DOI: 10.15514/ISPRAS-2025-37(5)-3

Generating Compact Residue Number Systems Bases

V.V. Lutsenko, ORCID: 0000-0003-4648-8286 <officialvladlutsenko@gmail.com>
M.G. Babenko, ORCID: 0000-0001-7066-0061 <mgbabenko@ncfu.ru>
North-Caucasus Federal University, Stavropol,
1, Pushkin st., Stavropol, 355017, Russia.

Abstract. Modern computational tasks involving large-number processing demand not only high precision but also significant operational speed. In this context, the residue number system provides an effective approach for parallel processing of large numbers, with applications in cryptography, signal processing, and artificial neural networks. The primary task in defining such a system is determining its basis. This paper presents an algorithm for generating compact residue number system bases based on the Diemitko theorem. The proposed algorithm generates bases 15.5% faster on average than Pseudo-Mersenne-based construction and 75.7% faster than the general filtering method. Comparative analysis demonstrates that using compact bases delivers an average 12% acceleration in modular operations compared to special moduli sets.

Keywords: residue number system; high-performance computing; special sets of moduli; generation of prime numbers; cryptography.

For citation: Lutsenko V.V., Babenko M.G. Generating compact residue number systems bases. Trudy ISP RAN/Proc. ISP RAS, vol. 37, issue 5, 2025. pp. 43-52. DOI: 10.15514/ISPRAS-2025-37(5)-3.

Acknowledgements. The research was supported by the Russian Science Foundation Grant No. 25-71-30007, https://rscf.ru/en/project/25-71-30007/.

Генерация компактных базисов системы остаточных классов

В.В. Луценко, ORCID: 0000-0003-4648-8286 <officialvladlutsenko@gmail.com>
М.Г. Бабенко, ORCID: 0000-0001-7066-0061 <mgbabenko@ncfu.ru>

Северо-Кавказский федеральный университет,
Россия, 355017, г. Ставрополь, ул. Пушкина, д. 1.

Аннотация. Современные вычислительные задачи, связанные с обработкой больших чисел, требуют не только высокой точности, но и значительной скорости операций. В данном контексте применение системы остаточных классов предлагает подход к параллельной обработке больших чисел, который применяется в криптографии, обработке сигналов и искусственных нейронных сетях. Ключевой задачей при построении системы остаточных классов является определение её базиса. В статье представлен алгоритм генерации компактных базисов системы остаточных классов, основанный на теореме Диемитко. Предложенный алгоритм генерирует базисы в среднем на 15,5% быстрее, чем построение базисов на основе псевдо-Мерсенновских чисел, и на 75,7% быстрее, чем метод общей фильтрации. Проведённый сравнительный анализ показал, что использование компактных базисов обеспечивает в среднем 12% ускорение модульных операций по сравнению со специальными наборами модулей.

Ключевые слова: система остаточных классов; высокопроизводительные вычисления; специальные наборы модулей; генерация простых чисел; криптография.

Для цитирования: Луценко В.В., Бабенко М.Г. Генерация компактных базисов системы остаточных классов. Труды ИСП РАН, том 37, вып. 5, 2025 г., стр. 43–52 (на английском языке). DOI: 10.15514/ISPRAS–2025–37(5)–3.

Благодарности. Исследование выполнено за счет гранта Российского научного фонда № 25-71-30007, https://rscf.ru/project/25-71-30007/.

1. Introduction

Modern computational problems involving the processing of large numbers require not only high accuracy but also significant speed of operations. In this context, unconventional arithmetic offers innovative approaches that optimize computation in various areas such as cryptography, signal processing and theoretical computer science. One of the key tools in this area is the Residual Number System (RNS), which dates back to the 1950s and is based on the Chinese Remainder Theorem (CRT) [1]. RNS is an alternative way of representing numbers based on modular arithmetic. Instead of dealing with numbers in a positional representation, RNS decomposes them into a set of residues obtained by division by pairwise prime numbers, called the RNS basis [2]. The main advantage of RNS is that the addition and multiplication operations are performed in parallel on each residue, which greatly speeds up the computation. Despite the cost of the inverse transformation, which in the worst case depends quadratically on the size of the basis, computations involving addition and multiplication become extremely performant. However, inverse transformation, division and comparison of numbers in RNS remain computationally challenging problems, but recent works propose efficient algorithms for these tasks [3-5].

RNS has been applied in signal processing [6], cryptography [7], and neural networks [8]. The research presented in this paper is relevant to a wide range of applications related to large number processing, including cryptographic systems since the 1990s [9], such as RSA, DH, ECC [10-11], as well as pairing methods, Euclidean lattice-based algorithms and homomorphic protocols [12].

In cryptography, where arithmetic operations are performed modulo large numbers that are often prime, the application of RNS becomes more challenging due to the need to perform modulo taking, which has led to active research aimed at selecting optimal bases to improve implementation efficiency [13-14]. RNS is also of particular interest for defense against error injection attacks, as the introduction of redundant elements at the basis level allows error detection mechanisms to be

organized [15]. In addition, the random choice of basis provides a different representation of data at each computation, which complicates the analysis of possible information leaks. Thus, the choice of basis is the first and most important task of RNS.

In this paper we present a method for generating RNS bases based on Diemitko's theorem. This approach allows us to obtain bases satisfying the compactness condition.

The article is structured as follows. Following the introduction, Section 2 covers the fundamentals of RNS. Section 3 reviews related works. Section 4 presents the proposed algorithm for generating compact RNS bases. Section 5 then evaluates the algorithm's performance. Finally, the key findings are summarized in the conclusion.

2. Residue Number System

RNS is based on the widely known CRT [16]. RNS argues that, knowing the smallest non-negative residues from dividing an integer X by the integer moduli $p_1, p_2, \dots, p_n$ it is possible to uniquely determine the residue from dividing X by the product of these moduli, provided that the moduli are pairwise coprime. RNS, unlike classical b-ary number systems, is not defined by a single fixed base, but by a set of moduli $\{p_1, p_2, \dots, p_n\}$ such that $gcd(p_i, p_j) = 1$ for all $i, j \in 1,2, \dots n, i \neq j$, where $gcd(\ \ )$ is the greatest common divisor. The product of these moduli $P = \prod_{i=1}^{n} p_i$ determines the dynamic range of the RNS. An integer $X \in [0, P)$ is represented as a vector composed of the smallest non-negative residues obtained by dividing X by p_i:

$$X = (x_1, x_2, \dots, x_n). \quad (1)$$

where $x_i = X(mod\ p_i)$, which is also denoted by $x_i = |X|_{p_i}$.

Consider RNS with the basis $\{4,5,7\}$. In this basis, we can mutually uniquely represent the numbers from the half-interval $[0; 140)$, since $P = 140$.

Table 1 shows the correspondences of numbers from the positional number system and the RNS.

Table 1. Representation of Numbers for RNS with the Basis {4,5,7}.

$0 \overset{RNS}{\longrightarrow} (0,0,0)$	$1 \overset{RNS}{\longrightarrow} (1,1,1)$	$2 \overset{RNS}{\longrightarrow} (2,2,2)$	$3 \overset{RNS}{\longrightarrow} (3,3,3)$
$4 \overset{RNS}{\longrightarrow} (0,4,4)$	$5 \overset{RNS}{\longrightarrow} (1,0,5)$	$6 \overset{RNS}{\longrightarrow} (2,1,6)$	$7 \overset{RNS}{\longrightarrow} (3,2,0)$
$8 \overset{RNS}{\longrightarrow} (0,3,1)$	$9 \overset{RNS}{\longrightarrow} (1,4,2)$	$10 \overset{RNS}{\longrightarrow} (2,0,3)$	$11 \overset{RNS}{\longrightarrow} (3,1,4)$

RNS defines basic operations on numbers, which are divided into two groups. The operations of the first group, which are sometimes called modular, include addition and subtraction of numbers without the possibility of determining the sign of the result, as well as multiplication. Such operations are performed component-wise on remainders, i.e. without forming carryovers between them. Let the numbers X, Y and R be represented as $(x_1, x_2, \dots, x_n)$, $(y_1, y_2, \dots, y_n)$ and $(r_1, r_2, \dots, r_n)$, respectively. Then for any modular operation $\circ$ we have

$$X \circ Y = (r_1, r_2, \dots, r_n), \quad (2)$$

where $r_i = |x_i \circ y_i|_{p_i}$.

Thus, the i-th digit of the result in RNS, r_i, is defined only in terms of $|x_i \circ y_i|_{p_i}$ and does not depend on any other digit r_j. This allows the realization of carry-free, high-speed (parallel) computer arithmetic and makes RNS an attractive number system for use in resource-intensive applications, especially those involving the processing of large numbers. It also provides high computational reliability since an error in the *i*-th digit has no effect on other digits and therefore can be efficiently localized and eliminated [15]. In turn, for operations of the second group, often called non-modular, it is not enough to know the values of individual residues and requires an estimate of the magnitude of numbers: the result of such an operation is either not a number in RNS at all, or the value of each

of its digits (residue) is not only a function of the values of the corresponding digits of the operands, but depends on the magnitude of these operands.

Algorithm 1 presents a method for performing modular operations in RNS.

Algorithm 1: Modular operations in RNS.

Input: $\{p_1, p_2, \dots, p_n\}, (x_1, x_2, \dots, x_n), (y_1, y_2, \dots, y_n), \circ \in \{+, -, \times\}$
Output: $R = (r_1, r_2, \dots, r_n)$
1. **for** $i = 1, i \le n, i++$ **do**:
1.2 $r_i = |x_i \circ y_i|_{p_i}$

Here is an example of addition in RNS.

Example 1 (Addition in RNS). Let us add two numbers $X = (2, 0, 3)$ and $Y = (3, 0, 1)$ in the basis {4,5,7}. Use (2) for addition:

$$X + Y = (|2 + 3|_4, |0 + 0|_5, |3 + 1|_7) = (1,0,4)$$

Hence $R = (1,0,4)$. Which is true since $25 \xrightarrow{RNS} (1, 0, 4)$.

3. Related Works

The choice of modulo set is very important to achieve a suitable RNS implementation. The modulo set affects the whole RNS architecture [17]. Special sets of moduli are widely used [18]. Table 2 presents the most well-known special sets of moduli.

Table 2. Special sets of moduli.

Number	Set	Year
1	$\{2^n - 1, 2^n, 2^n + 1\}$	1967
2	$\{2n - 1, 2n, 2n + 1\}$	1995
3	$\{2^{2n} + 1, 2^n + 1, 2^n - 1\}$	1997
4	$\{2^n - 1, 2^n, 2^{n-1} - 1\}$	1998
5	$\{2^n - 1, 2^n, 2^{n+1} - 1\}$	1999
6	$\{2^n - 1, 2^n, 2^{2n+1} - 1\}$	2008
7	$\{2^n - 1, 2^n, 2^{2n} + 1\}$	2008
8	$\{2^\alpha, 2^\beta - 1, 2^\beta + 1\}$	2008
9	$\{3^n - 2, 3^n - 1, 3^n\}$	2007
10	$\{2^n - 1, 2^n, 2^n + 1, 2^{n+1} + 1\}$	1999
11	$\{2^n - 1, 2^n, 2^n + 1, 2^{n+1} - 1\}$	2000
12	$\{2^n - 1, 2^n, 2^n + 1, 2^{2n} - 1\}$	2003
13	$\{2^n - 1, 2^n + 1, 2^n - 3, 2^n + 3\}$	2004
14	$\{2^n - 1, 2^n + 1, 2^{2n} - 2, 2^{2n+1} - 3\}$	2008
15	$\{2^n - 1, 2^n + 1, 2^n, 2^{2n} + 1\}$	2009
16	$\{2^n - 1, 2^n, 2^n + 1, 2^{2n+1} - 1\}$	2009
17	$\{2^n - 1, 2^n + 1, 2^{2n}, 2^{2n+1} - 1\}$	2009
18	$\{2^k, 2^n - 1, 2^n + 1, 2^{n+1} + 1\}$	2014
19	$\{2^k, 2^n - 1, 2^n + 1, 2^{n+1} - 1\}$	2014
20	$\{2^n - 1, 2^n, 2^n + 1, 2^n - 2^{(n+1)/2} + 1, 2^n + 2^{(n+1)/2} + 1\}$	2005
21	$\{2^n - 1, 2^n, 2^n + 1, 2^{n-1} - 1, 2^{n+1} - 1\}$	2007
22	$\{2^{n/2} - 1, 2^n, 2^{n/2} + 1, 2^n + 1, 2^{2n-1} - 1\}$	2009
23	$\{2^n - 1, 2^n, 2^n + 1, 2^n - 2^{(n+1)/2} + 1, 2^n + 2^{(n+1)/2} + 1, 2^{n\pm1} + 1\}$	2012
24	$\{2^n - 1, 2^{n+\beta}, 2^n + 1, 2^n - 2^{(n+1)/2} + 1, 2^n + 2^{(n+1)/2} + 1, 2^{n\pm1} + 1\}$	2012
25	$\{2^{n+\beta}, 2^n - 1, 2^n + 1, 2^n - k_1, 2^n + k_1, \dots, 2^n - k_f, 2^n + k_f\}$	2018

In the work [19], the sets of modules from Table 2 were investigated. As a result of the experiments, the set of modules $\{2^n - 1, 2^n, 2^n + 1\}$ turned out to be the most effective.

In cryptography, it is possible to use general-form moduli sets. By increasing the number of moduli, a higher degree of parallelism can be achieved. The literature often describes methods for generating RNS bases based on Pseudo-Mersenne numbers [20]. The paper [14] proposes a filtering method for constructing a very large RNS basis. However, these approaches do not account for the compactness condition of the RNS basis.

Definition 1. A set of moduli $\{p_1, p_2, \dots, p_n\}$, where $p_1 < p_2 < \cdots < p_n$, is compact if $p_n < 2p_1$.

Let's look at Examples 2 and 3.

Example 2. For the basis $\{2047, 2048, 2049\}$ the number $X = 3758423681$ in RNS it is presented as $X \xrightarrow{RNS} (673, 1665, 353)$.

Example 3. For the basis $\{3, 5, 626604229\}$ the number $X = 3758423681$ in RNS it is presented as $X \xrightarrow{RNS} (2, 1, 625402536)$.

As can be seen from Example 3, if the compactness condition is not met, the third residue of the number in RNS has the same bit length as the number in the positional numeral system, which negates the advantage of RNS. The total computational delay depends on the largest modulo in the system.

4. Generating Compact RNS Bases

To generate compact sets of moduli we can use the method of constructing prime numbers which is used in the standard STB 1176.2-99 (and the Russian standard GOST R 34.10-94 which has ceased to function), which is based on Diemitko's theorem [21].

Theorem 1: Let $n = qR + 1$, where q is a prime odd number, R is an even number, $R < 4(q + 1)$, i.e., $n < (2q + 1)^2$. If $a < n$ is found:

1) $a^{n-1} \equiv 1(mod\ n)$,
2) $a^{\frac{n-1}{q}} \not\equiv 1(mod\ n)$, then n is a prime number.

Thus, if we have a prime number q, then, by searching even numbers R, we construct numbers $n = qR + 1$ and test them for primality according to Diemitko's theorem until we obtain a prime number. By the obtained number we can construct another prime number.

Algorithm 2 allows us to obtain a larger prime number p, having length $|p| = t$, starting from a smaller prime number q, whose length is $|q| = \left\lceil \frac{t}{2} \right\rceil$.

Algorithm 2: Generating prime numbers using Diemitko's theorem (PrimeNumbers).

Input: t – the required dimensionality of the prime number, q – a prime number
Output: p
1. $R = \left\lceil \frac{2^{t-1}}{q} \right\rceil + \left\lceil \frac{2^{t-1}\xi}{q} \right\rceil$
2. **if** $R \not\equiv 0(mod\ 2)$ **then**
2.1 $R = R + 1$
3. $u = 0$
4. $n = (R + u)q + 1$
5. **if** $n > 2t$ **then**
5.1 Return to step 1
6. **if** $2^{(n-1)} \equiv 1(mod\ n)$ **and** $2^{(R+u)} \not\equiv 1(mod\ n)$ **then**
6.1 $p = n$
6.2 **break**
7. **else**
7.1 $u = u + 2$
7.2 Return to step 4

The process uses a random variable ξ uniformly distributed on the interval $(0,1)$. This value is generated in a linear congruent manner. For each step of the algorithm, a new value of ξ is calculated.

Some prime numbers generated by this method may not be defined as such, because at step 6 the verification of the condition of Diemitko's theorem is carried out only for the number $a = 2$ and not for all $a < p$. However, the probability that a randomly chosen number a fulfils the conditions of Diemitko's theorem for a prime number n is $\left(1 - \frac{1}{q}\right)$. Using the check exclusively for $a = 2$ turns out to be quite sufficient to exclude only a small number of prime numbers from consideration. The advantage of choosing $a = 2$ is due to the fact that the degree expansion of the number 2 in the binary representation system is performed very efficiently.

Here is an example of generating a prime number using Diemitko's theorem.

Example 4. For $q = 3 = 11_2$, let us generate a prime number of length $t = 4$.

Let's find R at $\xi = 0.5$:

$$R = \left\lceil \frac{8}{3} \right\rceil + \left\lceil \frac{8 \cdot 0.5}{3} \right\rceil = 3.$$

To satisfy parity, $R = R + 1 = 4$. Candidate prime numbers $p = 4 \cdot 3 + 1 = 13$.

Since, $2^{12}(mod\ 13) \equiv 1$ and $2^4(mod\ 13) \not\equiv 1$.

Hence, the sought prime number $p = 13 = 1011_2$.

Thus, using Algorithm 2, Algorithm 3 is developed, which allows to generate compact bases with moduli of the form $p_i = Rq_i + 1$.

Algorithm 3: Generation compact bases.

Input: $\{q_1, q_2, \dots, q_n\}, t$

Output: $b = \{p_1, p_2, \dots, p_k\}$

1. b append PrimeNumbers(q_1, t)
2. **for** $i = 2, i < n, i++$ **do**

2.1 p =PrimeNumbers(q_1, t)

2.2 **if** $p < 2p_1$ **then**

2.2.1 b append p

In the next section, we will consider the performance of the proposed algorithm.

5. Performance Evaluation

Modelling and computational experiments were conducted on a computer equipped with a 2.80 GHz Intel Core i7-7700HQ processor, 8 GB of 1196 MHz DDR4 RAM and a 512 GB SSD, running Windows 10 Home, using the high-level programming language C++.

The first stage of the experiment involved comparing the speed of generating compact bases based on the Diemitko's theorem against methods for generating very large RNS bases. For comparison, two approaches were selected: the Pseudo-Mersenne number construction method and the general filtering method. The performance results of the algorithms are presented in Table 3.

Table 3. Comparison of execution time for different bases generation approaches, ms.

Number of modulo	General filtering	Construction base of Pseudo-Mersenne	Generating compact bases
8	10324	5328	4561
12	25211	9523	7531
16	50164	15433	13467
20	79057	19544	15389
32	165897	28413	23953

The compact bases generation method, based on the Diemitko's theorem, exhibits superior performance across all tested cases, with execution times substantially lower than both the Pseudo-Mersenne construction and general filtering approaches.

The experimental results demonstrate that the compact basis generation method is on average 15.5% faster than the Pseudo-Mersenne-based construction method and 75.7% faster than the general filtering approach when varying the number of moduli from 8 to 32. The maximum performance advantage is observed for the 32-moduli basis, where the compact method outperforms the Pseudo-Mersenne approach by 15.7% and the general filtering method by 85.6%. The second stage involved constructing sets of moduli with a dynamic range size from 32 to 128 bits and comparing them with a special set $\{2^n - 1, 2^n, 2^n + 1\}$. The comparison sets are presented in Tables 4 and 5. Next, the sets of moduli were compared in performing modular operations (Algorithm 1). The results of modular operations execution time are presented in Tables 6-8. Based on the presented data, we can conclude that on average compact bases provide the following speed gains for modular operations: by 11.87% for addition, by 12.08% for subtraction, and by 12.43% for multiplication. Thus, the use of compact bases allows speeding up calculations by about 12% on average for all operations compared to the use of moduli of a special set. Especially noticeable speed increase is observed for 96 and 128 bits, which indicates the prospect of using the algorithm of compact bases generation when increasing the digit capacity of numbers.

Table 4. Bases generated by algorithm 3 for modular operations modeling.

Dynamic range size, bits	**Bases**
32	$\{1823, 1997, 1997\}$
64	$\{521, 599, 613, 617, 647, 761\}$
96	$\{4217, 4447, 4951, 5279, 5281, 5461, 5521, 6521\}$
128	$\{16633, 17317, 17579, 17747, 20287, 20981, 21067, 22079, 24179\}$

Table 5. Moduli sets $\{2^n - 1, 2^n, 2^n + 1\}$ *for modular operations modeling.*

Dynamic range size, bits	**Bases**
32	$\{2047, 2048, 2049\}$
64	$\{4194303, 4194304, 4194305\}$
96	$\{4294967295, 4294967296, 4294967297\}$
128	$\{8796093022207, 8796093022208, 8796093022209\}$

Table 6. Results of number addition modeling in RNS, μs.

Dynamic range size, bits	32	64	96	128
Bases $\{2^n - 1, 2^n, 2^n + 1\}$	143.384	167.211	195.984	223.263
Bases generated by the algorithm 3	142.184	140.301	162.101	193.652

Table 7. Results of number subtraction in RNS, μs.

Dynamic range size, bits	32	64	96	128
Bases $\{2^n - 1, 2^n, 2^n + 1\}$	143.641	168.023	196.112	221.287
Bases generated by the algorithm 3	143.1	140.871	161.539	189.975

Table 8. Results of multiplication of numbers in RNS, μs.

Dynamic range size, bits	32	64	96	128
Bases $\{2^n - 1, 2^n, 2^n + 1\}$	144.544	172.382	198.073	224.632
Bases generated by the algorithm 3	143.624	141.593	163.122	194.162

6. Conclusion

This paper presents a method for generating compact RNS bases based on Diemitko's theorem. Experimental results reveal that the compact basis generation method reduces computation time by an average of 15.5% compared to Pseudo-Mersenne-based construction and 75.7% compared to general filtering. These advantages become more pronounced with increasing system size, reaching performance improvements of 15.7% and 85.6% respectively for 32-moduli bases. The method's efficiency extends to modular arithmetic operations, where it provides approximately 12% faster execution for addition, subtraction, and multiplication compared to traditional $\{2^n - 1, 2^n, 2^n + 1\}$ moduli sets.

Thus, the proposed RNS basis generation approach demonstrates high computational speed while enhancing the efficiency of modular arithmetic operations. This is particularly crucial for cryptographic applications that require processing of large numbers. Future research will focus on optimizing non-modular RNS operations using the moduli generated by the proposed algorithm.

References

[1]. Sousa L. Nonconventional computer arithmetic circuits, systems and applications. IEEE Circuits Syst. Mag, 2021, vol. 21, no. 1, pp. 6-40.

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

[3]. Луценко В.В., Бабенко М.Г., Хамидов М.М. Высокоскоростной метод перевода чисел из системы остаточных классов в позиционную систему счисления. Труды ИСП РАН, том 36, вып. 4, 2024 г., стр. 117-132. DOI: 10.15514/ISPRAS-2024-36(4)-9. / Lutsenko V.V., Babenko M.G., Khamidov M.M. High speed method of conversion numbers from residue number system to positional notation. Proceedings of the Institute for System Programming of the RAS, 2024, vol. 36, issue 4, pp. 117-132 (in Russian). DOI: 10.15514/ISPRAS-2024-36(4)-9.

[4]. Луценко В.В., Бабенко М.Г., Черных А.Н., Лапина М.А. Оптимизация алгоритма деления чисел в системе остаточных классов на основе функции ядра Акушского. Труды ИСП РАН, том 35, вып. 5, стр. 157-168. DOI: 10.15514/ISPRAS-2022-35(5)-11. / Lutsenko V.V., Babenko M.G., Tchernykh A.N., Lapina M.A. Optimization of a number division algorithm in the residue number system based on the Akushsky core function. Proceedings of the Institute for System Programming of the RAS, 2023, vol. 35, issue 5, pp. 157-168 (in Russian). DOI: 10.15514/ISPRAS-2022-35(5)-11.

[5]. Shiriaev E. Kucherov N., Babenko M., Nazarov A. Fast operation of determining the sign of a number in rns using the akushsky core function. Computation, 2023, vol. 11, no. 7, pp. 124.

[6]. Cardarilli G. C., Nannarelli A., Re M. RNS applications in digital signal processing. Embedded Systems Design with Special Arithmetic and Number Systems, 2017, pp. 181-215.

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

[8]. Nakahara H., Sasao T. A High-speed Low-power Deep Neural Network on an FPGA based on the Nested RNS: Applied to an Object Detector. 2018 IEEE international symposium on circuits and systems (ISCAS). – IEEE, 2018, pp. 1-5.

[9]. Ananda Mohan P. V. RNS in Cryptography. Residue Number Systems: Theory and Applications. – Cham: Springer International Publishing, 2016, pp. 263-347.

[10]. Fournaris A. P., Papachristodoulou L., Sklavos N. Secure and efficient rns software implementation for elliptic curve cryptography. 2017 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW). – IEEE, 2017, pp. 86-93.

[11]. Fournaris A. P., Papachristodoulou L., Batina L., Sklavos N. Secure and efficient RNS approach for elliptic curve cryptography, 2016.

[12]. Zalekian A., Esmaeildoust M., Kaabi A. Efficient implementation of NTRU cryptography using residue number system. International Journal of Computer Applications, 2015, vol. 124, no. 7.

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

[14]. Bajard J. C., Fukushima K., Plantard T., Sipasseuth A. Generating very large RNS bases. IEEE Transactions on Emerging Topics in Computing, 2022, vol. 10, no. 3, pp. 1289-1301.

[15]. Lutsenko V., Zgonnikov M. Investigation of Neural Network Methods for Error Detection and Correction in the Residue Number System. International Workshop on Advanced Information Security Management and Applications. – Cham: Springer Nature Switzerland, 2024, pp. 194-206.
[16]. Omondi A. R., Premkumar A. B. Residue number systems: theory and implementation. – World Scientific, 2007, vol. 2.
[17]. Skavantzos A., Abdallah M., Stouraitis T. Large dynamic range RNS systems and their residue to binary converters. Journal of Circuits, Systems, and Computers, 2007, vol. 16, no. 02, pp. 267-286.
[18]. Molahosseini A. S., Teymouri F., Navi K. A new four-modulus RNS to binary converter. Proceedings of 2010 IEEE International Symposium on Circuits and Systems. – IEEE, 2010, pp. 4161-4164.
[19]. Lutsenko V.V., Kravtsov M.D., Gorlachev D.E., Mirny N.M. Research of special sets of moduli of the residue number system. Proceedings of the Institute for System Programming of the RAS, 2025, vol. 35, no. 5, pp. 157-168 (in Russian).
[20]. Kawamura S., Koike M., Sano F., Shimbo A. Cox-rower architecture for fast parallel montgomery multiplication. Advances in Cryptology—EUROCRYPT 2000: International Conference on the Theory and Application of Cryptographic Techniques Bruges, Belgium, May 14–18, 2000 Proceedings 19. Springer Berlin Heidelberg, 2000. pp. 523-538.
[21]. Diemitko N. Generating multiprecision integer with guaranted primality. Proc. of the SIFIP Int. Conf. on Comp. Sci., IFIP Security 88, Amsterdam, 19-21 May, 1988. pp. 1-8.

Информация об авторах / Information about authors

Владислав Вячеславович ЛУЦЕНКО – аспирант кафедры вычислительной математики и кибернетики факультета математики и компьютерных наук имени профессора Н.И. Червякова ФГАОУ ВПО «Северо-Кавказский федеральный университет». Сфера научных интересов: высокопроизводительные вычисления, система остаточных классов, умный город, нейронные сети, интернет вещей.

Vladislav Vyacheslavovich LUTSENKO – postgraduate student, Department of Computational Mathematics and Cybernetics, Faculty of Mathematics and Computer Science named after Professor N.I. Chervyakov, North Caucasus Federal University. Research interests: high-performance computing, residue number system, smart city, neural networks, Internet of Things.

Михаил Григорьевич БАБЕНКО – доктор физико-математических наук, заведующий кафедры вычислительной математики и кибернетики факультета математики и компьютерных наук имени профессора Н.И. Червякова ФГАОУ ВПО «Северо-Кавказский федеральный университет». Сфера научных интересов: облачные вычисления, высокопроизводительные вычисления, система остаточных классов, нейронные сети, криптография.

Mikhail Grigoryevich BABENKO – Dr. Sci. (Phys.-Math.), Head of the Department of Computational Mathematics and Cybernetics, Faculty of Mathematics and Computer Science named after Professor N.I. Chervyakov, North Caucasus Federal University. His research interests include cloud computing, high-performance computing, residue number systems, neural networks, cryptography.