$ **then** ▷ $\#divisors < 2$
      $B \leftarrow B \cup \{m\}$
      $M_r \leftarrow \{m' \in M_r | \gcd(m, m') = 1\}$
return B, M_r

IV. Base of Pseudo-Mersenne

A. Set $[2^n - 2^{n/2}, 2^n]$

Pseudo-Mersenne numbers are of the form $2^n - c$, where c is a relatively small constant, and have been proposed for practical use in [25]. They were always considered important for efficiency since modular reductions can be done faster and avoid the use of Barrett's multiplication's algorithm [26]. We are continuing the work of [22] for pseudo-Mersenne integers within

$$I_n = [2^n - 2^{n/2}, 2^n] \text{ where } n \text{ is even}$$

We can generalize our work over the case where n is odd or when the lower bound is actually larger, as in [22]; however, for simplicity, we keep n even to have a simple computation of $2^{n/2}$ and the lower bound. We construct a largest RNS base consisting of integers of those intervals. We experimentally restrict ourselves to the case of $n \in [16, 64]$ and show that in this range, there is no need to resort to an expensive maximum clique algorithm: the *factor recomposition filtering* is enough.

The only problem is the size of U_n, the unused primes to try. Let us apply the method we described previously to thin U_n into a smaller set R_n. We list the size of R_n in the function of n in table I. As we can see, the size of R_n is low enough for us to use a brute-force approach on the remaining numbers, which is no more than $|R_n|^3$ in our tests. Note that sometimes $R_n = 0$, and we stress this is not an error: this occurs anytime $s^3 > 2^n$.

n	16	18	20	22	24	26	28	30	32	34	36	38	40 to 64
R_n	7	0	10	60	6	21	19	13	1	283	0	1	0

Table I
Size of R_n

Surprisingly, it seems as if most of the time, E_1 and E_2 are all that is needed to create the maximum RNS base of numbers within I_n, i.e., the third step of our filtering process described in algorithm 3 gives us an empty output. We list below the only few exceptions where algorithm 3 did not give us an empty output:

- I_{16}: $65453 = 29 \times 37 \times 61$
- I_{20}: $1048207 = 73 \times 83 \times 173$
- I_{22}: $4193923 = 73^2 \times 787$ and $4193993 = 109^2 \times 353$

In the above cases, calling a graph algorithm is clearly unnecessary. Observing that $R_n = 0$ for $n > 38$, it would be tempting to conjecture further results: as n increases, the smallest unused prime would increase p since there would be more possibilities for q (as in generating $pq \in E_2 \subset I_n$). However, the density of prime numbers is also vanishing. We present in table II the set size of E_1 and E_2 for our tests, and the maximum size d of the RNS base we computed.

B. Set $[2^n - 256, 2^n]$

Here, we focus on the sets $[2^n - c, 2^n]$ where c holds in a single byte, since some applications rely on c being extremely small to perform efficient operations: the smaller c is, the more efficient is the associated fast modular reduction. Table III shows the results given by the *generic algorithm*. While the set is indeed a set of consecutive numbers, the *generic algorithm* still works well as the set that has only 256 elements for each value of n. Thus, we can avoid the issue of generating consecutive primes for large values of n.

V. Base of Solinas numbers

The selection of the elements of the bases in an RNS modular multiplication method is crucial and has a great

n	$E_1 \backslash \{2^n\}$	E_2	max size
16	21	25	48
18	38	45	84
20	70	65	137
22	129	117	249
24	251	198	450
26	477	340	818
28	871	571	1443
30	1578	1027	2606
32	2931	1851	4783
34	5667	3324	8992
36	10413	5971	16385
38	19799	10884	30684
40	37798	19856	57655
42	71805	36315	108121
44	137313	66828	204142
46	263004	122472	385477
48	504634	226507	731142
50	969072	419895	1388968
52	1863100	783247	2646348
54	3586713	1460078	5046792
56	6920100	2724323	9644424
58	13351601	5126172	18477774
60	25818361	9636792	35455154
62	49975064	18153932	68128997
64	96798093	34267158	131065252

Table II
SET SIZE OF E_1, E_2, AND max RNS SET SIZE d FOR EVEN $n \in [16, 64]$

n	16	18	20	22	24	26	28	30	32	34	36	38	40
d	48	52	45	46	50	50	46	48	49	50	47	52	47

n	42	44	46	48	50	52	54	56	58	60	62	64
d	48	50	50	50	48	48	50	49	48	46	49	46

Table III
MAXIMUM SET SIZE d FOR $m_i \in [2^n - 2^8, 2^n]$ FOR EVEN $n \in [16, 64]$

impact in the overall performance. [27] proposes specific sets of optimal RNS moduli with elements of Hamming weight of three whose inverses used in the mixed radix system (MRS) reconstruction have a very small Hamming weight. This property was exploited in RNS base conversions to completely remove and replace the products by few additions/subtractions and shifts, reducing the time complexity of modular multiplication. These bases were specially crafted to perform computations with operands of sizes 256 or more and are suitable for cryptographic applications such as the elliptic curve cryptography (ECC) protocols. The numbers proposed were Solinas numbers.

Solinas numbers were first introduced in the area of fast computations in [28] to counter the patent of Crandall [25] and focused on the sparsity of the moduli for fast modular reduction without the need of a multiplication or division.

Let us consider numbers in signed base 2, i.e., $x = \sum x_i 2^i$ where $x_i \in \{-1, 0, 1\}$. While the representation of each number is not unique, we say that the number $x > 0$ is a Solinas of Hamming weight w when there exists a representation of x such that the amount of $x_i \neq 0$ is less or equal to w. For example, for $w = 1$, x is a power of 2. Let us denote d_w the maximum size of a set of pairwise coprime Solinas numbers of Hamming weight w.

As pointed in [27], one of the main advantages of RNS is that the large prime typically used in ECC can be modified even if the RNS base is fixed. However, the product size of the RNS needs to be large enough, and the set size might have to be increased further than 6 (which is the largest size given in [27]). We thus apply our methods to determine the largest set size RNS bases for each moduli size of fixed Hamming weight. This allow us to determine to what extent the approach of [27] can be exploited.

The set of Solinas numbers is by no means a set of consecutive numbers; thus, the only reasonable filtering we apply here is the *generic filtering*. Unsurprisingly, we can obtain a largest set size almost instantaneously, and we list here the results for numbers within I_n (as defined in section IV).

n	16	18	20	22	24	26	28	30	32
d_3	11	12	13	12	17	16	15	18	20
d_4	24	37	40	48	55	65	72	92	90

n	34	36	38	40	42	44	46	48
d_3	18	21	22	21	23	27	23	29
d_4	113	133	126	140	172	163	193	178

n	50	52	54	56	58	60	62	64
d_3	29	24	28	31	28	33	31	30
d_4	210	240	262	244	291	294	314	325

n	128	256	512	1024
d_3	62	81	180	281

Table IV
MAXIMUM SIZES OF $d_3, d_4 \in I_n$ FOR EVEN $n \in [16, 64]$

VI. BASE OF QUADRATIC RESIDUES

Recently, [18] proposed two new RNS Montgomery reduction algorithms: one is a single-level Montgomery, namely, sQ-RNS, and the other one a double-level Montgomery, namely, dQ-RNS, which are derived by posing quadratic residuosity requirements on RNS bases. They obtain fewer numbers of unit multiplications than all previously proposed algorithms and confirmed their improvement over the R-RNS algorithm of Gandino et al. [29] with an FPGA implementation. Since their proposed

algorithms have more regularity and symmetry than do conventional ones, they suggested it might be worth studying software implementations for multicore processors. Another topic they suggested for future study was the improvement to the two base search algorithms proposed in this paper.

We mostly focus on the latter, by improving their base search algorithms. In doing so, we have the choice of which results to improve: the moduli quality for a fixed-size moduli n and fixed set size d of an RNS base or an improvement over the set size of the RNS basis. Incidentally, those results also expand the scope of future applications for multicore processors by lowering the size of c_i in moduli of the form $2^n - c_i$ for a fixed set size d of an RNS base or improving the maximum known set size of an RNS base given a fixed bound $k > c_i$ for numbers with moduli of the form $2^n - c_i$.

A. Q-RNS Definitions

sQ-RNS uses one base of size M, $\langle m_1, ..., m_d \rangle$, while dQ-RNS needs a supplementary base of size M', $\langle m'_1, ..., m'_d \rangle$, but both use a "QR" function:

Definition 2 (QR function)**.**
Let $a, m \in \mathbb{N}^*$ *and* $\gcd(a, m) = 1$.
Then, $\mathrm{QR}(a, m) = 1 \iff \exists x$ *s.t.* $x^2 = a \mod m$.
Otherwise, $\mathrm{QR}(a, m) = 0$.
Whenever $\mathrm{QR}(a, m) = 1$*, we say that* a *is a quadratic residue mod* m.

Note that this relation is not reflexive: every odd number is a quadratic residue mod 2, but the reverse is not true. This notation is heavily reminiscent of the Jacobi symbol. However, valuing nonquadratic residues to 0 instead of -1 as in the Jacobi symbol allows them to simplify base conditions with one-liner formulas. Those conditions are the following: given a prime p that is the cardinal of the ring of integers we want to represent in RNS form:

- sQ-RNS:
  $\prod_{i \neq j} \mathrm{QR}(m_j, m_i)\mathrm{QR}(m'_j, m'_i) \times$
  $\prod_{\forall i} \mathrm{QR}(p, m_i) \prod_{\forall i,j} \mathrm{QR}(m_j, m'_i) = 1$.
- dQ-RNS: $\prod_{\forall i} \mathrm{QR}(p, m_i) = 1$
  m_i, m'_i are squared numbers over $\mathbb{Z}$.

and for each basis, moduli take the form of

- $m_i = 2^n - c_i$ with $c_i < 2^k$ for M
- $m'_i = 2^n - c'_i$ with $c'_i < 2^{k'}$ for M'

where k, k' have to be as small as possible.

We will first propose our method for generating sQ-RNS bases and then follow up with the case of dQ-RNS bases. In the following discussions, we always assume p is given in advance. We say that a base or number is valid for dQ-RNS or sQ-RNS whenever it respects the above criteria given a known prime p.

B. sQ-RNS

The problem of finding a valid sQ-RNS base is not trivial. Let us summarize the properties required.

a) Two coprime bases of size M and M' are required.
b) $\forall m$ of the base of size M, $\mathrm{QR}(m, M/m) = 1$.
c) $\forall m'$ of the base of size M', $\mathrm{QR}(m', M'/m') = 1$
d) $\mathrm{QR}(M', M) = 1$, *but* $\mathrm{QR}(M, M') = 1$ is unneeded.
e) $\mathrm{QR}(p, M) = 1$, *but* $\mathrm{QR}(p, M') = 1$ is unneeded.

In short, the choice of p limits the choice of M, but the choice of M limits the choice of M'. In practice, the prime p is given first since it is often a fixed parameter of a cryptosystem. Note that, by application of the CRT, the conditions $\mathrm{QR}(m'_j, m'_i)$ and $\mathrm{QR}(m_j, m_i)$ as listed by [18] are equivalent to conditions $b), c)$.

To solve the problem using a classical maximum clique problem, the graph generated has to be nonoriented, which is not the case here since moduli relations are not reflexive. Our approach decomposes the problem into two subproblems. We first construct M with the smallest k possible for d, P fixed with the following conditions:

1) $\forall i$, $\mathrm{QR}(P, 2^n - c_i) = 1$, $c_i < 2^k$
2) $\forall i \neq j$, $\mathrm{QR}(2^n - c_i, 2^n - c_j) = 1$
3) At least d different values

We iterate the process increasing k until we find a solution: the first step is a prefiltering process over parameters P, k, the second step is an application of a clique algorithm with *generic filtering*, and the third step is to verify we gather enough integers. After obtaining M, the computation of M' is done in the same manner, searching for the lowest k' given M, d fixed:

4) $\forall i$, $\mathrm{QR}(M, 2^n - c'_i) = 1$, $c'_i < 2^{k'}$
5) $\forall i \neq j$, $\mathrm{QR}(2^n - c'_i, 2^n - c'_j) = 1$
6) At least d different values

After those steps are done, we obtain a valid sQ-RNS base. Our given solution is not provably optimal; however, we show in table V that our results significantly improve the ones reported in [18]: we managed to find either a larger set size d for each moduli size n. Table 1 in section 4.4 of [18] presented an example for $d = 4$. We present in table VI an example obtained by our algorithms. Note that the original table of [18] did not list k but only k' and used a nonintegral value that we round to the highest integer. By diminishing the bound k, we obtain pseudo-Mersenne moduli of size n that are closer to 2^n, leading to more efficient arithmetic.

C. dQ-RNS

The difference between sQ-RNS and dQ-RNS lies in the fact that a less restrictive requirement allow [18] to use squared numbers for moduli m_i, m'_i. Thus, the previous conditions noted b), c), d) in section VI-B are always verified. This should allow for an easier basis search. [18] requires that the moduli m_i takes the form of $m_i = \sigma_i^2$ where

$$\sigma_i = (2^{n/2} - c_i)^2 = 2^n - c_i 2^{n/2+1} + c_i^2$$

and thus, a simple improvement to [18]'s base search algorithm for dQ-RNS would be to reuse our algorithms for sQ-RNS by adding the extra conditions

P	$d=4$			$d=5$			$d=6$			$d=7$		
	n	k	k'	n	k	k'	n	k	k'	n	k	k'
NIST P-192	50	4	10	40	6	11	34	7	12	-	-	-
NIST P-224	58	4	8	47	6	13	39	7	13	34	7	12
NIST P-256	65	6	7	52	6	12	44	6	13	38	8	11
NIST P-384	98	4	9	79	7	8	66	7	12	56	8	13
NIST P-521	132	6	7	106	6	8	88	7	12	76	8	15
Curve25519	65	5	8	52	7	11	44	7	11	38	8	15

P	$d=8$			$d=9$			$d=10$			$d=11$		
	n	k	k'	n	k	k'	n	k	k'	n	k	k'
NIST P-256	33	9	14	-	-	-	-	-	-	-	-	-
NIST P-384	49	9	15	44	10	18	-	-	-	-	-	-
NIST P-521	66	8	16	59	9	16	53	9	20	48	10	22

Table V
RESULTS FOR MAXIMUM SIZE FOUND FOR SQ-RNS BASES

P	$c_i = 2^n - m_i$				$c'_i = 2^n - m'_i$				k	k'
192	**5**	**7**	**11**	**15**	**255**	**663**	**689**	**863**	**4**	**10**
	27	*117*	*351*	*951*	*1153*	*2567*	*2855*	*8543*	*10*	*13*
224	**5**	**7**	**11**	**15**	**63**	**111**	**159**	**227**	**4**	**8**
	57	*63*	*147*	*447*	*27*	*731*	*3807*	*7403*	*9*	*13*
256	**55**	**31**	**43**	**63**	**23**	**79**	**91**	**103**	**6**	**7**
	535	*751*	*3219*	*8031*	*49*	*979*	*2191*	*11,335*	*13*	*14*
384	**5**	**7**	**11**	**15**	**35**	**135**	**215**	**447**	**4**	**9**
	51	*855*	*4343*	*52,155*	*117*	*831*	*1571*	*1827*	*16*	*11*
521	**47**	**51**	**55**	**63**	**15**	**39**	**65**	**87**	**6**	**7**
	347	*363*	*527*	*38,835*	*725*	*5547*	*11,535*	*38,679*	*16*	*16*
C29	**3**	**9**	**19**	**23**	**31**	**85**	**211**	**231**	**5**	**8**
	535	*2191*	*3219*	*8031*	*49*	*751*	*979*	*11,335*	*13*	*14*

Table VI
VALUES c_i, c'_i AND THEIR SIZE k FOR SQ-RNS WITH $d=4$ AND SAME n.
OUR WORK IS IN **BOLD**, [18] IN *ITALIC*

- $2^n - c_i$ is a square number in $\mathbb{Z}$ with $c_i < 2^k$
- $2^n - c'_i$ is a square number in $\mathbb{Z}$ with $c'_i < 2^{k'}$

and removing the following conditions as they become always verified

- Condition 2) in the construction of M
- Condition 4) and 5) in the construction of M'

We note that the approach used for dQ-RNS by [18] suggested to use square numbers to make their technique simpler to apply and their base search easier. From our understanding, taking square numbers is not a requirement to make use of their algorithms: it might be possible to use our methods for sQ-RNS to find valid bases for their dQ-RNS algorithms without requiring square numbers. In particular, the main difference between their base search algorithm for dQ-RNS and sQ-RNS is their number pool. As we showed we can find pools of maximal size in an efficient manner, we can assume our proposition can improve the work of [18].

It might also be possible to construct simultaneously M and M' in the case of dQ-RNS: as the relationship between M and M' becomes always true, for two RNS base size of d, we could:

- Construct one large RNS base M'' of size $> 2d$ composed of squared pseudo-Mersenne numbers of size n.
- Extract if possible one base M out of M'' of size d which verifies $\mathrm{QR}(P, M) = 1$.
- Extract a base M' of size d from M''/M regardless of its relationship with the prime P.

Both methods should be able to extend the application scope of [18] through larger RNS bases. As their work is already complex, we prefer to leave further research concerning that matter for further work.

VII. BASE OF MONTGOMERY-FRIENDLY NUMBERS

In this section, we apply *generic filtering* to the family of "Montgomery-friendly" numbers designed for the modular reduction algorithm of Montgomery. They were introduced in [30], [31], [32] for applications on elliptic or hyperelliptic curves and are considered an alternative to pseudo-Mersenne primes as long as the Montgomery reduction is used. In those numbers, the high part of their binary decomposition is similar to a pseudo-Mersenne number, while the low part is determined by ± 1.

Recently, [33] exhibited a large family of Montgomery-friendly primes that gave rise to efficient modular reduction algorithms. [33] develop two main uses. The first one is dedicated directly to cryptography, in particular to isogeny-based approaches and more generally to ECC: [33] suggest more appropriate finite fields and curves in terms of complexity for the recommended security levels for both isogeny-based cryptography and ECC. The second use is purely arithmetic, and the authors of [33] propose families of alternative RNS bases; they show that, for dedicated architectures with word operators, they can reach, for a similar or better complexity, larger RNS bases with Montgomery-friendly pairwise coprimes than the RNS bases generally used in the literature with pseudo-Mersenne numbers. This is particularly interesting for modular arithmetic used in cryptography.

We focused solely on the latter : numbers of n-bits of the form $m_i = 2^{n/2}(2^{n/2} - c) \pm 1$, where $0 < c_i < 2^{n/2}$ (including $c_i = 0$ for $m_i = 2^n - 1$), and try to obtain the largest possible set size d of an RNS base while conserving the parameters of section 5.3 of [33].

In regard to specific implementations, the moduli size is usually fixed: thus, the larger the set size d is for a fixed moduli size, the wider the application scope is. In table VII, we present results achieved by our techniques which are at least equivalent, mostly better and up to two times better in some cases than those in section 5.3 of [33].

VIII. CONCLUSION

This work expands the scope of possibilities over research on the usage of RNS bases, their efficiency and security applications in two ways: firstly, by presenting more effective methods to obtain larger bases over any set; and secondly, by improving the bases exhibited in the recent literature using the aforementioned methods. We have explained our processes and

n	32	32	32	64	64	64	64	64	64
k	8	9	10	4	6	8	10	11	12
t	20	20	20	56	56	56	56	56	56
Our d	**70**	**122**	**122**	**8**	**21**	**64**	**214**	**255**	**255**
[33]'s d	**68**	**-**	**89**	**8**	**20**	**62**	**127**	**-**	**127**

n	16	16	32	32	32	64	64	64	64
k	4	4	4	6	8	10	12	13	14
t	10	7	24	24	24	48	48	48	48
Our d	**8**	**8**	7	**22**	**70**	**214**	**705**	**1319**	**2401**
[33]'s d	**7**	**-**	7	**21**	**65**	**205**	**688**	**1295**	**2365**

Table VII
MAXIMUM SIZE OF MONTGOMERY-FRIENDLY RNS BASES.
BOLD NUMBERS SHOW WHERE WE IMPROVE UPON [33]

left open further research on improving the aforementioned methods, pointing out directions toward basic number theory: in particular, our examples show the filtering process has similarities to sieving algorithms.

ACKNOWLEDGMENTS

This work was supported by the ANR project ARRAND 15-CE39-0002-01 and the MACAO collaboration (https://ssl.informatics.uow.edu.au/MACAO/).

REFERENCES

[1] A. Svoboda and M. Valach, "Operational circuits," *Stroje na Zpracovani Informaci, Sbornik III, Nakl. CSAV, Prague*, pp. 247–295, 1955.

[2] H. L. Garner, "The residue number system," in *Papers presented at the the March 3-5, 1959, western joint computer conference*. ACM, 1959, pp. 146–153.

[3] N. Szabo and R. Tanaka, *Residue Arithmetic and its Applications to Computer Technology*. McGraw-Hill, New York, NY, USA., 1967.

[4] D. Knuth, *Seminumerical Algorithms. The Art of Computer Programming, vol. 2.* Addison- Wesley, 1981.

[5] P. W. Beame, S. A. Cook, and H. J. Hoover, "Log depth circuits for division and related problems," *SIAM Journal on Computing*, vol. 15, no. 4, pp. 994–1003, 1986.

[6] J. van der Hoeven, "Fast chinese remaindering in practice," in *Mathematical Aspects of Computer and Information Sciences - 7th International Conference (MACIS)*, ser. Lecture Notes in Computer Science, Springer, Ed., vol. 10693, 2017, pp. 95–106.

[7] G. A. Jullien and W. C. Miller, "Application of the residue number system to computer processing of digital signals," in *IEEE Symposium on Computer Arithmetic ARITH 4*, 1978, pp. 220–225.

[8] R. Chaves and L. Sousa, "$2^n + 1, s^{n+k}, s^n - 1$: A new RNS moduli set extension," in *Euromicro Symposium on Digital System Design, 2004. DSD*, 2004, pp. 210–217.

[9] A. Hiasat, L. Sousa, and A. F. Anta, "On the design of RNS inter-modulo processing units for the arithmetic-friendly moduli sets $\{2^{n+k}, 2^n - 1, 2^{n+1} - 1\}$,," *The Computer Journal*, vol. 62, no. 2, 2019.

[10] 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, 1978.

[11] W. Diffie and M. Hellman, "New directions in cryptography," *IEEE Transactions on Information Theory*, vol. 22, no. 6, pp. 644–654, 1976.

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

[13] J.-C. Bajard, L.-S. Didier, and P. Kornerup, "Modular multiplication and base extensions in residue number systems," in *15th IEEE Symposium on Computer Arithmetic ARITH 15*, I. press, Ed., 2001, pp. 59–65.

[14] S. Kawamura, M. Koike, F. Sano, and A. Shimbo, "Cox-rower architecture for fast parallel montgomery multiplication," in *Proc. Int'l Conf. Theory and Application of Cryptographic Techniques: Advances in Cryptology (EUROCRYPT 2000)*, ser. Lecture Notes in Computer Science, Springer, Ed., no. 1807, 2000.

[15] R. C. C. Cheung, S. Duquesne, J. Fan, N. Guillermin, I. Verbauwhede, and G. X. Yao, "FPGA implementation of pairings using residue number system and lazy reduction," in *13th International Workshop on Cryptographic Hardware and Embedded Systems - CHES 13th*, Springer, Ed., vol. 6917, 2011, pp. 421–441.

[16] K. Bigou and A. Tisserand, "Single base modular multiplication for efficient hardware RNS implementations of ECC," in *17th International Workshop on Cryptographic Hardware and Embedded Systems - CHES'17*, ser. Lecture Notes in Computer Science, Springer, Ed., vol. 9293, 2015, pp. 123–140.

[17] A. Lesavourey, C. Nègre, and T. Plantard, "Efficient leak resistant modular exponentiation in RNS," in *24th IEEE Symposium on Computer Arithmetic, ARITH 2017*, I. C. Society, Ed., 2017, pp. 156–163.

[18] S. Kawamura, Y. Komano, H. Shimizu, and T. Yonemura, "RNS montgomery reduction algorithms using quadratic residuosity," *Journal of Cryptographic Engineering*, vol. 9, no. 4, pp. 313–331, Nov 2019. [Online]. Available: https://doi.org/10.1007/s13389-018-0195-8

[19] J. C. Bajard, J. Eynard, and F. Gandino, "Fault detection in RNS montgomery modular multiplication," in *21st IEEE Symposium on Computer Arithmetic, ARITH*, I. C. Society, Ed., 2013, pp. 119–126.

[20] J.-C. Bajard, J. Eynard, and N. Merkiche, "Multi-fault attack detection for RNS cryptographic architecture," *IEEE 23nd Symposium on Computer Arithmetic*, July 2016.

[21] J.-C. Bajard and N. Merkiche, "Double level montgomery cox-rower architecture, new bounds," in *Smart Card Research and Advanced Applications - 13th International Conference, CARDIS*, Springer, Ed., vol. 8968, 2014, pp. 139–153.

[22] B. Gérard, J. Kammerer, and N. Merkiche, "Contributions to the design of residue number system architectures," in *2015 IEEE 22nd Symposium on Computer Arithmetic*, June 2015, pp. 105–112.

[23] R. M. Karp, "Reducibility among combinatorial problems," in *Complexity of computer computations*. Springer, 1972, pp. 85–103.

[24] E. Tomita, "Efficient algorithms for finding maximum and maximal cliques and their applications," in *11th International Conference and Workshops on Algorithms and Computation, WALCOM*, ser. Lecture Notes in Computer Science, Springer, Ed., vol. 10167, 2017, pp. 3–15.

[25] R. E. Crandall, "Method and apparatus for public key exchange in a cryptographic system," U.S. Patent #5159632, 1992.

[26] P. Barrett, "Implementing the rivest shamir and adleman public key encryption algorithm on a standard digital signal processor," in *Conference on the Theory and Application of Cryptographic Techniques*. Springer, 1986, pp. 311–323.

[27] J. C. Bajard, M. Kaihara, and T. Plantard, "Selected RNS bases for modular multiplication," in *2009 19th IEEE Symposium on Computer Arithmetic*, 2009, pp. 25–32.

[28] J. A. Solinas, "Generalized mersenne numbers," Technical Report CORR-99-39, University of Waterloo, Tech. Rep., 1999.

[29] 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, 2012.

[30] M. Hamburg, "Fast and compact elliptic-curve cryptography," Cryptology ePrint Archive, Report 2012/309, 2012, https://eprint.iacr.org/2012/309.

[31] J. W. Bos, C. Costello, H. Hisil, and K. Lauter, "Fast cryptography in genus 2," in *EUROCRYPT 2013: 32nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Athens, Greece, May 26-30, 2013*, ser. Lecture Notes in Computer Science, vol. 7881. Springer, 2013, pp. 194–210.

[32] J. W. Bos, C. Costello, P. Longa, and M. Naehrig, "Selecting elliptic curves for cryptography: an efficiency and security analysis," *Journal of Cryptographic Engineering*, vol. 6, no. 4, pp. 259–286, Nov 2016. [Online]. Available: https://doi.org/10.1007/s13389-015-0097-y

[33] J.-C. Bajard and S. Duquesne, "Montgomery-friendly primes and applications to cryptography," Cryptology ePrint Archive, Report 2020/665, 2020, https://eprint.iacr.org/2020/665.