Method of Multi-Bit Numbers Multiplication in Residue Number System for Asymmetric Cryptosystems

Mykhailo Kasianchuk [a], Ihor Yakymenko [a], Vasyl Yatskiv[a], Mikolaj Karpinski[b,c] and Solomiya Yatskiv[a]

[a] *West Ukrainian National University, 11 Lvivska str., Ternopil, 46009, Ukraine*
[b] *University of Bielsko-Biala, 2, Willova str., Bielsko-Biala, 43-309, Poland*
[c] *Ternopil Ivan Puluj National Technical University, 56, Ruska str, Ternopil, 46001, Ukraine*

Abstract

Nowadays the requirements for modern information security systems stability and speed are constantly growing. Therefore, the development of methods for parallel processing of multi-bit numbers in asymmetric cryptographic algorithms is an urgent task. Modern algorithms in most cases have strictly consistent structures based on the positional binary numeral system, which causes certain functional limitations. An important area is the usage of non-positional residue number system. It allows successfully parallelizing the processes of addition, multiplication and exponentiation of multi-bit numbers. These are basic operations in asymmetric systems for information flows protection in computer systems. However, there are some difficulties in recovering a decimal number from its residues due to the most time-consuming operation of finding a multiplicative inverse element by moduli. To eliminate this problem, it is advisable to use a modified perfect form of the residue number system.

In the paper methods for multiplying multi-bit numbers in the residue number system and its modified perfect form are proposed, which, in contrast to the existing ones, allow reducing the bit size of operands and executing arithmetic operations in parallel. Analytical expressions of time complexities depending on factors bit-size and number of moduli are constructed for developed methods. As a result, it was determined that the complexity significantly increases with increasing bit size and decreasing number of modules. For effective software implementation of the proposed methods, a block diagram is designed and the appropriate algorithmic implementations are developed, also decision on the programming environment is substantiated. Experimental studies of the time characteristics of multiplication at different ratios between moduli in the residue number system have been carried out. Graphical dependencies of time characteristics on bit size of input parameters are provided.

Keywords

Multiplication operation, residue number system, modulo system, asymmetric cryptography, bit size, time complexity

1. Introduction

At the current stage of societal development, the scope of large numbers usage and operations on them is not limited to specialized science intensive tasks.

IntelITSIS'2022: 3rd International Workshop on Intelligent Information Technologies and Systems of Information Security, March 23–25, 2022, Khmelnytskyi, Ukraine
EMAIL: kasyanchuk@ukr.net (M. Kasianchuk); iyakymenko@ukr.net (I. Yakymenko); jazkiv@ukr.net (V. Yatskiv); mpkarpinski@gmail.com (M. Karpinski); solyamiya@ukr.net (S. Yatskiv)
ORCID: 0000-0002-4469-8055 (M. Kasianchuk); 0000-0003-3446-1596 (I. Yakymenko); 0000-0001-9778-6625 (A. 3); 0000-0002-8846-332X (V. Yatskiv); 0000-0001-7470-7314 (S. Yatskiv)

In recent years, asymmetric cryptography systems [1] have become increasingly important. Their implementation requires calculations on multi-bit integer operands with a few thousands of decimal digits [2].

In most cases, users have to use computer systems with limited performance to execute the client part of crypto-protocols. This determines the importance of increasing the speed of operations on large numbers in their software, hardware, or software and hardware implementation [5,6].

With regard to asymmetric cryptography (RSA cryptosystems, Rabin, El Gamal, electronic digital signature algorithms, encryption on elliptical curves [5, 6]), the greatest attention should be paid to optimizing the performance of multiplication operations and its derivatives (exponentiation, in particular, exponent 2) [7], as they account for 45% and 24% (in the case of 2048-bit key) of the total complexity of cryptocurrency operations.

The most common among positional number systems for today is the binary system that has a strictly consistent structure. This limits its ability to process information in parallel.

Usage of non-positional number systems, one of which is the residue number system (RNS) [8], allows eliminating this drawback. It also has some known disadvantages, for example difficulties in division [9] and comparison [10] operations, but it can be successfully used in asymmetric cryptosystems to parallelize the processing of multi-bit numbers while adding, multiplying and exponentiating.

2. Related works

2.1. Theoretical foundations of RNS

The theoretical basis of RNS is algebra and number theory. Any integer N, written in positional, in particular, decimal number system, is represented as a set (b_1, b_2, … , b_k) in the RNS. b_i values are the smallest non-negative residues from the division of the number N by fixed numbers (or moduli) p_1, p_2, ... , p_k ($b_i=N \bmod p_i$), where k is the number of moduli.

The moduli must be natural and pairwise coprime numbers. In addition, the inequality $0 \le N \le P-1$, where $P=\prod_{i=1}^{k} p_i$ - a number that determines the condition of bitwise calculations overflow.

Arithmetic operations (addition, multiplication, exponentiation) are performed separately for each low-bit modulo. After that, obtained results are converted into a positional number system, mainly using the Chinese remainder theorem (CRT) [8].

The reverse conversion into a positional number system is usually based on the CRT:

$$N = \left(\sum_{i=1}^{k} b_i B_i \right) \bmod P, \quad (1)$$

where $B_i=M_i m_i$, $M_i = \frac{P}{p_i}$, $m_i = M_i^{-1} \bmod p_i$.

2.2. Application of RNS in computer systems

RNS usage in computer systems can significantly increase the speed of integer arithmetic operations implementation, which is very important for asymmetric cryptography. In particular, in [11] a method of applying floating-point intervals for non-modular calculations in RNS was proposed.

In [12] this method was improved and it was experimentally demonstrated that for random residues and a 128 modules set with 2048-bit dynamic range, the proposed implementation reduces the operating time by 39 times and memory consumption by 13 times compared to the implementation based on mixed transformations.

In [13] it was shown that the usage of RNS in the Montgomery method is an effective way to increase the speed of modular multiplication, but its time complexity still remains high for multi-bit

numbers processing. Significant acceleration can be achieved by moving to the processor level of arithmetic operations or cloud technologies usage [14].

In [5, 15] secure and effective approaches for RNS application in cryptography on elliptic curves are presented. They are especially effective as a response to attacks through the side channel leakage and during the malfunctions introduction in the computer system.

Paper [2] present effective algorithms for implementing RSA-cryptographic system based on RNS, experimental studies of which have shown that they have greater speed and resistance to brute force attacks compared to classical ones.

2.3. Selection of specialized RNS modules sets

One of the ways to increase the computers operating speed in RNS is the choice of specialized module sets, which significantly affects the execution time of both modular and non-modular operations. Therefore, RNS offers many module sets of different types and quantities for certain applications that significantly affect all parts of the hardware implementation, including direct converters, modular arithmetic channels, reverse converters.

In the vast majority of works module type 2^k, $2^k \pm 1$ is considered, which enables rational usage of bit grid registers [14, 15].

However, the search for inverse elements by module is characterized by considerable computational complexity and in number theory it is realized by a complete search of possible options, using the Euclidean algorithm or Euler's theorem [18, 19].

In [20] the modified perfect form (MPF) of RNS is proposed, in which $M_i \mod p_i = \pm 1$. This eliminates the operation of finding the inverse element and the calculations are performed according to the following formula:

$$N = \left(\sum_{i=1}^{n} \pm b_i M_i \right) \bmod P, \tag{2}$$

because $m_i = \pm 1$.

In addition, [20] presents the theoretical basis for the construction of a three-modulo MPF RNS. However, currently there are no experimental studies of time characteristics for performing arithmetic operations, in particular, multiplication in the RNS and its MPF. This is the purpose of this work.

3. Proposed model

3.1. Multiplication method in the residue number system

Let's consider the product $b=a \cdot c$ of two numbers a and c, written in the positional decimal number system, using RNS with a set of modulo p_i. Their product $P = \prod_{i=1}^{k} p_i$ must exceed the desired result. First it is needed to find the remainders from the division of multiplicands by each of the modulo: $a_i = a \mod p_i$; $c_i = c \mod p_i$. Then obtained remains are multiplied by the appropriate moduli: $b_i = a_i \cdot c_i \mod p_i$.

The desired product is obtained as a result of restoring the positional (in particular, decimal) representation of the number of its residues according to the formula (1).

To reduce the operands on which operations are performed when using the CRT, expression (1) should be written as follows:

$$b = a \cdot c = \left(\sum_{i=1}^{k} M_i \left((m_i \cdot b_i) \bmod p_i \right) \right) \bmod P. \tag{3}$$

3.2. Algorithmic implementation of the proposed multiplication method in RNS

The step-by-step implementation of this method can be presented in the following way.

1. Start: p_i , i=1…k, a, c.
2. Residues are being searched a_i= a mod p_i c_i = c mod p_i.
3. b_i =a_i∗ c_i mod p_i is calculated.
4. The value $P = \prod_{i=1}^{k} p_i$ is being searched.
5. The basic RNS parameters are searched M_i =P/p_i.
6. m_i = $(M_i)^{-1}$mod p_i is calculated.
7. Operations are performed using RNS $b = a \cdot c = \left(\sum_{i=1}^{k} M_i \left((m_i \cdot b_i) \bmod p_i \right) \right) \bmod P$.
8. End: b .

For the effective software implementation of the proposed multiplication method in RNS its block diagram is developed and presented in Figure 1.

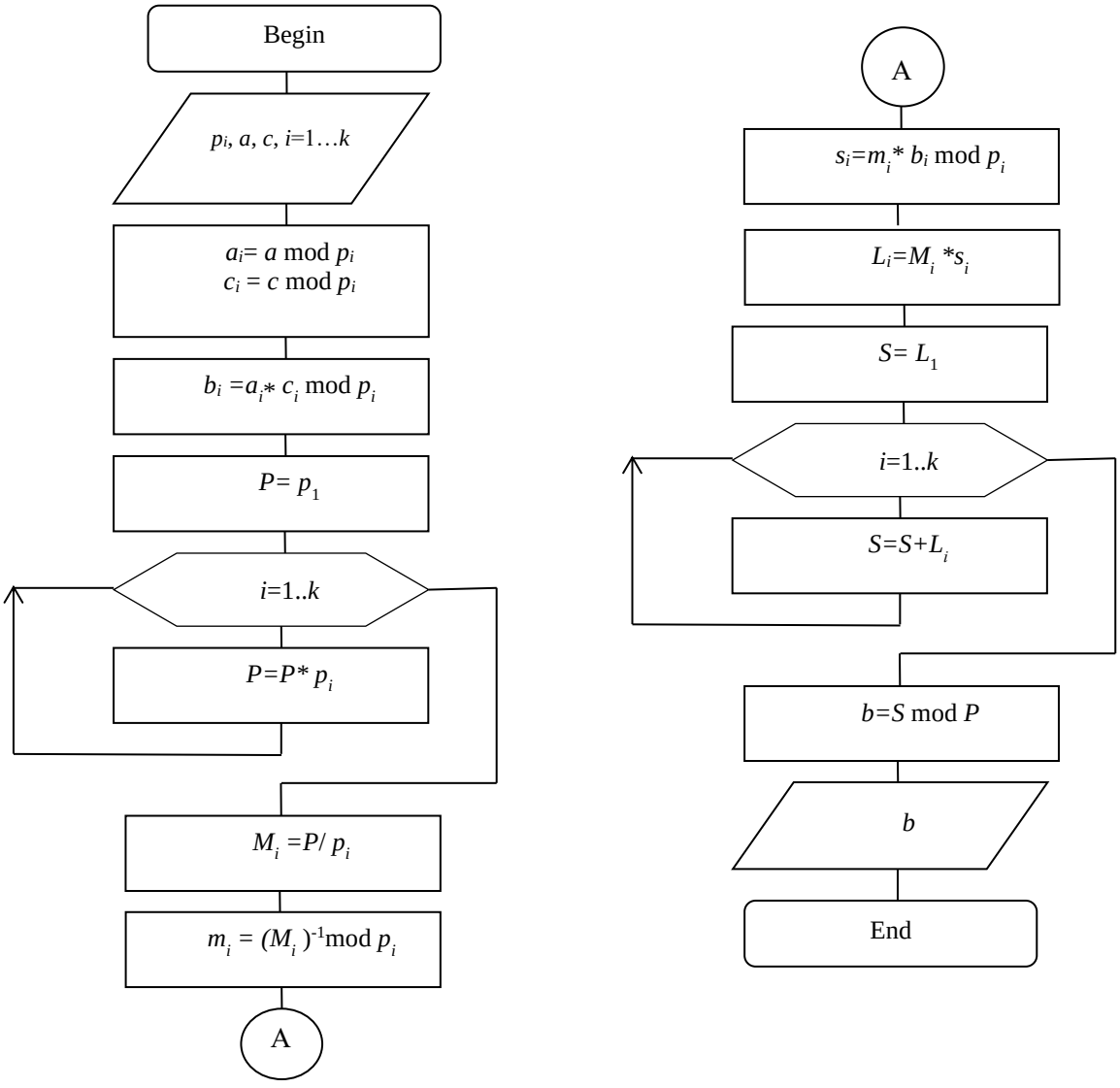


Figure 1: Block diagram of the algorithm for multiplying multi-bit numbers in RNS

3.3. Research of the multiplication method time complexity in RNS

The main operation of n-bit numbers multiplication by modulo in RNS is residues finding [19], finding the inverse value by modulo [16] (in MPF RNS it is not present), product of residues by modulo, restoring the decimal representation of the number from its residues.

Therefore, for determining the complexity of the proposed method the complexity of the above mentioned operations, which are presented in table 1, must be taken into account.

Table 1.

Time complexity of basic multiplication operations in RNS

№	Basic operations	Time complexity in RNS	Time complexity in MPF RNS
1.	$m_i = M_i^{-1} \bmod p_i$	$O\left(k \cdot n \cdot \log_2\left(\frac{n}{k}\right)\right)$	-
2.	$c(\bmod p_i) = c_i$, $a(\bmod p_i) = a_i$	$O\left(\log_2 n/2\right)$	$O\left(\log_2 n/2\right)$
3.	$c_i \cdot a_i(\bmod p_i)$, k-times $\left(\frac{n}{k}\right)$ - bit numbers	$O\left(\left(\frac{n^2}{k}\right)\right)$	$O\left(\frac{n^2}{k}\right)$
4.	$N = \left(\sum_{i=1}^{k} b_i M_i m_i\right) \bmod P$	$O\left(\frac{2n^2}{k}\right)$	$O\left(\frac{n^2}{k}\right)$

Given the tabular data, the multiplication time complexity in a conventional RNS will be $O1\left(\frac{3n^2}{k} + k \cdot n \cdot \log_2\left(\frac{n}{k}\right) + \log_2 n/2\right)$. Since there is no operation of multiplicative inverse element search by modulo in the MPF RNS, the time complexity will be reduced accordingly: $O2\left(\frac{3n^2}{k} + \log_2 n/2\right)$.

Figure 2 shows the surface, which demonstrates the dependence of the multiplication operation complexity $O1(n, k)$ on the bit-size and the number of modulo factors. It is determined that the complexity increases significantly with increasing n and decreasing k.

Examples of performing a multiplication operation in RNS

To demonstrate the proposed method, let's consider the RNS with three moduli (k=3): p_1=1579, p_2=1627, p_3=1705. Their product P=4380201265 is a 33-bit number. Let's find the product of two 16-bit numbers a=37831 and c=43529, which will be definitely less than P. Table 2 shows the values of intermediate values from formula (3), which are used to find the product b.

Table 2.

Intermediate values for finding the product

i	1	2	3
p_i	1579	1627	1705
M_i	2774035	2692195	2569033
$M_i \bmod p_i$	1311	1137	1303
m_i	1361	342	1582
a=37831, c=43529			
$a_i = a \bmod p_i$	1514	410	321
$c_i = c \bmod p_i$	896	1227	904
b_i	183	327	334

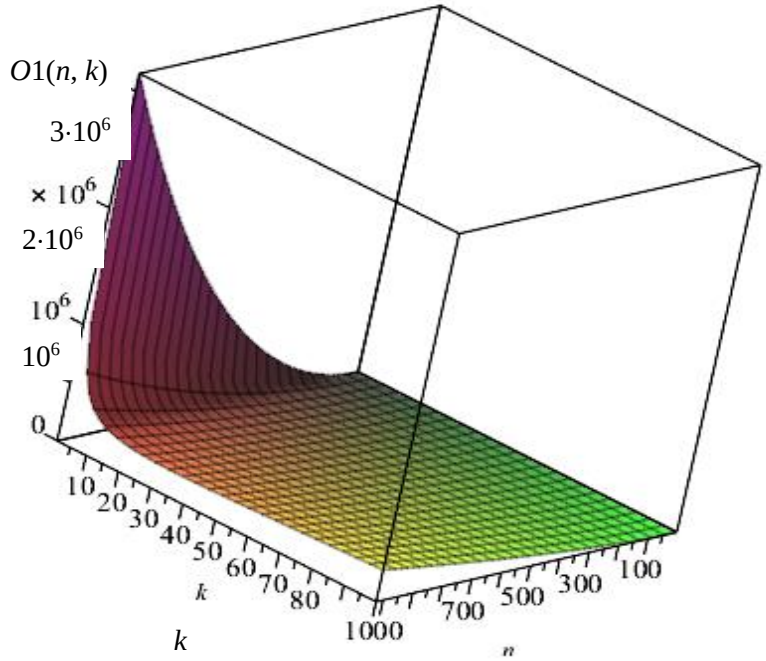


Figure 2: Dependence of the multiplication operation cc n xity on the factors bit-size and the number of module

3.4.

Then b = (2774035·((1361·183)mod1579) + 2692195·((342·327)mod1627) + +2569033·((1582·334)mod1705))mod4380201265=(2774035·1160+2692195·1198+ +2569033·1543)mod 4380201265=1646745599.

Therefore, instead of multiplying two 16-bit numbers, it is needed to multiply the 22-bit number by 11-bit number.

The calculation can be simplified taking into account the property of congruence that m_i, b_i mod p_i=(m_i, b_i - p_i) mod p_i. It is advisable for the case when the parameters m_i, b_i are greater than the half of the corresponding module.

For our example m_1=1361 mod 1579= =-218mod1579, m_3=1582mod1705=-123mod1705.

Hence b=(-2774035·((218·183)mod1579) + 2692195·((342·327)mod1627)- - 2569033·((123·334)mod1705))mod4380201265=(-2774035·419 + 2692195·1198 - -2569033·162)mod 4380201265=1646745599.

A significant reduction in computational complexity can be achieved by using a set of moduli that form MPF RNS (M_i mod p_i=±1), for example p_1=1025, p_2=2049, p_3=2051.

The values of corresponding parameters are shown in table 3.

It is shown that in this case the bulky modular operation of finding the inverse element is eliminated. Then b=(-4202499·74+2102275·1181-2100225·250)mod 4307561475=1646745599.

4. Results and Discussions

4.1. Rationale for choosing the programming environment

A high-level general-purpose programming language Python was chosen for the software implementation of multiplication operation in RNS and MPF RNS [20, 25-27]. It is focused on improving developer productivity and code readability.

Python kernel syntax is simple and minimal. At the same time, the standard library includes a large number of useful functions.

Python code is organized into functions and classes that can be combined into moduli (they, in turn, can be combined into packages).

An example of entering input parameters is shown in Figure 3.

Table 3.
Intermediate values for finding the product in MPF RNS

i	1	2	3
p_i	1025	2049	2051
M_i	4202499	2102275	2100225
$M_i \bmod p_i$	1024 mod 1025= = -1 mod 1025	1	1
m_i	1024 mod 1025= = -1 mod 1025	1	1
	a=37831, c=43529		
$a_i=a \bmod p_i$	931	949	913
$c_i=c \bmod p_i$	479	500	458
b_i	74	1181	1801 mod 2051= = -250 mod 2051

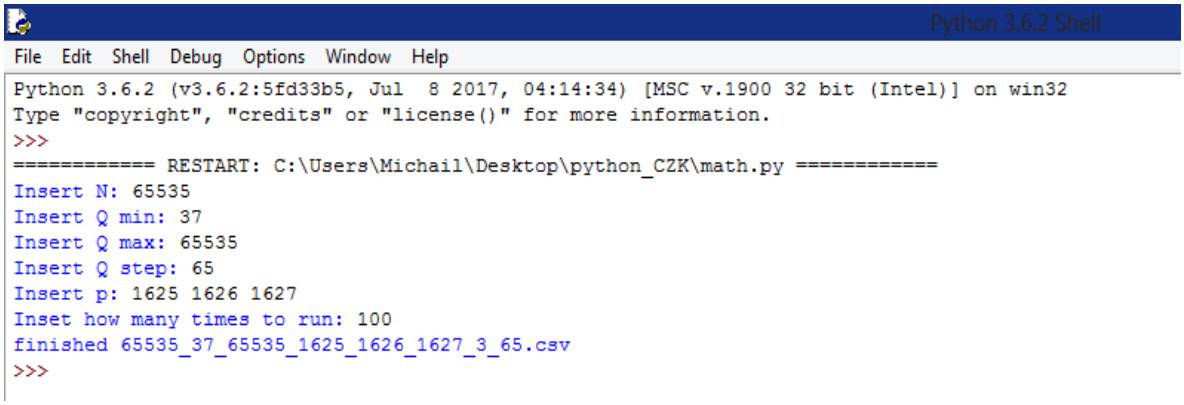


Figure 3: The main program window

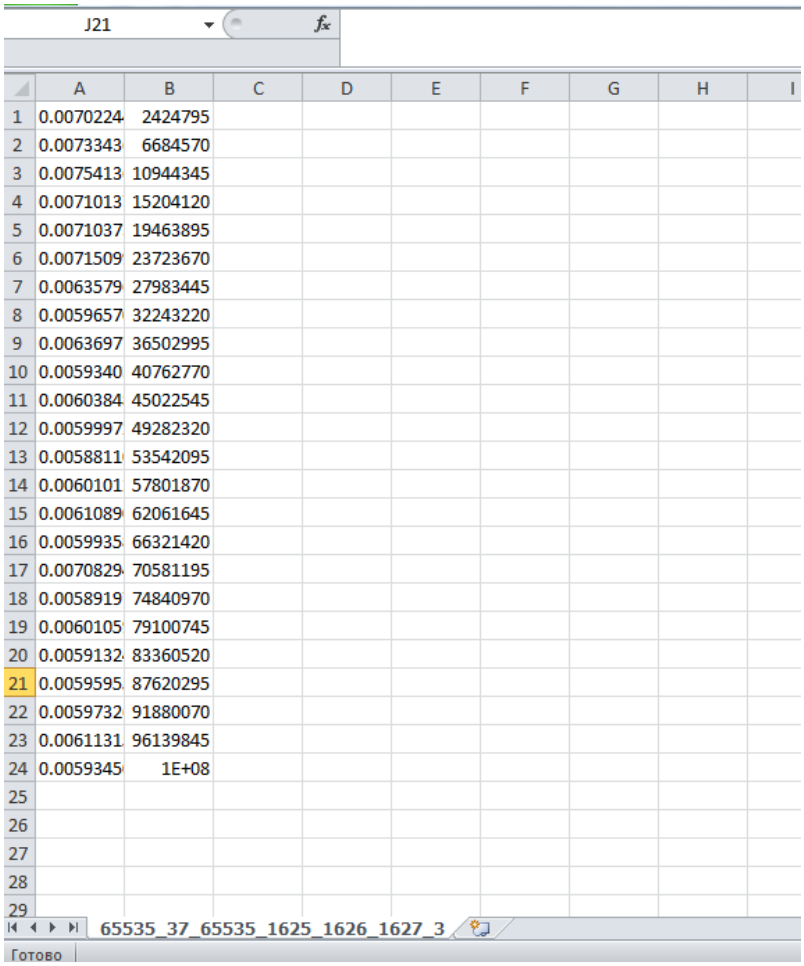
The results are stored in a file with *.csv extension. Its name is written in the last line of the main window and includes all input parameters.

An example of the generated file with the results of two numbers multiplication and the operation time is shown in Figure 4.

4.2. Experimental studies of the multiplication operation software implementation in a conventional RNS

Figure 5 presents the time characteristics of the multiplication operation $b=a*c$ in the three-module RNS with a fixed multiplier a=65536 with two different module systems (first case - the moduli have a little difference: p_1=1625=$\left\lfloor \sqrt[3]{65536^2} \right\rfloor$, p_2=1626, p_3=1627 - dotted lines, the second case - moduli have a big difference: p_1=163, p_2=1627, p_3=16381 - solid lines). The product of the moduli in both systems exceeds 2^{32}.

The second factor *c* varied from 67 to *a* with an interval of 1311. The last one determined the number of obtained calculations, which was equal to 50. The horizontal lines indicate the average time of calculations for each case.



J21

	A	B	C	D	E	F	G	H	I
1	0.0070224	2424795							
2	0.0073343	6684570							
3	0.0075413	10944345							
4	0.0071013	15204120							
5	0.0071037	19463895							
6	0.0071509	23723670							
7	0.0063579	27983445							
8	0.0059657	32243220							
9	0.0063697	36502995							
10	0.0059340	40762770							
11	0.0060384	45022545							
12	0.0059997	49282320							
13	0.0058811	53542095							
14	0.0060101	57801870							
15	0.0061089	62061645							
16	0.0059935	66321420							
17	0.0070829	70581195							
18	0.0058919	74840970							
19	0.0060105	79100745							
20	0.0059132	83360520							
21	0.0059595	87620295							
22	0.0059732	91880070							
23	0.0061131	96139845							
24	0.0059345	1E+08							
25									
26									
27									
28									
29									

65535_37_65535_1625_1626_1627_3

Готово

Figure 4: Example of the received file

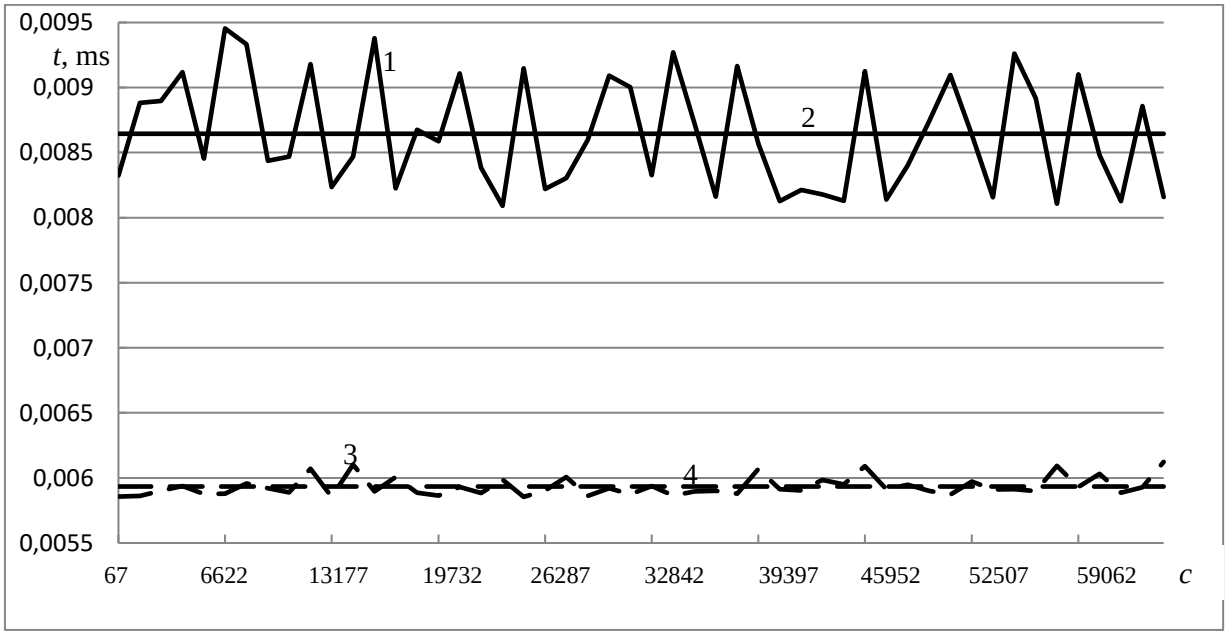


Figure 5: Time characteristics of the multiplication operation in the three-module RNS

As shown in Figure 5, chart 1 is oscillating. The average execution time of the multiplication operation (line 2) is 0.008645 ms. In the second case, the multiplication time (Figure 3) does not fluctuate significantly.

The average time (line 4) is 0.005934 ms, which is 1.46 times less than in the previous case. Therefore, in order to increase the speed in the RNS, pairwise coprime modulo must be chosen in such a way that they differ as little as possible from each other.

4.3. Experimental studies of the multiplication operation software implementation in the MPF RNS

For the MPF RNS research, system of moduli with a significant difference between them (p_1=651, p_2=691, p_3=11246) was chosen by the formula obtained in [25-27]:

$$p_3 = p_1 + \frac{p_1^2 \pm 1}{p_2 - p_1}. \quad (4)$$

During a three-moduli MPF RNS construction according to formula (4), the system of the same bit-size moduli can not be selected. The smallest difference between the moduli will be following:

$$p_{2,3} = 2p_1 \pm 1. \quad (5)$$

Based on this, the following moduli were selected: p_1=1025, p_2=2049, p_3=2051. Again, the product of the moduli in both cases exceeds 2^{32}.

The input parameters were the same as for conventional RNS. The calculations were performed according to the expression for MPF RNS:

$$b = \left(-b_1M_1 + b_2M_2 + b_3M_3\right) \bmod P, \quad (6)$$

The obtained results are presented in Figure 6. The solid line shows the multiplication time (curve 1) and the average time (line 2) for 50 p values when p_1=651, p_2=691, p_3=11246, dotted line (graphs 3, 4) shown results respectively for p_1=1025, p_2=2049, p_3=2051.

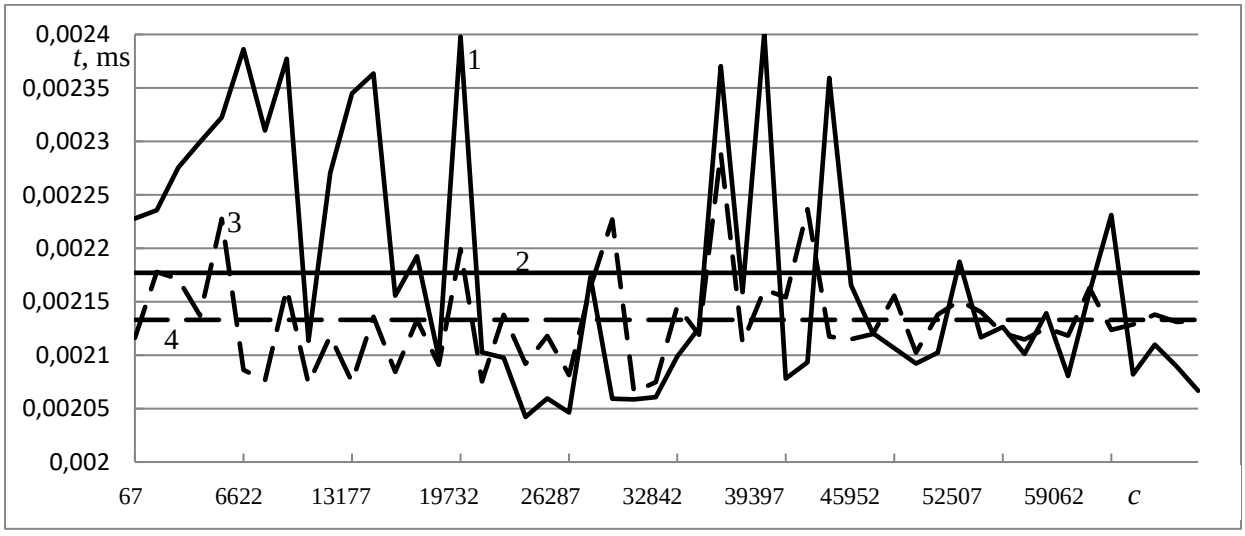


Figure 6: Time characteristics of multiplication in the three-moduli MPF RNS

It can be seen that in both cases at small c values, the amplitude of oscillations is large, with increasing c it decreases except for a small segment in the second half of the range of value c changes.

The average time for the modulo system p_1=651, p_2=691, p_3=11246 is 0,002177 мс (line 2), and for p_1=1025, p_2=2049, p_3=2051 - 0,002133 ms (line 4), which is 1.02 times less than in the previous case.

A comparison of Figures 5 and 6 shows a significant increase in performance due to the MPF RNS usage.

4.4. Comparative analysis of multiplication results in conventional RNS and MPF RNS

Further studies were performed for numbers whose bit-size n varied from 16 to 24 bits. Four cases of the modulo system construction were considered:

1) RNS moduli significantly different from each other;

2) moduli are three consecutive numbers, the first and third of which are odd: $p_1 \approx \left[\sqrt[3]{2^{2n}}\right]$, $p_2 = p_1+1$, $p_3 = p_1+2$;

3) moduli are calculated by the following formulas: $p_2 = p_1+1$, $p_3 = p_1(p_1+1)-1$;

4) moduli are calculated by the following expressions: $p_2 = 2p_1-1$, $p_3 = 2p_1+1$.

In all cases the product of the modulo is the smallest, but greater than 2^{2n}. In the third and fourth cases, the modulo systems form MPF RNS. The first factor in the product $b=a*c$ was fixed: $a=2^n-1$, which corresponds to the maximum number of the specified bit-size. The second factor c changed from the initial value $c = 2^n - \left[\frac{2^n}{1000}\right]+1$ with the interval of $\left[\frac{2^n}{1000}\right]$. Therefore, 1000 different values of the number c and, accordingly, the execution time of 1000 multiplication operations $b=a*c$ with a fixed value a and a variable c.

Further, for each bit-size the average operation execution time was determined by formula (3). In addition, for cases 3 and 4, the average time $t_{av\ 1}$ for executing operation of two numbers multiplication by formula (6) was determined.

To eliminate accidental effects, all calculations were repeated 100 times. The corresponding sets of modulo, as well as the average calculation time for numbers with different bit-size are presented in Table 4. Figure 7 shows the graphs of the average time of the multiplication by formula (3) dependence on the bit-size of numbers n that are used according to the table 4 (the graph number corresponds to the case number in Table 4.

Figure 7 shows that the most time is spent on a conventional RNS in the case when the moduli significantly different from each other.

Moreover, the graph growth is almost linear with increasing bit-size. Graphs 2 and 4 are almost linear at small bit-size, more intensive growth of graphs is observed at n=19 and n=21 respectively. And the third graph is close to linear over the entire considered range.

Analysis of figure 7 shows that the usage of moduli that either form the MPF RNS, or differ little from each other, allows increasing the speed of the computing system.

Figure 8 presents graphs that show dependence of the average multiplication time on the bit-size n for the third (curve 1) and fourth (curve 2) cases of table 3, the moduli in which form MPF RNS, with the same input parameters using formula (6).

Analysis of Figures 7 and 8 shows that the average computation time in MPF RNS decreases by approximately 2.5-3 times compared to the usual integer RNS form.

Conclusion

The paper proposes methods for multiplying multi-bit numbers in ordinary integer-valued RNS and MPF RNS, which, unlike the existing ones, allow reducing the operands bit-size and parallel the arithmetic operations execution. Analytical expressions for the developed methods time complexity depending on factors bit-size and number of moduli are constructed. As a result, it was determined that the complexity increases significantly with increasing bit-size and decreasing number of moduli.

For effective software implementation of the proposed methods, a block diagram is designed and the appropriate algorithmic implementations are developed. Experimental studies of the time

characteristics of multiplication at different ratios between moduli in RNS have been carried out. Graphical dependences of time characteristics on bit size of input parameters are provided. As a result of numerical experiments, it is determined that the average execution time of the multiplication operation is 1.46 times bigger when the moduli are significantly different than when they are almost the same. The speed of the algorithm for multi-bit numbers multiplication in MPF RNS is investigated. It is shown that the average time of the multiplication operation does not depend on the ratio between moduli.

A comparative analysis of time results in ordinary RNS and MPF RNS for various sets of moduli and numbers with different bit-size is conducted. It was found that using MPF RNS instead of the ordinary integer-valued RNS allows reducing multiplication operation time by approximately 3 times.

Table 4.
Sets of moduli and average computation time for numbers with different bit-size

n		16	17	18	19	20	21	22	23	24
Case 1	p_1	163	235	341	501	737	1093	1627	2429	3641
	p_2	1627	2587	4097	6503	10323	16387	26009	41287	65539
	p_3	16381	28413	49165	84541	144523	245807	416147	701881	1179703
t_{av}, μs		8,154	8,362	8.526	9,819	9,280	9,571	9,649	9,705	9,911
Case 2	p_1	1625	2581	4095	6501	10321	16385	26007	41285	65537
	p_2	1626	2582	4096	6502	10322	16386	26008	41286	65538
	p_3	1627	2583	4097	6503	10323	16387	26009	41287	65539
t_{av}, μs		5,891	5,95	5,962	5,966	6,199	5,335	6,682	7,185	7,304
Case 3	p_1	256	362	512	724	1024	1448	2048	2896	4096
	p_2	257	363	513	725	1025	1449	2049	2897	4097
	p_3	65791	131405	262655	524899	1049599	2098151	4196351	8389711	16781311
t_{av}, μs		5,461	5,680	5,618	5,689	5,650	5,684	5,673	5,732	5,693
$t_{av\,1}$, μs		2,152	2,215	2,255	2,285	2,292	2,287	2,293	2,297	2,310
Case 4	p_1	1025	1626	2581	4097	6502	10322	16385	26008	41286
	p_2	2049	3251	5161	8193	13003	20643	32769	52015	82571
	p_3	2051	3253	5163	8195	13005	20645	32771	52017	82573
t_{av}, μs		5,551	5,610	5,632	5,646	5,665	5,744	5,956	5,959	6,079
$t_{av\,1}$, μs		2,158	2,227	2,244	2,261	2,271	2,279	2,319	2,329	2,345

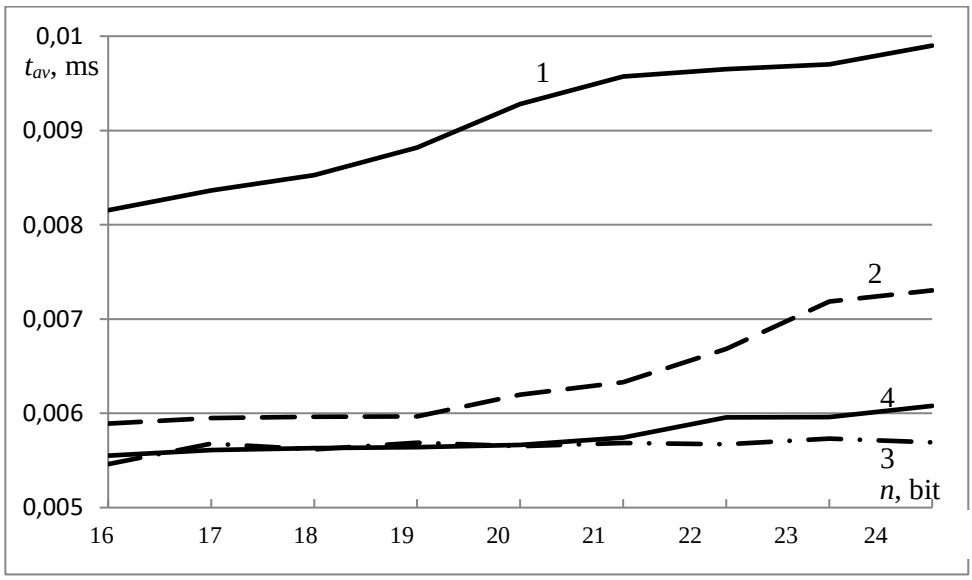


Figure 7: Graphs of the average time dependency of multiplication operation by formula (4) on the number bit-size

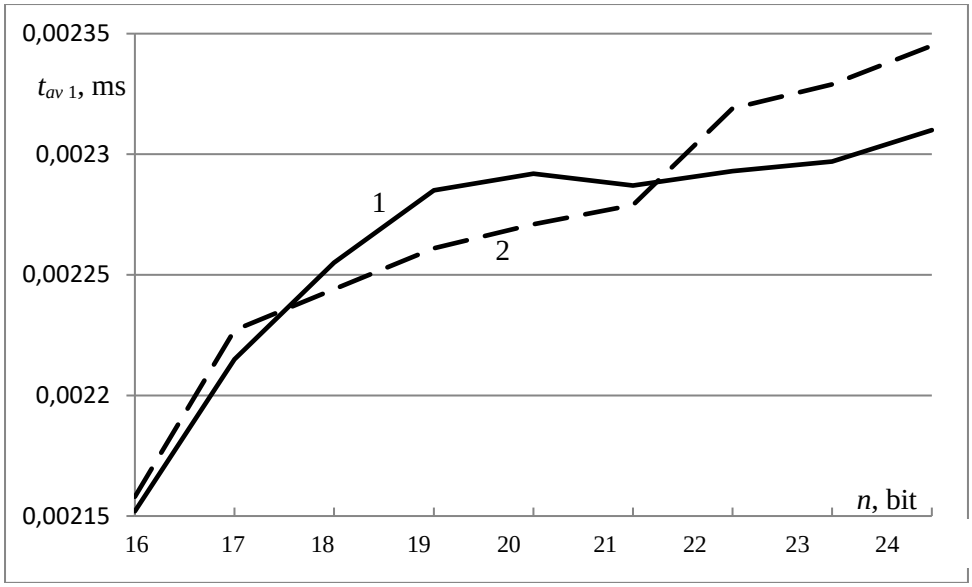


Figure 8: Graphs of the dependence of the average multiplication time on the bit-size while using the formula (7)

References

[1]. J.Bajard, J. Eynard, N. Merkiche, Multi-fault attack detection for RNS cryptographic architecture. Computer Arithmetic (ARITH 2016): Proceedings of the 23nd IEEE Symposium, Silicon Valley, CA, USA, 2016, pp. 16–23. https:// doi.org/10.1109/ARITH.2016.16.

[2]. I.R. Fadulilahi, E.K. Bankas, J.B.A.K. Ansuura, Efficient Algorithm for RNS Implementation of RSA. International Journal of Computer Applications, Vol. 127 (5), 2015, pp. 14-19.

[3]. Oksana Pomorova, Oleg Savenko, Sergii Lysenko, Andrii Kryshchuk. Multi-Agent Based Approach for Botnet Detection in a Corporate Area Network Using Fuzzy Logic. Communications in Computer and Information Science. 2013. Vol. 370. PP.243-254.

[4]. Bohdan Savenko, Sergii Lysenko, Kira Bobrovnikova, Oleg Savenko, George Markowsky. Detection DNS Tunneling Botnets. Proceedings of the 2021 IEEE 11th International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications, Cracow, Poland, September 22-25, 2021.

[5]. A.P. Fournaris, L. Papachristodoulou, L. Batina, N. Sklavos, Secure and Efficient RNS Approach for Elliptic Curve Cryptography. Trustworthy Manufacturing and Utilization of Secure Devices (TRUDEVICE 2016): Proceedings of the 6th Conference, Barcelona, 2016, pp. 121-126.

[6]. S. Asif, M.S. Hossain, Y. Kong, W. Abdul, A Fully RNS based ECC Processor. Integration, 61, 2018, pp.138–149. https://doi.org/10.1016/j.vlsi.2017.11.010.

[7]. V. Adki, S. Hatkar, A Survey on Cryptography Techniques. International Journal of Advanced Research in Computer Science and Software Engineering, Vol. 6 (6), 2016, pp. 469-475. https://doi.org/10.15587/2706-5448.2020.202099.

[8]. P.V. Ananda Mohan, Residue Number Systems: Theory and Applications. Birkhäuser, 2016, 351 p. https://doi.org/10.1007/978-3-319-41385-3.

[9]. K. Phalakarn and A. Surarerks, Alternative Redundant Residue Number System Construction with Redundant Residue Representations, 2018 3rd International Conference on Computer and Communication Systems (ICCCS), 2018, pp. 457-461, doi: 10.1109/CCOMS.2018.8463305.

[10].B. Raghavaiah and Omprakash. Implementation of Hamming coding in Residue Number System, 2018 International Conference on Current Trends towards Converging Technologies (ICCTCT), 2018, pp. 1-5, doi: 10.1109/ICCTCT.2018.8551122.

[11].W. K. Jenkins. Contributions of Graham Jullien and William Miller to Residue Number System Arithmetic Technology. 2018 IEEE 61st International Midwest Symposium on Circuits and Systems (MWSCAS), 2018, pp. 157-160, doi: 10.1109/MWSCAS.2018.8623919.

[12].K. Givaki et al. Using Residue Number Systems to Accelerate Deterministic Bit-stream Multiplication. 2019 IEEE 30th International Conference on Application-specific Systems, Architectures and Processors (ASAP), 2019, pp. 40-40, doi: 10.1109/ASAP.2019.00-33.
[13].J.-C. Bajard, J. Eynard, N. Merkiche, Montgomery Reduction within the Context of Residue Number System. Arithmetic Journal of Cryptographic Engineering, № 2, 2017, pp. 121-132. https://doi.org/10.1007/s13389-017-0154-9
[14].O.P. Markovskyi, N. Bardis, N. Doukas, S. Kirilenko, Secure Modular Exponentiation in Cloud Systems. Information Technology, Computational and Experimental Physics (CITCEP 2015): Proceedings of the Congress, Krakow, Poland, 2015, pp. 266-269.
[15].A.P. Fournaris, L. Papachristodoulou, L. Batina, N. Sklavos, Residue number system as a side channel and fault injection attack coun-termeasure in elliptic curve cryptography. Design and Technology of Integrated Systems in Nanoscale Era (DTIS): Proceedings of the 2016 International Conference, 2016, pp. 1–4. https://doi.org/10.1109/DTIS.2016.7483807.
[16].Z. Torabi, G. Jaberipur, A. Belghadr. Fast division in the residue number system {2n + 1, 2n, 2n - 1} based on shortcut mixed radix conversion. Comput. Electr, Eng., 83, 2020, 106571. https://doi.org/10.1016/j.compeleceng.2020.106571.
[17].S. Kumar, C. Chang, T.F. Tay. New Algorithm for Signed Integer Comparison in {2n+k , 2n − 1, 2n + 1, 2n±1 − 1} and Its Efficient Hardware Implementation. IEEE Trans. Circuits Syst. I Regul. Pap., 64, 2017, pp. 1481–1493. https://doi.org/10.1109/TCSI.2016.2561718.
[18].T. Rajba, A. Klos-Witkowska, S. Ivasiev, I. Yakymenko, M. Kasianchuk, Research of Time Characteristics of Search Methods of Inverse Element by the Module. Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications (IDAACS–2017): Proceedings of the 2017 IEEE 9th International Conference, Bucharest, Romania, V.1, September, 2017, pp.82–85. https://doi.org/10.1109/IDAACS.2017.8095054.
[19].Hu. Zhengbing, I. Dychka, M. Onai, A. Bartkoviak, The Analysis and Investigation of Multiplicative Inverse Searching Methods in the Ring of Integers Modulo M. International Journal of Intelligent Systems and Applications (IJISA), Vol. 8, №11, 2016, pp. 9-18. https://doi.org/10.5815/ijisa.2016.11.02.
[20].Ya.M.Nykolaychuk, M.M.Kasianchuk, I.Z.Yakymenko, Theoretical Foundations of the Modified Perfect Form of Residue Number System. Cybernetics and Systems Analysis, Vol. 52, №2, 2016, pp. 219-223. https://doi.org/10.1007/s10559-016-9817-2.
[21].S. Ivasiev, I. Yakymenko, M. Kasianchuk, R. Shevchuk, M. Karpinski, O. Gomotiuk, Effective algorithms for finding the remainder of multi-digit numbers. Advanced Computer Information Technology (ACIT–2019): Proceedings of the International Conference. Ceske Budejovice (Czech Republic), 2019, pp. 175-178. https://doi.org/10.1109/ACITT.2019.8779899.
[22].J. Stewart. Python for Scientists. Cambridge: Cambridge University Press, 2014, 230 p. https://doi.org/10.1017/CBO9781107447875.
[23]. A. Kumar and S. P. Panda, A Survey: How Python Pitches in IT-World, 2019 International Conference on Machine Learning, Big Data, Cloud and Parallel Computing (COMITCon), 2019, pp. 248-251, doi: 10.1109/COMITCon.2019.8862251.
[24].Alejandro Garces, Convex Programming in Python, in Mathematical Programming for Power Systems Operation: From Theory to Applications in Python , IEEE, 2022, pp.61-83, doi: 10.1002/9781119747291.ch4.
[25].L. Yu, "Empirical Study of Python Call Graph, 2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE), 2019, pp. 1274-1276, doi: 10.1109/ASE.2019.00160.
[26].H. Ren, L. Yang, L. Jiang, Y. Bai, W. Lu and J. Chang, A Computational-thinking-oriented Progressive Teaching Mode for Python Course, 2021 IEEE 3rd International Conference on Computer Science and Educational Informatization (CSEI), 2021, pp. 81-84, doi: 10.1109/CSEI51395.2021.9477642.
[27].A. Safari and A. A. Ghavifekr, International Stock Index Prediction Using Artificial Neural Network (ANN) and Python Programming, 2021 7th International Conference on Control, Instrumentation and Automation (ICCIA), 2021, pp. 1-7, doi: 10.1109/ICCIA52082.2021.9403580.