Generating Residue Number System Bases
Jean-Claude Bajard, Kazuhide Fukushima, Shinsaku Kiyomoto, Thomas Plantard, Arnaud Sipasseuth, Willy Susilo

To cite this version:

HAL Id: hal-03457951
https://hal.sorbonne-universite.fr/hal-03457951
Submitted on 1 Dec 2021

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.
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 \( \frac{1}{\log(l)} \) 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 robustness. 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 \( \text{CRT} \) the function such that given any RNS base of product size \( M \), \( \text{CRT}(\langle x \rangle_{\mathcal{M}}) = x_M \). 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 \( I = [I_{\text{min}}, I_{\text{max}}] \), 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], \( I_{\text{min}} \) and \( I_{\text{max}} \) are close enough to make \( E_1 \) unique: in particular, [22] only considered the case of pseudo-merсенne numbers within a same bit size. In most RNS applications, \( I_{\text{min}} \) and \( I_{\text{max}} \) 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 \( I \setminus \{k_1 \cup E_2\} \). 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 algorithms 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 $E_2$ with algorithm 2, 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:** $I_{\text{min}}, I_{\text{max}}$ the interval range

**Output:** $E_1$ a base part and $P$ the unused remaining primes.

1. $E_1 \leftarrow \{\}$, $P \leftarrow \{\}$
2. for $a \in [I_{\text{min}}, I_{\text{max}}]$ do
   3. if $a$ is prime then $E_1 \leftarrow E_1 \cup \{a\}$
4. for $(a \in [2, \lfloor \sqrt{I_{\text{max}}} \rfloor])$ and $(a$ is prime) do
   5. if $\exists k > 1$ s.t $a^k \in [I_{\text{min}}, I_{\text{max}}]$ then $E_1 \leftarrow E_1 \cup \{a^k\}$
   7. else
   8. $P \leftarrow P \cup \{a\}$
9: 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 3. 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 $k$ be the smallest value such that $s^k > I_{\text{max}}$. We define $R = \{p \in U | s^k p < I_{\text{max}}\}$.

**Algorithm 2 SecondStep.** build $E_2$ and thin down primes

**Input:** $I_{\text{min}}, I_{\text{max}}$ the interval range, and $P$ a set of primes

**Output:** $E_2$ a base part and $R$ a list of leftover primes.

1. $E_2 \leftarrow \{\}$, $R \leftarrow \{\}$
2. for $a \in P$ do
   3. $b \leftarrow \text{NextPrime}(\lfloor \frac{I_{\text{max}}}{a} \rfloor)$ // Get smallest larger prime
   4. $x \leftarrow a \times b$
   5. if $x \in [I_{\text{min}}, I_{\text{max}}]$ then $E_2 \leftarrow E_2 \cup \{x\}$
   7. $P \leftarrow P \setminus \{a\}$
   8. else
   9: $s \leftarrow a^2$ // Smallest unused prime squared
 10: Break the loop
11: for $a \in P$ do
12: $P \leftarrow P \setminus \{a\}$
13: $b \leftarrow \text{NextPrime}(\lfloor \frac{I_{\text{min}}}{a} \rfloor)$
14: $x \leftarrow a \times b$
15: if $x \in [I_{\text{min}}, I_{\text{max}}]$ then $E_2 \leftarrow E_2 \cup \{x\}$
16: $R \leftarrow R \setminus \{a\}$
19: 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 $p^2 q$. Since $s$ is the smallest prime and $s^2 p$ 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:** $I_{\text{min}}, I_{\text{max}}$ 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.

1: if $K \leq I_{\text{max}}$ then
2: if $K \geq I_{\text{min}}$ then
3: RES $\leftarrow$ RES $\cup K$
4: else if $|R| \geq 1$ then
5: $p \leftarrow R[1]$
6: $R_p \leftarrow R[p]$ $\triangleright$ Test all lists without $p$
7: while $K \leq I_{\text{max}}$ do
8: RES $\leftarrow$ ThirdStep($I_{\text{min}}, I_{\text{max}}, R_p, K, \text{RES}$)
9: $K \leftarrow K \times p$ $\triangleright$ Test all powers of $p$
10: 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:** $I_{\text{min}}, I_{\text{max}}$ the interval range.

**Output:** $C$ a set of pairwise coprime numbers, RES potential additions to grow $C$

1: $C_1, P \leftarrow \text{FirstStep}(I_{\text{min}}, I_{\text{max}})$
2: $C_2, R \leftarrow \text{SecondStep}(I_{\text{min}}, I_{\text{max}}, P)$
3: $C \leftarrow C_1 \cup C_2$
4: RES$\leftarrow \{\}$
5: RES$\leftarrow$ ThirdStep($I_{\text{min}}, I_{\text{max}}, R, 1, \text{RES}$)
6: return $C$, RES $\triangleright$ 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: $B \subseteq M$ a RNS base, and leftovers $M_r$
1: $B \leftarrow \{\}$
2: $M_r \leftarrow M$
3: while $M_r$ decreases in size do
4:   for $m \in M_r$ do
5:     $f \leftarrow m$ \> Initial common divisors
6:     for $m' \in M_r \backslash \{m\}$ do
7:       $d \leftarrow \gcd(m,m')$
8:       if $d \neq 1$ then \> Test divisor unicity
9:         $f \leftarrow \gcd(d,f)$
10:        if $f = 1$ then Break \> #divisors $\geq 2$
11:       if $f \neq 1$ then \> #divisors $< 2$
12:         $B \leftarrow B \cup \{m\}$
13:         $M_r \leftarrow \{m' \in M_r | \gcd(m,m') = 1\}$
14: 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 [25]. We are continuing the work of [22] for pseudo-Mersenne integers within

$I_n = [2^n - 2^n/2, 2^n]$ where $n$ 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 $s^3 > 2^n$.

<table>
<thead>
<tr>
<th>$n$</th>
<th>$R_n$</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>7</td>
</tr>
<tr>
<td>18</td>
<td>0</td>
</tr>
<tr>
<td>20</td>
<td>10</td>
</tr>
<tr>
<td>22</td>
<td>6</td>
</tr>
<tr>
<td>24</td>
<td>6</td>
</tr>
<tr>
<td>26</td>
<td>26</td>
</tr>
<tr>
<td>28</td>
<td>25</td>
</tr>
<tr>
<td>30</td>
<td>13</td>
</tr>
<tr>
<td>32</td>
<td>31</td>
</tr>
<tr>
<td>34</td>
<td>283</td>
</tr>
<tr>
<td>36</td>
<td>0</td>
</tr>
<tr>
<td>38</td>
<td>1</td>
</tr>
<tr>
<td>40</td>
<td>0</td>
</tr>
</tbody>
</table>

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 5 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 × 37 × 61
- $I_{20}$: 1048207 = 73 × 83 × 173
- $I_{22}$: 4193923 = 73² × 787 and 4193993 = 109² × 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 $pq \in E_2 \subseteq I_n$). However, the density of prime numbers is also vanishing. We present in table II the 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


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

**Table II**

**Set size of $E_1$, $E_2$, and max RNS set size $d$ for even $n \in [16, 64]$**

<table>
<thead>
<tr>
<th>$n$</th>
<th>$d$</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>18</td>
</tr>
<tr>
<td>18</td>
<td>20</td>
</tr>
<tr>
<td>20</td>
<td>22</td>
</tr>
<tr>
<td>22</td>
<td>24</td>
</tr>
<tr>
<td>24</td>
<td>26</td>
</tr>
<tr>
<td>26</td>
<td>28</td>
</tr>
<tr>
<td>28</td>
<td>30</td>
</tr>
<tr>
<td>30</td>
<td>32</td>
</tr>
<tr>
<td>32</td>
<td>34</td>
</tr>
<tr>
<td>34</td>
<td>36</td>
</tr>
<tr>
<td>36</td>
<td>38</td>
</tr>
<tr>
<td>38</td>
<td>40</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$d$</th>
<th>42</th>
</tr>
</thead>
<tbody>
<tr>
<td>44</td>
<td>46</td>
</tr>
<tr>
<td>48</td>
<td>50</td>
</tr>
<tr>
<td>50</td>
<td>52</td>
</tr>
<tr>
<td>52</td>
<td>54</td>
</tr>
<tr>
<td>54</td>
<td>56</td>
</tr>
<tr>
<td>56</td>
<td>58</td>
</tr>
<tr>
<td>60</td>
<td>62</td>
</tr>
<tr>
<td>62</td>
<td>64</td>
</tr>
</tbody>
</table>

**Table III**

**Maximum set size $d$ for $m_i \in [2^n - 2^4, 2^n]$ for even $n \in [16, 64]$**

<table>
<thead>
<tr>
<th>$n$</th>
<th>16</th>
<th>18</th>
<th>20</th>
<th>22</th>
<th>24</th>
<th>26</th>
<th>28</th>
<th>30</th>
<th>32</th>
</tr>
</thead>
<tbody>
<tr>
<td>$d_3$</td>
<td>11</td>
<td>12</td>
<td>13</td>
<td>12</td>
<td>24</td>
<td>26</td>
<td>28</td>
<td>30</td>
<td>32</td>
</tr>
</tbody>
</table>

| $d_4$ | 24 | 37 | 40 | 48 | 55 | 65 | 72 | 92 | 90 |

<table>
<thead>
<tr>
<th>$n$</th>
<th>34</th>
<th>36</th>
<th>38</th>
<th>40</th>
<th>42</th>
<th>44</th>
<th>46</th>
<th>48</th>
</tr>
</thead>
<tbody>
<tr>
<td>$d_3$</td>
<td>18</td>
<td>21</td>
<td>22</td>
<td>21</td>
<td>23</td>
<td>27</td>
<td>23</td>
<td>29</td>
</tr>
<tr>
<td>$d_4$</td>
<td>113</td>
<td>133</td>
<td>126</td>
<td>140</td>
<td>172</td>
<td>163</td>
<td>193</td>
<td>178</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$n$</th>
<th>50</th>
<th>52</th>
<th>54</th>
<th>56</th>
<th>58</th>
<th>60</th>
<th>62</th>
<th>64</th>
</tr>
</thead>
<tbody>
<tr>
<td>$d_3$</td>
<td>29</td>
<td>24</td>
<td>28</td>
<td>31</td>
<td>28</td>
<td>33</td>
<td>31</td>
<td>30</td>
</tr>
<tr>
<td>$d_4$</td>
<td>210</td>
<td>240</td>
<td>262</td>
<td>244</td>
<td>291</td>
<td>294</td>
<td>314</td>
<td>325</td>
</tr>
</tbody>
</table>

**Table IV**

**Maximum sizes of $d_3, d_4 \in I_n$ for even $n \in [16, 64]$**

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]).

VI. BASE OF QUADRATIC RESIDUES

Recently, [13] 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

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

Definition 2 (QR function).
Let $a, m \in \mathbb{N}^*$ and $\gcd(a, m) = 1$.
Then, \( \text{QR}(a, m) = 1 \iff \exists x \text{ s.t. } x^2 = a \mod m. \)
Otherwise, \( \text{QR}(a, m) = 0. \)
Whenever \( \text{QR}(a, m) = 1 \), 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 allow for an easier basis search, [18] requires that the moduli have to be as small as possible.

b) \( \forall m \) of the base of size $M, \text{QR}(m, M/m) = 1.$
c) \( \forall m' \) of the base of size $M', \text{QR}(m', M'/m') = 1.$
d) \( \text{QR}(M', M) = 1, \text{ but } \text{QR}(M, M') = 1 \) is unneeded.
e) \( \text{QR}(p, M) = 1, \text{ but } \text{QR}(p, M') = 1 \) 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 \( \text{QR}(m'_i, m'_i) \) and \( \text{QR}(m_j, m_k) \) 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) \( \forall i, \text{QR}(P, 2^n - c_i) = 1, \ c_i < 2^k \)
2) \( \forall i \neq j, \text{QR}(2^n - c_i, 2^n - c_j) = 1 \)
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) \( \forall i, \text{QR}(M, 2^n - c'_i) = 1, \ c'_i < 2^{k'} \)
5) \( \forall i \neq j, \text{QR}(2^n - c'_i, 2^n - c'_j) = 1 \)
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 $m_i$ take the form of $m_i = \sigma_i^2$ where

\[ \sigma_i = \left(2^{n/2} - c_i\right)^2 = 2^n - c_i 2^{n/2+1} + c_i^2 \]

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.
We can find pools of maximal size in an efficient manner, we dQ-RNS and sQ-RNS is their number pool. As we showed the main difference between their base search algorithm for algorithms without requiring square numbers. In particular, understanding, taking square numbers is not a requirement to suggested to use square numbers to make their technique we could:

- Construct one large RNS base \( M' \) of size \( d > 2d \) composed of squared pseudo-Mersenne numbers of size \( n \).
- Extract if possible one base \( M' \) out of \( M'' \) of size \( d \) which verifies \( \text{QR}(P,M) = 1 \).

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 \( \pm 1 \).

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 \( m_i = 2^{n/2}(2^{n/2} - c_i) \pm 1 \), 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
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 AARRAND 15-CE39-0002-01 and the MACAO collaboration (https://ssl.informatics.uow.edu.au/MACAO/).

REFERENCES


Table VII

<table>
<thead>
<tr>
<th>n</th>
<th>32</th>
<th>32</th>
<th>32</th>
<th>64</th>
<th>64</th>
<th>64</th>
<th>64</th>
<th>64</th>
<th>64</th>
</tr>
</thead>
<tbody>
<tr>
<td>k</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>4</td>
<td>6</td>
<td>8</td>
<td>10</td>
<td>11</td>
<td>12</td>
</tr>
<tr>
<td>t</td>
<td>20</td>
<td>20</td>
<td>20</td>
<td>56</td>
<td>56</td>
<td>56</td>
<td>56</td>
<td>56</td>
<td>56</td>
</tr>
<tr>
<td>Our d</td>
<td>70</td>
<td>122</td>
<td>122</td>
<td>8</td>
<td>21</td>
<td>64</td>
<td>214</td>
<td>255</td>
<td>255</td>
</tr>
<tr>
<td>33’s d</td>
<td>68</td>
<td>-</td>
<td>89</td>
<td>8</td>
<td>20</td>
<td>62</td>
<td>127</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>n</th>
<th>16</th>
<th>32</th>
<th>32</th>
<th>32</th>
<th>64</th>
<th>64</th>
<th>64</th>
<th>64</th>
<th>64</th>
</tr>
</thead>
<tbody>
<tr>
<td>k</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>6</td>
<td>8</td>
<td>10</td>
<td>12</td>
<td>13</td>
<td>14</td>
</tr>
<tr>
<td>t</td>
<td>10</td>
<td>7</td>
<td>24</td>
<td>24</td>
<td>24</td>
<td>48</td>
<td>48</td>
<td>48</td>
<td>48</td>
</tr>
<tr>
<td>Our d</td>
<td>8</td>
<td>8</td>
<td>7</td>
<td>22</td>
<td>60</td>
<td>214</td>
<td>705</td>
<td>1319</td>
<td>2401</td>
</tr>
<tr>
<td>33’s d</td>
<td>7</td>
<td>-</td>
<td>-</td>
<td>7</td>
<td>21</td>
<td>65</td>
<td>205</td>
<td>688</td>
<td>1295</td>
</tr>
</tbody>
</table>

BOLD NUMBERS SHOW WHERE WE IMPROVE UPON