Generating Very Large RNS Bases

Jean Claude Bajard, Kazuhide Fukushima, Thomas Plantard, Arnaud Sipasseuth

Generating Very Large RNS Bases

Jean Claude Bajard*, Kazuhide Fukushima †, Thomas Plantard and Arnaud Sipasseuth†
* Sorbonne Université, CNRS, INRIA, IMJ-PRG, Ouragan, Paris, France
† Information Security Laboratory of KDDI Research, Inc, Japan
Email: jean.bajard@inria.fr, ka-fukushima@kddi-research.jp, thomas.plantard@gmail.com,
ar-sipasseuth@kddi-research.jp

Abstract—Residue Number Systems (RNS) are proven to be effective in speeding up computations involving additions and products. For these representations, there exists efficient modular reduction algorithms that can be used in the context of arithmetic over finite fields or modulo large numbers, especially when used in the context of cryptographic engineering. Their independence allows random draws of bases, which also makes it possible to protect against side-channel attacks, or even to detect them using redundancy. These systems are easily scalable, however the existence of large bases for some specific uses remains a difficult question. In this paper, we present four techniques to extract RNS bases from specific sets of integers, giving better performance and flexibility to previous works in the litterature. While our techniques do not allow to solve efficiently every possible case, we provide techniques to provably and efficiently find the largest possible available RNS bases in several cases, improving the state-of-the-art on various works of the recent literature.

Index Terms—Residue Number Systems, Setwise Coprime, Modular Arithmetic, Cryptography.

1 Introduction

Scientifc Context: Unconventional arithmetic offers original and effective approaches in various domains of application [1]. The use of Residue Number System (RNS) appeared in the 1950s as arithmetic [2], [3]. This representation is directly inspired by the Chinese remainder theorem [4], [5]. The main interest lies in the speed of the addition and multiplication operations which are distributed on small values that are the modular remainders on a set of pairwise coprime numbers named RNS base. Despite a conversion cost which in the worst case is quadratic in the size of the base, calculations involving additions and products become extremely profitable (i.e. the inner product, convolution products, operations on large numbers, etc.). However, division and comparison remain costly operations in RNS, recent works propose interesting approaches [6], [7], [8]. The three main areas of application are signal processing [9], [10], [11], cryptography but also in theoretical computer science to reach complexity bounds [12], [13]. The work in this paper is relevant for all applications on large numbers including cryptography since the 90's [14] with RSA, DH, ECC [15], [16], pairing [17], Euclidean lattices, homomorphic protocols [18], [19], etc.

In a domain such as cryptography, arithmetic operations are performed modulo large numbers that are often prime, the use of RNS becomes more complicate as modular reduction requires a conversion of RNS bases [20], [21], [22], [23]. This last point has generated a rich literature, in particular around the choice of bases for efficient implementations [15], [24], [25], [26]. RNS are also particularly interesting for countering attacks by faults, as the addition of redundancy elements at the base level makes it possible to set up fault detection [27]. Finally, the random drawing of bases ensures that the same calculation produces different patterns at each evaluation, making learning possible leakage of information more difficult [28].

Implementations are constrained by the limitations of the architectures we deals with. The question we are faced with is what sizes can be reached. This is very dependent on the operators of the target architecture which limit the size of the elements of the RNS base. Finding the largest possible base with this constraint on its elements thus becomes a major issue [29], [30]. In order to answer this challenge, an approach proposes after a filtering to build a co-primality graph and to perform a brute force search of maximal cliques [29]. This problem is known to be NP-complete [31], [32].

In order to achieve larger RNS base sizes, considering the cost of finding a maximal clique, filtering to reduce the graph size becomes crucial.

Main results: In this paper, we introduce four filtering methods that reduce the size of the graph. Our approach significantly improves the one of [29]. The RNS bases we are able to build are larger than those proposed in the literature by maintaining the same constraints. One can find in [26] a large overview of RNS bases with their advantages. The first algorithm is targeted towards intervals. The second algorithm is targeted towards random sets. The third algorithm is an adaptation of the first algorithm for "non-interval semi-random" sets. The fourth one is a heuristic approach, aimed to accelerate computations.

In the literature: Different approaches found in the literature deal with obtaining an easy reduction for each modulo and an efficient RNS base extension. Most of them use classical pseudo-Mersenne or Solinas moduli. We mention here some particular approaches that condition the moduli. In the Cox-Rover algorithm [21] authors suggest to use pseudo-Mersenne coprimes with an interdependence between the number of moduli, their pseudo-Mersenne form and the truncation error of the computation. This could reduce significantly the possibilities depending on the target architecture. With the double Montgomery approach [29], the authors use a Montgomery reduction at the modulo level, they relax the constraints which allows to significantly increase the number of moduli. Then quadratic RNS [26] offers an efficient bases exchange but adding a quadratic property of the moduli. Finally, a recent approach suggest friendly Montgomery numbers [33] to increase the number of coprimes while keeping an efficient base extension. We will illustrate how our methods allow to build large RNS bases for each case.

Organization of the paper: After introducing some useful background for the understanding of this article, we introduce in Section 3 four filtering approaches. We present two basic alternatives to [29] for intervals and random sets, then specialize those basic alternatives to propose another two alternatives to deal with specialized sets for a total of four filtering techniques. Then, we propose new bases for the most relevant cases of families of moduli used in the state of the art of RNS optimization [26].

2 Background

2.1 Residue Number Systems and notations

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

The Chinese remainder theorem (CRT) ensures that there is an isomorphism between $\prod_{i=1}^{d} \mathbb{Z}_{m_i}$ to $\mathbb{Z}_M$. Thus, for each $x \in \mathbb{Z}$, $\langle x \rangle_{\mathcal{M}}$ is unique and corresponds to $x_M = x \mod M$. Sometimes the same notation is used for both $\mathcal{M}$ and M. Note that the search for a large RNS base, i.e. reaching large sizes for M, often means increasing d, as the elements of an RNS base are limited in size depending on the architectures used. If the machine words are of a given size n in binary, then each element of the database is smaller than 2^n. In practice, we would like to have the same size for all residues, in order to balance the load between all computing units.

We start by introducing some specific notations.

Notation 1. *We note $\mathbb{P}$ the set of prime numbers. For any set I, we note $I_{\min}$ its smallest element, $I_{\max}$ its largest element. We note $I_{\mathbb{P}}$ the set of primes that can divide at least one non-zero element of I (e.g. to simplify in some algorithms, $I_{\mathbb{P}}$ can be replaced by the set of primes smaller than $I_{\max}$).*

Notation 2 (Sets of maximum possible pairwise coprimes)**.** *We denote $\Phi(S, I)$ the set of all sets of pairwise coprime numbers of maximal size containing elements of $S \cup I$.*
I is the initial set of numbers considered, and S is a set of filtered values. Almost all the values of S are pairwise coprimes.

Example 1. *Let $S = \{\}$ and $I = \{2, 3, 4, 11, 17, 121\}$:*

- $\mathcal{M}_1 = \{121, 17, 2, 3\} \in \Phi(S, I)$
- $\mathcal{M}_2 = \{11, 17, 4, 3, 2\} \notin \Phi(S, I)$ *as* $\gcd(2, 4) \neq 1$
- $\mathcal{M}_3 = \{11, 17, 3\} \notin \Phi(S, I)$*:* $|\mathcal{M}_3| < |\mathcal{M}_1|$
- $\mathcal{M}_4 = \{11, 17, 2, 3\} \in \Phi(S, I)$
- $\mathcal{M}_5 = \{11, 17, 4, 3\} \in \Phi(S, I)$
- $\mathcal{M}_6 = \{121, 17, 4, 3\} \in \Phi(S, I)$

By bruteforce, we can easily see that $\{\mathcal{M}_1, \mathcal{M}_4, \mathcal{M}_5, \mathcal{M}_6\} = \Phi(S, I)$.

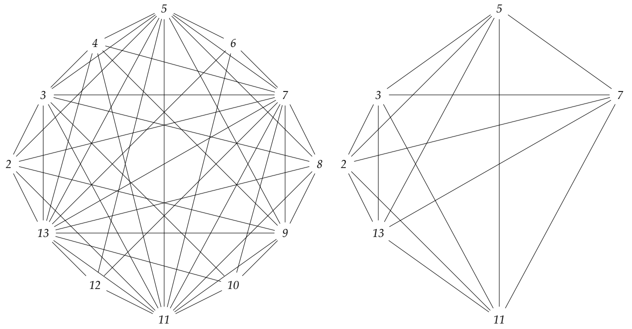
The goal of this paper is to find *at least one* element of $\Phi(\{\}, I)$ for specific I. Thus, we try to reduce the problem by finding smaller sets $S' \subseteq I$ and $I' \subseteq I$ such that $\Phi(S', I') \subseteq \Phi(\{\}, I)$. In the previous example, If $S' = \{121\}$ and $I' = \{2, 3, 4, 17\}$ then $\Phi(S', I') = \{\mathcal{M}_1, \mathcal{M}_6\} \subseteq \Phi(\{\}, I)$. At the end of the process of filtering we could obtain $S = \{2, 3, 4, 17, 121\}$ and $I = \{\}$. The work of [29] proposed a solution when I is an interval.

2.2 Maximum RNS bases are maximum cliques

In [29], methods were proposed to find a maximum size RNS base within an interval of consecutive numbers. The authors of [29] saw the problem of finding an element of $\Phi(\{\}, I)$ as a graph problem. The approach is the following:

1) $\forall a \in I$, create a node tagged with a.
2) $\forall a_1, a_2 \in I$ s.t $\gcd(a_1, a_2) = 1$, add an edge (a_1, a_2).
3) A maximum size RNS base from I is a clique from the resulting graph.

Example 2. *Let $I = [2, 13]$ be a set of 12 consecutive numbers. On the left is the graph constructed as per the above described process, 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: it is also a graph that can be constructed from an element of $\Phi(\{\}, I)$.*



Note that the solution is not unique: for example, node 2 can be replaced by node 4 or 8 and 3 by 9, thus $\Phi(\{\}, I)$ has at least six elements. $\Phi(\{2, 3\}, I)$ however, is a singleton. A clique, maximal or not, will always represent a set of pairwise coprime numbers in this model.

As also pointed by [29], one issue with this initial approach is that the maximum clique problem is known to be NP-complete, thus before tackling the graph problem, [29] proposes to simplify the search space by "filtering" $\Phi(\emptyset, I)$ into a smaller set. It is very clear from the above examples, that given a set I, we have $\Phi(I \cap \mathbb{P}, I) \subseteq \Phi(\emptyset, I)$, thus part of the problem in finding an element of $\Phi(\emptyset, I)$ can be reduced to find an element of $\Phi(I \cap \mathbb{P}, I)$, the difficulty comes from filtering $\Phi(I \cap \mathbb{P}, I)$ even further. Thus, they propose some solutions to ease the computations. Their main observation is the following lemma:

Lemma 1 ([29]'s maximality preprocessing)**.**
Let $I = [I_{\min}, I_{\max}]$ and $\delta = I_{\max} - I_{\min} > 0$. Given

$$S = \{a^k b \in I \text{ s.t } a \leq \delta,\ b > \delta \text{ or } b = 1,\ \{a, b\} \subset \mathbb{P} \cup \{1\}\}$$

we have $\Phi(S, I) \subseteq \Phi(\{\}, I)$ and $\Phi(S, I) \neq \emptyset$.

In other words, [29] can construct a subset where there is at least *one* maximum RNS base containing this set S, reducing the search space. After constructing S, the search space can be even further simplified: one only needs to replace I with the set $I_S = \{x \in I | \forall m \in S, \gcd(m, x) = 1\}$ i.e the elements of I coprime to S. we thus have $\Phi(S, I_S) = \Phi(S, I) \subseteq \Phi(\{\}, I)$, and finding an element of $\Phi(S, I_S)$ is computationally much easier.

As pointed by [29] the set S is easily computable. In fact, for some small parameters, [29] reduce I_S to an empty set, outputting $S \in \Phi(\{\}, I)$ as a solution. The approach however can have some limitations after computing S:

- For large I, I_S can be too large to be reasonably processed and a clique algorithm might still be required.
- For large I, I_S can be itself expensive to compute.
- If I is not an interval, the approach cannot be used.

To alleviate the first issue, [29] propose to heuristically prune the graph: constructing S was essentially pruning the graph from elements containing only one or two distinct primes, thus their suggestion was to continue further by pruning combinations of 3 distinct prime factors, then 4, etc... However, as they also pointed out, this heuristic pruning method beyond S does no longer guarantee a maximal size RNS base.

In our work, we propose upgraded pruning methods to help alleviate all three issues. Those methods, while not proven to be able to avoid the resolution of an NP-complete problem, are sure to provide an optimal solution. We also provide a highly efficient greedy version of our methods which do not guarantee optionality, even if often achieved. 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.

3 Two novel basic filtering techniques

Here we propose two alternatives filtering techniques, initially proposed at [34], that directly improve over the theoretical work of [29]. Similarly to [29], the aim is to minimize the size of the set we give as an input to a graph algorithm, while relying on a same principle of picking a provably optimal choice at each iteration to guarantee the maximality of the output.

The first method modifies [29] in a way such that computing $\gcd$ is *never* necessary, and that the initial set I is never stored, only prime numbers are (i.e. $I_{\mathbb{P}}$). This number of primes can be significantly smaller than the size of the initial set I. The second method is not a direct improvement of [29] but rather an alternative method to deal with sets that were not intervals: Lemma 1 can only applies if the initial set given as an entry is an interval. It is also more efficient than the other methods for sets of small size, even intervals, as it does not require any knowledge of the prime numbers.

Important note: These filtering techniques give at least one RNS base with the maximum number of moduli extracted from an initial set, they do not select the best base according to particular criteria (for example the size of the product of the moduli), among the, *potentialy numerous*, bases with the maximum number of moduli.

3.1 First method: filtering by factors recomposition

Constructing coprime moduli sets within an interval by analyzing their factorization was the core idea of [29]. Their method call a clique algorithm over all remaining possible numbers, constructing S and I_S beforehand. Our first improvement over [29] is the following: when dealing with an interval, *there is no need to store the interval at all.* We just need the list of any prime number that can be found in a decomposition of an integer within the interval. Prime lists and code to compute them can be found in various places on the internet ([35], [36], [37]) if primes required are not overly large (i.e beyond 2^{64}), and checking whether an integer is within an interval can be done with the help of only two values: $I_{\min}$ and $I_{\max}$.

Here, the first method we present is essentially the following: we thin the leftover primes, thus lowering the possible combinations, accelerating the process. This algorithm can ensure a maximum RNS base after a call to a clique algorithm to leftover conflicting combinations, similarly to [29]. In [29], their core observation is that any integer prime $p > \delta$ (with $\delta = I_{\max} - I_{\min}$) can only appear once in the decomposition of integers of I. We add a generalized observation, which severely limits the amount of prime numbers we have to consider.

Knowing that any prime $p > \sqrt{I_{\max}}$ cannot be powered in a said decomposition, we can further limit the amount of primes to consider. Let us denote $\beta = \max(\delta, \sqrt{I_{\max}})$. We explain below our method to compute E_1 the set of prime powers within I with the algorithm **FirstStep**, keeping information about the primes not being used for E_1 as those are necessary later.

Algorithm 1 FirstStep

```
Input: $P$ all the primes smaller than $\beta$
    $E_1 \leftarrow \mathbb{P} \cap I$
    for $p \in P$ do
        if there is a $k \geq 1$, such that $p^k \in I$ then
            Put the largest $p^k$ into $E_1$
            Eliminate $p$ from $P$
Output: $E_1, P$.
```

Example 3. *If $I = [2^{16} - 2^8, 2^{16}]$ then*

1 $P = \{2, 3, 5, ..., 251\}$ *and* $E_1 = \{65287, ..., 65521\}$
2 $2^{16} \in I$*, so eliminate 2 from P and add* 2^{16} *to* E_1
3 $P = \{3, 5, ..., 251\}$ *and* $E_1 = \{65287, ..., 65521, 2^{16}\}$
4 *No other* $p \in P$ *can be transferred and eliminated*
End $|P| = 53$ *and* $|E_1| = 22$

Here we reduced the problem to $\Phi(E_1, I') \subseteq \Phi(\{\}, I)$, where I' is I minus all the primes or power of primes. Note that we do not store I nor I', this is one of the main advantage compared to the approach of [29].

After **FirstStep**, if E_1 is not a solution by itself (most often not), then we can continue our process using the leftover primes within P to find candidates to expand E_1.

In a **SecondStep**, we then construct E_2 the set of products of two distinct primes taking one prime factor as large as possible, and fill in a set RES product of two integers that lie outside [29]'s proof (lemma 1), but we wish to keep information on the primes we pick for E_2 and those we will not pick as those represent the necessary information to complete the missing elements of a RNS base of maximal size. When a product of two primes does not satisfy lemma 1, if this product belongs to I then we store it in a set RES, and if one of this two primes could appear in a product of three primes, which belongs to I, we store them in a set R. RES and R are used in the **ThirdStep**.

Construction of E_2, R, RES is done with Algorithm **SecondStep**. Note that in the case $\delta = \sqrt{I_{max}}$, RES is empty at the end of **SecondStep**.

Algorithm 2 SecondStep

```
Input: the remaining primes $P$, output of FirstStep
    while $P \neq \emptyset$ do
        $a \leftarrow \min(P)$ and $b \leftarrow$ PreviousPrime($\lceil \frac{I_{\max}}{a} \rceil$)
        if $ab \in I$ and $a \leq \delta$ and $b > \delta$ then        ▷ (lemma 1)
            puts $ab$ into $E_2$
            Remove $a, b$ from $P$ (if $b \in P$)
        else $s \leftarrow a^2$ and break.   ▷ $s$ the smallest product of
    two primes, is used for selecting elements for $R$
    while $P \neq \emptyset$ do
        $a \leftarrow \min(P)$ and $b \leftarrow$ PreviousPrime($\lceil \frac{I_{\max}}{a} \rceil$)
        Remove $a, b$ from $P$ (if $b \in P$)
        if $ab \in I$ and $a \leq \delta$ and $b > \delta$ then        ▷ (lemma 1)
            puts $ab$ into $E_2$
        else
            if $ab \in I$ and $a < b$ then store $ab$ in $RES$
            if $as \leq I_{\max}$ then store $a$ in $R$
            if $bs \leq I_{\max}$ then store $b$ in $R$
Output: $E_2, R, RES$
```

Example 4. *For $I = [2^{10} - 46, 2^{10}]$, $P = \{3, 5, \ldots, 41, 43\}$ and $E_1 = \{983, 991, 997, 1009, 1013, 1019, 1021, 1024\}$. We find $E_2 = \{(979 = 11 * 89), (995 = 5 * 199), (1003 = 17 * 59), (1011 = 3 * 337)\}$ with $R = \{7, 13, 19\}$ and $RES = \{(989 = 23 * 43)\}$. The break has occurred for $a = 7$ where we passed in the second while loop. When P is finally empty, $|E_2|$ has 4 elements, R has 3 elements and RES one.*
We have now simplified the problem to $\Phi(E_1 \cup E_2, I) \subseteq \Phi(\{\}, I)$, and the only elements we need to check are now combinations of elements of R and elements of RES.

After **SecondStep** we have constructed E_2 such that $\Phi(E_1 \cup E_2, I) \subseteq \Phi(E_1, I) \subseteq \Phi(\{\}, I)$. However, we *still* need to further process the remaining candidates in the case RES and/or R are not empty. We basically reconstruct I_S by bruteforce for $S = E_1 \cup E_2$ in order to reduce the search to $\Phi(E_1 \cup E_2, I) = \Phi(S, I_S)$ where I_S is essentially composed of the integers within I coprime to S which should be exactly RES and the missing combinations involving primes in R. Constructing I_S, if not proven to be empty, is done by algorithm **ThirdStep**, which stores in RES any possible product combination of the powers of the remaining primes within R fitting I.

Algorithm 3 ThirdStep

```
Input: $I_{min}$, $I_{max}$, $k$, $R$,$RES$
    if $k \leq I_{max}$ then
        if $k \geq I_{min}$ then $RES = RES \cup \{k\}$
        else
            if $R \neq \emptyset$ then
                $p \leftarrow \min(R)$
                $R \leftarrow R \backslash \{p\}$
                while $k \leq I_{max}$ do
                    $RES \leftarrow$ThirdStep($I_{min}$, $I_{max}$, $k$, $R$,$RES$)
                    $k \leftarrow k * p$
Output: $RES$
```

ThirdStep can have a very large complexity for large R. However, the recursion falls short when the elements of R are large: the level of recursion excluding calls to $K = 1$ is bounded by $\lceil \log_r(I_{\max}) \rceil$ for $r = \min(R)$. Thus it is very important that previous steps eliminate as many small primes as possible *before* **ThirdStep**.

Note that the output of **ThirdStep** does not theoretically guarantee a set of pairwise coprime integers since given primes p, q, x, w, integers $a = p \times q \times z$ 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.

Example 5. *With the example for $I = [2^{16} - 2^8, 2^{16}]$, Algorithm* ***SecondStep*** *gives $R = \{29, 37, 47, 53, 61, 71, 73\}$ and* $\mathrm{RES} = \emptyset$. *Luckily here, only one combination within elements of R can fit within I, giving $29 \times 37 \times 61 = 65453$. The maximum clique from a singleton is itself, thus we have actually our maximum clique and $E_1 \cup E_2 \cup \{65453\} \in \Phi(\{\}, I)$.*

The entire process is thus the concatenation of the three above algorithms, described in Algorithm 4 **FactorFilter**. Do note that at the end of Algorithm **SecondStep**, if both R and RES are empty, then C is a maximum clique and **ThirdStep** would actually do nothing as there is nothing to search within, as E_1 and E_2 have been constructed using only ″proven picks″ that do not reduce the size of an optimal solution.

Algorithm 4 FactorFilter

```
Input: $I_{\min}, I_{\max}$ the interval range.
Output: $C$ a set of pairwise coprime numbers, RES potential additions to grow $C$
    $E_1, P \leftarrow$ FirstStep($I_{\min}, I_{\max}$)
    $E_2, R, \mathrm{RES} \leftarrow$ SecondStep($I_{\min}, I_{\max}, P$)
    $C \leftarrow E_1 \cup E_2$
    RES$\leftarrow$ ThirdStep($I_{\min}, I_{\max}, R, 1, \mathrm{RES}$)
    return $C$, RES        ▷ If RES is empty, then $C$ is maximal
```

During the whole process, not a single gcd has been computed, nor was any value of I beyond its extremities needed. The remainder set I_S can also be emptied early i.e not constructed at all. Also note that the function **PreviousPrime**, while expensive in theory for arbitrary numbers in arbitrary applications, can be very efficiently called in our case if we already have access to the list of all necessary primes: we will never need a prime larger than $I_{\min}$ beyond the construction of E_1.

3.2 Second method: generalizing filtering for any set

This second proposed method works for any set, and is better than the first method on intervals anytime the set of integers I we need to manage is lower than the set of primes $I_{\mathbb{P}}$ we have to manage. To the best of our knowledge, there is no other comparable preexisting technique besides the direct clique method.

This second method relies on the following lemma:

Lemma 2 (Divisor pick for general sets)**.**

If $\exists y = ab \in I$ *and* $\{\gcd(y, x), x \in S\} = \{1\}$, *s.t* $\exists a > 1, \forall x \in I$, $a | \gcd(x, y)$ *or* $\gcd(x, y) = 1$.

Then $\Phi(S \cup \{y\}, I) \subseteq \Phi(S, I)$

This lemma can be understood by seeing that if such y exists, then for any element $\mathcal{M} \in \Phi(S, A)$, either there is an element divisible by a which can then be replaced by y without changing pairwise coprimality relations, either there is not and thus $\mathcal{M}$ is not maximal as you could include y (a contradiction). The method to find an element of $\Phi(S, I)$ is then the following:

1) Scan elements of I until such y is found. If no such y exists quit, else go to step 2
2) $S \leftarrow S \cup \{y\}$ and $I \leftarrow \{m \in I \text{ s.t } \gcd(y, m) = 1\}$, return to step 1.

The full process is described by Algorithm 5. Note that this filter can also be applied *after* the previous filter since the output of **ThirdStep** is also a set, thus our two algorithms are not mutually exclusive. After this filtering algorithm over I, to further increase the set size of S, we would need to call a maximum clique algorithm over the leftover I_S: however, in our experiments, I_S is often empty, leaving S as the final base. Unlike algorithm 4, algorithm 5 has a straightforward update of $\Phi(S, I) \leftarrow \Phi(S', I')$ after each loop iteration.

Example 6. *Let us select integers step-by-step: first let us set* $I = [968, 972, 3328, 1701, 875, 1445, 2873, 539, 493, 1573]$.

1) *The first integer,* 968 *has* gcd 4 *with* 3328 *and* 11 *with* 539. *Since* $\gcd(4, 11) = 1$, *we cannot select* 968.
2) *The same goes on for every integer up to* 539*: their possible* gcd *cannot be reduced to powers of a same prime.*
3) *We can however pick* 493*: it shares* 17 *as* gcd *with* 1445 *and* 2873*, and* 1 *with other integers. With this information, we update to* $S = \{493\}$, $I = [968, 972, 3328, 1701, 875, 539, 1573]$.
4) *We retry from* 968 *to* 1701 *with no success.*
5) 875 *has* gcd 7 *with both* 1701 *and* 539*, but* 1 *with others. With this information we can set* $S = \{493, 875\}$ *and* $I = [968, 972, 3328, 1573]$.
6) *We ignore* 968 *again, but pick* 972*, its* gcd *is* 4 *with* 968 *and* 3328 *but* 1 *with others. We update to* $S = \{493, 875, 972\}$ *and* $I = [1573]$.
7) *We finish with* $S = \{493, 875, 972, 1573\}$ *and* $I = \emptyset$, *which completes the search.*

Algorithm 5 Generic Filtering

```
Input: $I = \{m_i\}$ a set of integers
Output: $S \subset I$ an RNS base, and leftovers $I_S$
$S \leftarrow \{\}$
$I_S \leftarrow I$
while $I_S$ decreases in size do
    for $m \in I_S$ do
        $f \leftarrow m$                                  ▷ Initial common divisors
        for $m' \in I_S \backslash m$ 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                                ▷ $\#divisors < 2$
            $S \leftarrow S \cup \{m\}$
            $I_S \leftarrow \{m' \in I_S | \gcd(m, m') = 1\}$
            Break
return $S, I_S$
```

4 Fitering non-interval specific sets

We propose here another two filtering methods to complement those presented above, in case the target set is neither an interval, nor is it a completely random set. First, one called in this paper *the third method*, which introduces a modification of the first method. This third method adapted to work on *any* set of integers, which can replace the above two methods for very large sets (but does not outperform the previous methods otherwise). We will show that this method also guarantees maximality of the result. The second one, called fourth method, is a simpler, but distinct, method for extracting a RNS base from any random set. This method *does not guarantee maximality* of the result and therefore cannot replace any of the algorithms we propose nor the original work of [29]. However, the interesting part of this technique is its efficiency and simplicity and the possibility to *prove ad-hoc maximality* under certain conditions.

4.1 General Idea

Those new algorithms were designed to address some of the shortcomings of the previous two algorithms: one is effective for any given small set, the other starts to be effective for any given large interval but when the bounds have a similar binary size. Both algorithms do not give satisfactory results when the bounds of the interval have a different size, or when the set is a large combination of disjoint intervals. Note, that solving the problem efficiently under any set is difficult: any "maximum clique problem" can be transformed into an equivalent instance of a "largest pairwise coprime subset problem" in polynomial time, thus an efficient algorithm for any set is an efficient algorithm for a general NP-hard problem. However, we present here adapted algorithms when the entry sets are neither intervals nor completely random. Thus those supplementary algorithms are designed following two observations:

- The efficiency of the first method is limited when the small factors have *many* possibilities, failing to pick a choice and resorts to exhaustive search in the final steps of the algorithm.

- The efficiency of the second algorithm is limited by the fact that processing the factorization of *each* element of the entry set has to be done, thus limiting the size of the sets that can be reasonably be processed.

Thus we alleviate those issues with the following:

- Instead of just iterating on small factors, it might be better to also start with the *largest* factors, which are guaranteed to have *very limited* choices no matter how wide are the bounds. Choosing large factors first will also eliminate some small factors that could have had *many* solutions.
- If we have a low-cost function (in memory and speed) which determines whether an integer is part of the initial set or not, we could recompose by factors and use that function without ever storing the set. Whenever this is possible, we can achieve an efficiency gain on any *large* structured set over the generic method.

4.2 Third method: generalizing the recomposition technique

The third method reuses the philosophy of the interval-based algorithm. However as some properties on intervals might no longer hold on generic sets we have to make some generalizations.

4.2.1 Properties to justify maximality

As we are managing more diverse sets, we also need more general properties to justify our algorithms will still output a maximum size RNS.

Property 1 (Prime powers can always included)**.**
Let $a \in \mathbb{P}$*,* $\forall x \in S$ $\gcd(a, x) = 1$*, and* $\exists k \geq 1, a^k \in I$*.*
Then $\forall k \geq 1$ *such that* $a^k \in I$*,*
$\Phi(S \cup \{a^k\}, I \backslash \{x, \gcd(x, a) > 1\}) \subseteq \Phi(S, I)$*.*
Consequently, $\forall \mathcal{M} \in \Phi(S, I)$*,* $\exists m \in \mathcal{M}$ *such that* $a|m$*.*

The above property while obvious is necessary when filtering. The property below is a generalization of this and the lemma 1 of [29]:

Property 2 (Primes with limited representations)**.**
Let $a, b \in \mathbb{P}$*,* $(\forall x \in S, \gcd(ab, x) = 1)$*,*
$k, k' \geq 1$ *s.t* $a^k b^{k'} \in I$ *and* $(\forall m \in I,\ a|m \implies b|m)$*.*
Then $\Phi(S \cup \{a^k b^{k'}\}, I \backslash \{x \text{ s.t } \gcd(x, ab) > 1\}) \subseteq \Phi(S, I)$

Property 2 does not make any size arguments nor does it require unicity of a representation: in a sense it is also a specialization of lemma 2 for prime recompositions.

The two above properties does not make use of the strongest feature of the first method which is being able to exclude primes per size argument. However, we can have a close version of it since any finite set has a lower and upper bound. Let

$$P = \{p_1, ..., p_s\} \text{ where } p_1 < ... < p_s$$

be the remaining primes factors we can use to construct new elements of I that are coprime to our previously selected integers. Then

Property 3 (Excluding large elements)**.**
p_s *can be excluded whenever:*

- $p_1 p_s > I_{\max}$
- $\forall i \in [1, s]$*,* $p_i p_s \notin I$ *and* $p_1^2 p_s > I_{\max}$
- *Let* $X_3 = \{m = p_i p_j p_s \text{ s.t } \forall i, j \in [1, s],\ m < p_1^4\}$*.*
  If $\exists m \in X_3$ *s.t* $m > I_{\max}$ *and*
  $\forall m' \in X_3, m' < m \implies m' \notin I$

Intuitively, if we note X_k the set of combinations of k primes of R that are lower that p_1^{k+1}, we can generalize the property even further as: if for all $k' < k$ elements of $X_{k'}$ are not in I and X_k is as above then p_s can be excluded. However, computing those sets X_k is not trivial: for X_3, we only know that $p_1^2 < p_1 p_2 \leq p_i p_j$ for $i \neq j$ but we cannot compare $p_1 p_3$ and p_2^2 without actually computing their values. To avoid bruteforcing, we do not rely on the full property or its generalization.

4.2.2 Crafting the algorithm parts

The algorithm we craft here will roughly follow the same principles as the three-step method of recomposing by factors, with some slight modifications:

- First step gather *all* possible primes.
- Second step eliminate both largest and smallest primes simultaneously, iterating pincer-wise.
- Third step reconstruct all remaining candidates in I.

Let f_I be a function that outputs **TRUE** for an entry x if and only if $x \in I$. Note that $I_{\min}, I_{\max}$ the integral range can be used to construct f_I, even when I is not an interval: any finite set of numbers have a minimal and maximal value. We do not specify any f_I to keep a general algorithm, but keep in mind that this whole section relies on such a function to test whether an element is part of any given set I.

Instead of managing all the primes from $[2, I_{\{\max\}}] \cap \mathbb{P}$, we should only consider the primes in $I_{\mathbb{P}}$. For example, if I requires every factor to be $1 \mod 8$ or to be quadratic residues modulo some big prime, we can eliminate a lot of prime factors as preprocessing. However, for random generic sets, $I_{\mathbb{P}}$ is not always easily available and we can choose to take $\mathbb{P} \cap [2, I_{\max}]$ instead: the algorithm will be less efficient, but the end result will still allow to find a $M \in \Phi(\emptyset, I)$.

The reasoning used in previous algorithms are still valid and useful for optimization. Instead of storing all the primes into one set, we will split the storage into two sets for efficiency consideration: P_L will contain all the primes within $[2, \sqrt{I_{\max}}[$, and P_H will contain all the primes within $[\sqrt{I_{\max}}, I_{\max}]$. In practice, if we had a complete description of $I_{\mathbb{P}}$, we can possibly reduce those sets. For $a, b \in P_H$, we have $ab \notin I$: to create an element of I from $a \in P_H$, either $a \in I$ or one has to multiply a by elements of P_L. The first step is then as described in algorithm 6.

Example 7. *Let us take the example of:*
$I = \{a \in [2^{16} - 2^8, 2^{16}]$*, with* $c = 2^{16} - a$ *and* $hw(c) < 4\}$ *where* $hw(c)$ *is the number of 1 in the binary representation of c. We obtain* $E_1 = \{65407, 65519, 65536\}$ *and the sets of primes* $|P_L| = 42$ *and* $|P_H| = 56$ *whose numbers are smaller than if we do not check the belonging to* $I_{\mathbb{P}}$ *(where* $|P_L| = 53$ *and* $|P_H| = 169$*).*

Before we explain the second step, we briefly explain the quite simple exhaustive search for the undecided combinations in algorithm 7.

Algorithm 6 NewFirstStep, build initial primes

```
Input: $I_{\mathbb{P}}$, $f_I$ s.t $f_I(a) = \text{TRUE} \iff a \in I$
Output: Initial base $E_1$ and unused primes $P_L$, $P_H$.
$E_1 \leftarrow \{\}, P_L \leftarrow \{\}, P_H \leftarrow \{\}$
for $a \in I_{\mathbb{P}}$ do
    if $\exists k \geq 1$ s.t $f_I(a^k)$ then
        $E_1 \leftarrow E_1 \cup \{a^k\}$
    else if $a < \sqrt{I_{\max}}$ then
        $P_L \leftarrow P_L \cup \{a\}$
    else
        $P_H \leftarrow P_H \cup \{a\}$
return $E_1$,$P_L$,$P_H$
```

Algorithm 7 NewThirdStep, a recursive procedure

```
Input: $I_{\max}$ the set range, $R$ a list of leftover primes and $f_I$
    a function to verify inclusion in $I$
Output: RES contains the remaining clique candidates.
$R_2 = \{1\}$
for $p \in R$ do
    $R_3 = \{\}$
    for $p_2 \in R_2$ do
        $p_3 \leftarrow p_2$
        while $p_3 \leq I_{\max}$ do
            $R_3 \leftarrow R_3 \cup \{p_3\}$
            $p_3 \leftarrow p_3 p$
        $R_2 \leftarrow R_2 \cup R_3$
$RES = \{\}$
for $p \in R_2$ do
    if $f_I(p)$ then $\text{RES} \leftarrow \text{RES} \cup \{p\}$
return RES
```

Before we describe the middle procedure we will name **NewSecondStep** between **NewFirstStep** and **NewThirdStep**, we need to explain how we plan to select the valid combinations and when we can terminate **NewSecondStep** and move on to **NewThirdStep**. Basically, we also proceed by a step-by-step construct-or/and-reject using the above properties:

1) If $p_1 p_s > I_{\max}$, use 3, discard p_s and repeat.
2) If there is one unique i such that $p_i p_s \in I$ and $p_1^2 p_s > I_{max}$, then we use property 2 and update $S \leftarrow S \cup \{p_i p_s\}$, discard p_1, p_s and repeat.
3) If $\forall i, p_i p_s \notin I$ and $p_1^2 p_s > I_{max}$, then discard p_s and repeat (property 3).
   If $p_1^2 p_s \in I$ and $p_1 p_2 p_s > I_{\max}$, then discard p_1, p_s and update $S \leftarrow S \cup \{p_1^2 p_s\}$ (property 2).
4) If there is at least 2 distinct values i, j with $i < j < s$ such that $p_i p_s, p_j p_s \in I$, check if $p_1^2 p_s > I_{\max}$. If it is not the case, we move to **NewThirdStep**. If it is the case, we need to check if for all valid i at least one of them can be *easily computed to verify property 2*[1]: we can then update with $S \leftarrow S \cup \{p_i p_s\}$ and $R \cup R \backslash \{p_i, p_s\}$ and repeat. If not (or there is no easier way than bruteforce), then we move to **NewThirdStep**.

Hence, **NewSecondStep** presented in Algorithm 8. Note that this version does not make use of property 2, as we are currently unaware of a way to use it generically. Now that we have introduced in details all three core building blocks, we can build by concatenation the general algorithm **GenericFactorFilter** in algorithm 9.

Algorithm 8 NewSecondStep, build E_2 and filter primes

```
Input: $I_{\max}$, $P_L$, $P_H$ two sets of primes, $f_I$ to verify inclusion
    within $I$
Output: $E_2$, $P_L$, $P_H$ the leftover primes
$E_2 \leftarrow \{\}$
while $P_L \neq \emptyset$ and $P_H \neq \emptyset$ do
    $a \leftarrow \min(P_L)$, $b \leftarrow \max(P_H)$
    if $|P_L| > 1$ and $ab \times \min(P_L \backslash \{a\}) \leq I_{\max}$ then
        Break                        ▷ Too many choices to explore
    else
        $S \leftarrow \{s \in P_L, f_I(sb)\}$
        if $f_I(a^2 b)$ and $S \subseteq \{a\}$ then
            $E_2 \leftarrow E_2 \cup \{a^2 b\}$
            $P_H \leftarrow P_H \backslash \{b\}$, $P_L \leftarrow P_L \backslash \{a\}$
        else if $|S| = 0$ then
            $P_H \leftarrow P_H \backslash \{b\}$
        else if $f_I(a^2 b)$ or $|S| > 1$ then
            Break
        else
            $s \leftarrow \min(S)$
            $E_2 \leftarrow E_2 \cup \{sb\}$
            $P_H \leftarrow P_H \backslash \{b\}$, $P_L \leftarrow P_L \backslash \{s\}$
return $E_2$, $P_L$, $P_H$
```

Example 8. *Let us take the example of:*
$I = \{a \in [2^{16} - 2^8, 2^{16}]$, *with* $c = 2^{16} - a$ *and* $hw(c) < 4\}$ *where* $hw(c)$ *is the number of 1 in the binary representation of c. We obtain with* ***NewSecondStep***
$E_2 = \{65501 = 17 * 3853, 65503 = 31 * 2113, 65515 = 5 * 13103, 65523 = 3 * 21841, 65531 = 19 * 3449\}$ *and the sets of primes are now such* $|P_L| = 37$ *and* $|P_H| = 13$.
Then we apply ***NewThirdStep*** *with* $R = P_L \cup P_H$, *and* $K = 1$, *we obtain* $RES = \{65471 = 7 * 47 * 199, 65527 = 7 * 11 * 23 * 37, 65533 = 13 * 71^2\}$

Algorithm 9 GenericFactorFilter

```
Input: $I_{\max}$, $f_I$, $I_{\mathbb{P}}$
Output: $C$ a set of pairwise coprime numbers, RES potential additions to grow $C$
$E_1, P_L, P_H \leftarrow$ NewFirstStep($I_{\mathbb{P}}, f_I$)
$E_2, P_L, P_H \leftarrow$ NewSecondStep($I_{\max}, P_L, P_H, f_I$)
$C \leftarrow E_1 \cup E_2$, $R \leftarrow P_L \cup P_H$, RES $\leftarrow \{\}$
RES $\leftarrow$ NewThirdStep($I_{\max}, R, 1, \text{RES}, f_I$)
return $C$, RES                ▷ If RES is empty, then $C$ is maximal
```

4.3 Fourth method: pick first, prove later

The three previous methods relied on proving the maximality of the result on a step-by-step iteration. We present here a new approach: we cannot prove that our intermediate choice will leave an optimal result, but we can easily verify in some cases that the final result is indeed optimal. In the

[1] i.e, among the remaining combinations m, if $p_i | m \implies p_s | m$. Note that *easy* depends on the properties on the entry: for any generic set there is no easy way.

last method, we chose to separate the remaining primes into two sets P_L, P_H, after E_1 was constructed. Then we have the simple yet very effective property

Property 4 (Ad-hoc proof of maximality)**.**
Let E_1, P_L, P_H be the result of ***NewFirstStep*** *on some initial set of numbers I. Let $\mathcal{M} \subset I$ be a RNS base. Then*

$$|\mathcal{M}| = |E_1| + |P_L| \implies \mathcal{M} \in \Phi(\{\}, I)$$

We can trivially see that after **NewFirstStep**, only products of at least two distinct primes can form a valid element of I. By construction, elements of P_H cannot be powered or combined within each other (as we would have an element larger than $I_{\max}$). Thus, the largest possible outcome is for every other choice to be combining one element of P_L and one element of P_H. We assume P_L and P_H have their elements sorted in increasing order: the largest element is at the last position and the smallest element at the first position. We then provide some heuristic completion algorithms that do not guarantee optimality (but where *some* results can be proven to be maximal). The philosophy here is simple: construct couples $ab \in I$, regardless of whether or not a or b have multiple possibilities, and giving priority to the largest possible combinations. The algorithms can be constructed by swapping **NewSecondStep** by **GreedyMerge** (algorithm 10) when needed.

Algorithm 10 GreedyMerge

```
Input: $f_I$, $P_L$, $P_H$
Output: $E_2$ pairwise coprimes, $P_L$ and $P_H$ leftover primes
$E2 = \{\}$
$aP_L = P_L$
while $aP_L \neq \emptyset$ do
    $a \leftarrow \min(aP_L)$
    $aP_H = P_H$
    while $aP_H \neq \emptyset$ do
        $b \leftarrow \max(aP_H)$
        if $f_I(ab)$ then
            $E_2 \leftarrow E_2 \cup \{ab\}$,
            $P_L \leftarrow P_L \backslash \{a\}$, $P_H \leftarrow P_H \backslash \{b\}$
            break
        $aP_H \leftarrow aP_H \backslash \{b\}$
    $aP_L \leftarrow aP_L \backslash \{a\}$
return $E_2, P_L, P_H$
```

Example 9. *Let us take the example of:*
$I = \{a \in [2^{16} - 2^8, 2^{16}],$ *with* $c = 2^{16} - a$ *and* $hw(c) < 4\}$ *where $hw(c)$ is the number of 1 in the binary representation of c.*
We obtain with ***GreedyMerge***
$E_2 = \{65501 = 17 * 3853, 65503 = 31 * 2113, 65515 = 5 * 13103, 65523 = 3 * 21841, 65531 = 19 * 3449\}$ *and the sets of primes are now such $|P_L| = 37$ and $|P_H| = 51$.*
Then we apply ***NewThirdStep*** *with $R = P_L \cup P_H$, and $K = 1$, we obtain* $RES = \{65471 = 7 * 47 * 199, 65527 = 7 * 11 * 23 * 37, 65533 = 13 * 71^2\}$

5 Construction of elementary RNS bases

5.1 Base of Pseudo-Mersenne

In this section, we present the use of our basic alternative filtering techniques on Pseudo-Mersenne Numbers. 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 [38]. We are continuing the work of [29] for pseudo-Mersenne integers within an interval we denote solely for this section $I_n = [2^n - c, 2^n]$, for simplicity, we keep n even to have a simple computation of $c = 2^{n/2}$ for now and only attempt to find the size of a RNS base $\mathcal{M} \in \Phi(\emptyset, I_n)$. We conduct experiments for $n \in [16, 64]$. Let R_n be the set R outputted by **SecondStep** when the entry set is I_n and the primes given were already filtered out by *FirstStep*. We list the number of primes within R_n in Table 1. We can observe from Table 1 that the size of R_n is low enough to make **ThirdStep** inexpensive. Note that $R_n = \emptyset$ is the result of **SecondStep** and directly implies that $\{E_1 \cup E_2\} \in \Phi(E_1 \cup E_2, I_n) \subseteq \Phi(\emptyset, I_n)$.

Table 1
Size of R_n, output of **SecondStep**

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

The *only* cases where **ThirdStep** did not output an empty set are the following:

- 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.

We could conjecture that for $n \geq 40$ then $R_n = \emptyset$ in such cases but as the density of prime numbers vanishes as n grows we prefer not to.

We present some results for E_1, E_2 and $d = |\mathcal{M}|$ for $\mathcal{M} \in \Phi(\emptyset, I_n)$

Table 2
Set size of E_1, E_2, and $\max$ RNS set size d for even $n \in [16, 64]$

n	$E_1 \backslash \{2^n\}$	E_2	max size
16	21	25	48
24	251	198	450
32	2931	1851	4783
40	37798	19856	57655
48	504634	226507	731142
56	6920100	2724323	9644424
64	96798093	34267158	131065252

For c small, as $c = 256$, the amount of elements is not larger than the number of primes to consider, thus the **Generic Filtering** is a better choice. We present our experimental results on $c = 256$ in Table 3, listing the maximal size $d = |\mathcal{M}|$ for $\mathcal{M} \in \Phi(\emptyset, [2^n - 256, 2^n])$.

Table 3
Maximum set size d for $m_i \in [2^n - 2^8, 2^n]$ 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	

5.2 Base of Solinas numbers

In this part we exhibit our results for the research of Solinas numbers: while every number can be written as $x = \sum x_i 2^i$ where $x_i \in \{-1, 0, 1\}$, Solinas numbers are such that the amount of $x_i \neq 0$ is low, where w called the weight of x is the number of non-zero x_i. [39] exhibited how to use those numbers efficiently: in particular, they showed how one can avoid using multiplications using those numbers and replace them by more efficient additions, shifts and substractions. However, [39] only exhibited RNS bases with 6 integers.

We use the **Generic Algorithm** to determine d_w the size of the elements of $\Phi(\emptyset, I_{n,w})$ where $I_{n,w}$ are the elements of $[2^n - 2^{n/2}, 2^n]$ with weight w. Our experimental results are presented in 4.

Table 4
Maximum sizes of $d_3, d_4 \in I_n$ for even $n \in [16, 1024]$

n	16	32	48	64	128	256	512	1024
d_3	11	20	29	30	62	81	180	281
d_4	24	90	178	325	-	-	-	-

6 Construction of specific RNS bases

6.1 Base of Montgomery-friendly numbers

This section considers "Montgomery-friendly" numbers: introduced in [40], [41], [42], they exhibit efficient properties whenever Montgomery reduction is involved within the computations. [33] showcased that those numbers can have comparable or better performance and scalability compared to pseudo-Mersenne numbers, focusing on 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 $m_i = 2^n - 1$).

We use the *Generic Filtering* to further improve their results, showcasing that their approach has better scalability than they previously showed, exhibiting the size d of the elements of $\Phi(\emptyset, I)$ while [33]'s found sizes $|\mathcal{M}|$ are sometimes two times smaller: see the comparison table in Table 5. Parameters n, k, t are taken straight of section 5.3 of [33].

Table 5
Maximum size of Montgomery-friendly RNS bases.
bold numbers show where we improve upon [33]

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
Found d	**70**	**122**	**122**	**8**	**21**	**64**	**214**	**255**	**255**
[33]'s $\lvert\mathcal{M}\rvert$	**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
Found d	**8**	**8**	7	**22**	**70**	**214**	**705**	**1319**	**2401**
[33]'s $\lvert\mathcal{M}\rvert$	**7**	**-**	7	**21**	**65**	**205**	**688**	**1295**	**2365**

6.2 Base of quadratic residues

Previous examples focused on finding exact elements of $\Phi(\emptyset, I)$. In some cases, the problematic is different for finding large RNS bases: coprimality is nothing more than one condition rather than the sole condition for a base to be valid. Such is the case of the RNS bases required by [26]: they exhibited powerful algorithms, tested on FPGA, which outperforms state-of-the-art algorithms at that time. However, specific bases with further requirements were necessary. While they managed to exhibit basis that were large enough for some practical applications, the scalability of their approach was relatively unknown: researching better bases was left as an open question. [26] proposed two algorithms: sQ-RNS and dQ-RNS, with two specific sets of conditions. While we cannot prove to find optimal bases, we show we can still improve the currently known results.

6.2.1 Q-RNS Definitions

The work of [26] relies on a bivariate function (close to the Jacobi symbol), the QR function, which creates asymetric relations between the moduli: it is the main reason simple graph algorithms will not work.

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

Elements of a basis $\mathcal{M}$ have to be of the form $m_i = 2^n - c$. Previously, we either aimed to maximize the size d of $\mathcal{M}$. Here, we also aim to bound c to $c < 2^k$ if $|\mathcal{M}| = d$ is fixed.

6.2.2 sQ-RNS

sQ-RNS requires two distinct bases $\mathcal{M} = \{m_1, ..., m_d\}$ and $\mathcal{M}' = \{m'_1, ..., m'_d\}$, with $M = m_1..m_d$ and $M' = m'_1...m'_d$ such that

$$\prod_{i \neq j} \mathrm{QR}(m_j, m_i)\mathrm{QR}(m'_j, m'_i) \prod_{\forall i} \mathrm{QR}(p, m_i) \prod_{\forall i,j} \mathrm{QR}(m'_j, m_i)$$

is equal to 1. The prime p is usually fixed by the (cryptographic) system we plan to use the base for. Since $\mathrm{QR}(M', M)\mathrm{QR}(p, M) = 1$, we believe $\mathcal{M}$ should be generated first: p, which is fixed, influences M the same way M influences M'. Our strategy is then the following:

- Create $\mathcal{M}$ s.t $\mathrm{QR}(p, M) \prod_{i \neq j} \mathrm{QR}(m_i, M/m_i) = 1$
- Then $\mathcal{M}'$ s.t $\mathrm{QR}(M', M) \prod_{i \neq j} \mathrm{QR}(m'_i, M'/m'_i) = 1$

First, we aim to construct $\mathcal{M}$ minimizing k given values d, p fixed by the (crypto)system verifying:

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

In order to do so, we increment k from $k = 1$ until the process can successfully complete: first filter with p, k to verify 1), then apply *generic filtering* to find moduli verifying 2) and repeat until we have d moduli m_i to verify 3).

Once a suitable $\mathcal{M}$ is found, $\mathcal{M}'$ is constructed similarly by incrementing from $k' = k$ given $\mathcal{M}$ until we can verify the following conditions:

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

Table 6
Results for maximum size found for sQ-RNS bases

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 7
Values c_i, c'_i and their size k for sQ-RNS with $d = 4$ and same n. Our work is in **bold**, [26] in *ITALIC*

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*
*	**3**	**9**	**19**	**23**	**31**	**85**	**211**	**231**	**5**	**8**
	535	*2191*	*3219*	*8031*	*49*	*751*	*979*	*11,335*	*13*	*14*

*: Curve25519

6) At least d different values

Once the process is finished, we have obtained valid sQ-RNS bases $\mathcal{M}, \mathcal{M}'$. We cannot prove our result is optimal, but experimentally we still improve the results given by section 4.4 of [26]: table 6 shows larger sizes d can be found given (n, k), and table 7 shows smaller k for fixed (n, d).

6.2.3 dQ-RNS

As far as we understood, dQ-RNS was conceived to make base searching easier than sQ-RNS. The condition for dQ-RNS was to obtain a single base $\mathcal{M}$ of size $2d$ such that $\mathrm{QR}(p, M) = 1$ with moduli m_i being square numbers. In that regard, our previous approach still works except we ignore the construction of $\mathcal{M}'$ and focus on a larger $\mathcal{M}$.

This simplification can lead to different approaches to find dQ-RNS bases. For example, to research two RNS base size of d for dQ-RNS within a set of pseudo-mersenne numbers I (defined by (n, k)) given p, we could:

1) Remove all $m \in I$ such that $\mathrm{QR}(p, m) \neq 1$ from I.
2) Remove all $m \in I$ such that m is not square from I.
3) Find, if possible $\mathcal{M}'' \in \Phi(\emptyset, I)$ with $|\mathcal{M}''| \geq 2d$.
4) Split $\mathcal{M}''$ into $\mathcal{M}, \mathcal{M}'$ such that $|\mathcal{M}| = |\mathcal{M}'| = d$

Using our filtering methods on both techniques, we should be able to improve the known scalability of [26] through larger RNS bases. The work of [26] is non-trivial, and obtaining *provably maximal* bases without an exhaustive search seems to be a hard task at hand, and we leave this as an interesting open question.

7 Supplementary applications

In this section we provide some examples applications where the third and fourth method can extract RNS bases more efficiently.

7.1 Large Interval of many possible bitsizes

It is known that for $[2, p]$ the largest subset of pairwise coprime numbers are basically all the primes until p. We previously treated the cases $[2^n - 2^{n/2}, 2^n]$, we are now considering the cases $[2^{n-2}, 2^n]$: experimentally, the final graph given by the first method on those sets are not empty, simply because unicity of representation cannot be guaranteed in intermediate steps. In particular, we can show in those specific examples that *NewFirstStep+GreedyMerge* outperforms *NewFirstStep+NewSecondStep*, while proving maximality of the result using property 4 (i.e no need for further processing).

Table 8
E_2's size within $I = [2^{n-2}, 2^n]$ after *NewFirstStep* constructs E_1

n	16	17	18	19	20	21	22	23	24
GreedyMerge	24	32	41	59	81	110	152	204	282
NewSecondStep	23	31	30	50	64	109	123	183	216

7.2 Large Intersection of many intervals

Let $I = I_1 \cap ... \cap I_k$. If I is large, using the generic filtering is not ideal, and the first method does not apply whenever I is not an interval: we can imagine I being a mix of Montgomery-friendly integers and pseudo-mersenne integers of different sizes, or more exotic large sets. In this case, we can take a simple arbitrary example where I is a mix between pseudo-Mersenne numbers of various sizes:

$$I = \{a \in [2^k - c, 2^k] \mid k \in [19, 21], c < 2^{16}\}$$

It is very easy to verify whether or not an integer belongs to I, and it also seems that on this example the heuristic attempt using **GreedyMerge** also directly outputs an optimal RNS basis giving E_2 of size 210 for a total RNS base of 229 moduli (E_1 has size 19). However, there is a limit to this approach over the generic approach: the amount of primes to manage can be much larger than $|I|$. If that is the case, the *Generic Filtering* method (algorithm 5) could be preferred.

7.3 RNS base coprime with a very large smooth integer

This application is more interesting outside the field of cryptography, where we attempt to compute ring operations over a non-prime moduli that might be smooth. In cryptography we usually have to deal with large primes, thus we usually focus on constructing on RNS bases on small moduli that are guaranteed to be coprime to the large prime (to keep operations valid). Suppose for example we want to simplify operations on random rings $\mathbb{Z}_N$ using a RNS approach where N's full factorization is unknown: we then have to create novel RNS bases on the fly for every new N given. We can reuse our algorithms by preselecting primes, excluding small prime numbers that are not coprime with N. If a largest base is not needed but rather a base with a total bitsize of $2\log_2(N)$, finding good RNS basis on the fly could be done more efficiently than with previous methods.

Table 9
Time Comparison

Set	Factors (Alg 9)	Generic (Alg 5)	Filtering [29]	Clique [29]	New Greedy (Alg 10)	Greedy [26], [33]
$[2^9, 2^{10}]$	<**0.01s**	0.06s	< 0.01s	> 1h	< 0.01s	(< 0.01s)
$[2^{10}, 2^{11}]$	<**0.01s**	0.11s	86s	-	< 0.01s	(< 0.01s)
$[2^{11}, 2^{12}]$	<**0.01s**	0.34s	2612	-	0.12s	(0.01s)
$[2^{13}, 2^{14}]$	**0.01s**	30s	-	-	0.12s	(0.01s)
$[2^{14}, 2^{15}]$	**8.76s**	141s	-	-	0.02s	(0.44s)
$[2^{24} - 2^{12}, 2^{24}]$	13.4s	0.26s	**0.03s**	-	(16.7s)	(0.05s)
$[2^{28} - 2^{12}, 2^{26}]$	1257s	0.36s	**0.05s**	-	(811s)	(0.08s)
$[2^{52} - 2^{12}, 2^{64}]$	-	**0.24s**	156s	-	-	(0.07s)
$[2^{512} - 2^{12}, 2^{512}]$	-	**0.34s**	> 1h	-	-	(0.07s)
10% of $[2^{18}, 2^{19}]$	**0.5s**	48s	n/a	-	(0.21)	(1.22s)
50% of $[2^{16}, 2^{17}]$	**0.52s**	154s	n/a	-	(0.05s)	(1.7s)
80% of $[2^{15}, 2^{16}]$	**4.9s**	248s	n/a	-	0.02s	(1.14s)

— is indicating when computational time was too long to be relevant
() is indicating when algorithm did not return optimal basis

bold is indicating the best algorithm for corresponding set

7.4 Comparison

We present in Table 9 computation time of state of the art methods as well as our new proposals. Our test were perfromed on a $i7 - 8565$ CPU based laptop and implemented in MAGMA[2]. We perform our test on 3 types of set of moduli.

- First, we simply grow the size of our set of moduli to show the impact of the size of the initial set on each algorithm.
- Secondly, we keep a fix size, but we increase the size of the moduli themselves to show how it impacts those methods.
- Finally, we change the density of moduli in a set, once again to show its impact.

The first four algorithms (from left to right in the table) guarantee an optimal basis, the last two do not. Note that we do not test Alg 4 as some of the samples are not intervals, but test instead Alg 9 as it is simply a slightly slower generic version.

Table 9 clearly indicates that the best choice:

- when the size of the set is large, is to recompose by factors when available
- when the size of elements is large, is to apply the generic technique for random sets

We note also that our new greedy algorithm allows to often obtain optimal bases in an efficient time if the set is reasonably dense. Previous greedy solutions rarely give an optimal basis, whatever is the type of sets studied.

8 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.

In this extended version, we also presented two novel algorithms, offering more adapted tools to situations the previous works could not handle. One of those algorithms is heuristic and can prove maximality only under certain conditions, which differs from the previous approaches. Those can also still be improved and provide applications that can apply not just to cryptography but to practical integer computer arithmetic in the broad sense, for example computations under integral rings whenever the moduli are random and extremely large.

References

[1] L. Sousa, "Nonconventional computer arithmetic circuits, systems and applications," *IEEE Circuits and Systems Magazine*, vol. 21, no. 1, pp. 6–40, 2021.

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

[3] H. L. Garner, "The residue number system," in *western joint computer conference*. ACM, 1959, pp. 146–153.

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

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

[6] S. Kawamura, Y. Komano, H. Shimizu, S. Osuka, D. Fujimoto, Y. Hayashi, and K. Imafuku, "Efficient algorithms for sign detection in rns using approximate reciprocals," *IEICE TRANSACTIONS on Fundamentals of Electronics, Communications and Computer Sciences*, vol. 104, no. 1, pp. 121–134, 2021.

[7] K. Isupov and V. Knyazkov, "Interval estimation of relative values in residue number system," *Journal of Circuits, Systems and Computers*, vol. 27, no. 01, p. 1850004, 2018.

[8] M. Babenko, A. Tchernykh, and V. Kuchukov, "Improved modular division implementation with the akushsky core function," *Computation*, vol. 10, no. 1, p. 9, 2022.

[9] G. A. Jullien and W. C. Miller, "Application of the residue number system to computer processing of digital signals," in *ARITH*, 1978.

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

[11] A. Hiasat, L. Sousa, and A. F. Anta, "On the design of RNS intermodulo 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.

[12] 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.

[13] J. van der Hoeven, "Fast chinese remaindering in practice," in *7th Intl Conf (MACIS)*, ser. LNCS, vol. 10693, 2017, pp. 95–106.

[14] D. Schoinianakis, "Residue arithmetic systems in cryptography: a survey on modern security applications," *Journal of Cryptographic Engineering*, vol. 10, no. 3, pp. 249–267, 2020.

[15] K. Bigou and A. Tisserand, "Single base modular multiplication for efficient hardware RNS implementations of ECC," in *CHES*, 2015.

[2] http://magma.maths.usyd.edu.au/magma

[16] S. Asif and Y. Kong, "Highly parallel modular multiplier for elliptic curve cryptography in residue number system," *Circuits, Systems, and Signal Processing*, vol. 36, no. 3, pp. 1027–1051, 2017.

[17] S. Duquesne, "Rns arithmetic in $\mathbb{F}_{p^k}$ and application to fast pairing computation," *Journal of Mathematical Cryptology*, vol. 5, no. 1, pp. 51–88, 2011.

[18] S. Halevi, Y. Polyakov, and V. Shoup, "An improved rns variant of the bfv homomorphic encryption scheme," in *CT-RSA*, 2019.

[19] A. Kim, Y. Polyakov, and V. Zucca, "Revisiting homomorphic encryption schemes for finite fields," in *ASIACRYPT*, 2021.

[20] 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.

[21] S. Kawamura, M. Koike, F. Sano, and A. Shimbo, "Cox-rower architecture for fast parallel montgomery multiplication," in *EUROCRYPT*, 2000.

[22] L. Djath, K. Bigou, and A. Tisserand, "Hierarchical approach in rns base extension for asymmetric cryptography," in *ARITH*, 2019.

[23] J. Ahsan, M. Esmaeildoust, A. Kaabi, and V. Zarei, "Efficient fpga implementation of rns montgomery multiplication using balanced rns bases," *Integration*, 2022.

[24] 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 *CHES*, 2011.

[25] A. Lesavourey, C. Nègre, and T. Plantard, "Efficient leak resistant modular exponentiation in RNS," in *ARITH*, 2017.

[26] 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.

[27] J.-C. Bajard, J. Eynard, and N. Merkiche, "Multi-fault attack detection for RNS cryptographic architecture," *ARITH*, 2016.

[28] J. Courtois, L. Abbas-Turki, and J.-C. Bajard, "Resilience of randomized rns arithmetic with respect to side-channel leaks of cryptographic computation," *IEEE Transactions on Computers*, vol. 68, no. 12, pp. 1720–1730, 2019.

[29] B. Gérard, J. Kammerer, and N. Merkiche, "Contributions to the design of residue number system architectures," in *ARITH*, 2015.

[30] K. Bigou and A. Tisserand, "Hybrid position-residues number system," in *ARITH*, 2016.

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

[32] E. Tomita, "Efficient algorithms for finding maximum and maximal cliques and their applications," in *WALCOM*, 2017.

[33] J. C. Bajard and S. Duquesne, "Montgomery-friendly primes and applications to cryptography," *Journal of Cryptographic Engineering*, vol. 11, no. 4, pp. 399–415, 2021.

[34] J. C. Bajard, K. Fukushima, S. Kiyomoto, A. Sipasseuth, T. Plantard, A. Sipasseuth, and W. Susilo, "Generating residue number system bases," in *ARITH*, 2021.

[35] T. O. e Silva, "Tables of values of pi(x) and of pi2(x)," 2018, http://sweet.ua.pt/tos/primes.html.

[36] Chris K. Caldwell, "The first fifty million primes," 2021, available at https://primes.utm.edu/lists/small/millions/.

[37] https://primesieve.org, "primesieve, a prime number generator."

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

[39] J. C. Bajard, M. Kaihara, and T. Plantard, "Selected RNS bases for modular multiplication," in *ARITH*, 2009.

[40] M. Hamburg, "Fast and compact elliptic-curve cryptography," Cryptology ePrint Archive, Report 2012/309, 2012.

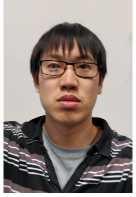
[41] J. W. Bos, C. Costello, H. Hisil, and K. Lauter, "Fast cryptography in genus 2," in *EUROCRYPT*, 2013.

[42] 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.

Jean Claude Bajard is professor at Sorbonne Universite. He is member of the Ouragan Inria team and of the Institut de Mathématiques de Jussieu-Paris Rive Gauche. His research area is computer arithmetic.

Kazuhide Fukushima is a senior manager at the information security laboratory of KDDI Research Inc. in Japan, focusing on post-quantum cryptography, symmetric-key cryptography, and AI/ML security.

Thomas Plantard is a cryptographic researcher. His research focus is on post-quantum cryptography and computer arithmetic for cryptography.



Arnaud Sipasseuth is a researcher at KDDI Research Inc in Japan, focusing mainly on computer arithmetic and their application to cryptography.