The Method of Joint Execution of the Basic Operations of the Rabin Cryptosystem

Mykhailo Kasianchuk [a], Ihor Yakymenko [a], Solomiya Yatskiv[a], Oksana Gomotiuk [a] and Lesia Bilovus [a]

[a] *West Ukrainian National University, 11 Lvivska str., Ternopil, 46009, Ukraine*

Abstract

The process of performing arithmetic operations of addition, multiplication, and exponentiation, where the operands are multi-bit numbers, is extremely common in applied problems of number theory, computational and discrete mathematics. This applies to methods of pattern recognition, digital signal processing, tamper-proof coding, statistical data processing, etc. However, their most important application is in asymmetric cryptography. In this field, for example, in the Rabin cryptosystem, it is often necessary to simultaneously perform the Euclidean algorithm and multiply two multi-bit numbers.

The work presents new methods of finding the greatest common divisor based on the formation of appropriate matrices, which allows parallelization of calculations. Their advantages over the existing ones are indicated, in particular, the possibility of increasing speed. It was established that these methods, in addition to finding the greatest common divisor, make it possible to factorize it simultaneously.

In the C+ high-level programming language, a program implementation of the simultaneous execution of the Euclidean algorithm and the multiplication of two multi-bit numbers by the classical Euclidean algorithm and the proposed methods was carried out. Corresponding experimental studies of the time characteristics of this software implementation have been conducted. The proposed method involves the use of intermediate results of the Euclidean algorithm and access to the table of squares stored in the computer's memory. The Rabin encryption scheme based on the proposed method of multiplying multi-bit numbers is given.

Numbers of different digits were used for the experiment. It is shown that in the vast majority of cases considered, the proposed method is characterized by a higher speed of processing multi-bit numbers. The average time of operations is reduced by approximately 1.43 times. All calculations were repeated 5000 times to eliminate random influences on the time characteristics of the computer. The proposed method can be effectively used when simultaneous execution of the Euclidean algorithm and multiplication of two multi-bit numbers is required.

Keywords 1

Multiplication operation, Euclidean algorithm, Rabin cryptosystem, prime numbers, time characteristics, number digits.

1. Introduction

IntelITSIS'2023: 4nd International Workshop on Intelligent Information Technologies and Systems of Information Security, March 22–24, 2023, Khmelnytskyi, Ukraine

EMAIL: kasyanchuk@ukr.net (M. Kasianchuk); iyakymenko@ukr.net (I. Yakymenko); solyamiya@ukr.net (S. Yatskiv); oksana_homotuk@ukr.net (O. Gomotiuk); lesya_bilovus@i.ua (L. Bilovus)

ORCID: 0000-0002-4469-8055 (M. Kasianchuk); 0000-0003-3446-1596 (I. Yakymenko); 0000-0001-7470-7314 S. Yatskiv); 0000-0002-2856-8541 (O. Gomotiuk); 0000-0003-4882-4511 (L. Bilovus)

Currently, the performance of arithmetic operations on multi-bit numbers is very widely used in various fields of science and technology, in particular, when solving problems of computational, applied and discrete mathematics [1-2]. Moreover, each of the corresponding algorithms has its own area of effective use depending on the bit rate, calculation model, programming language, hardware or software implementation [3]. Multi-bit number processing processes are extremely common in digital signal and image processing [4-6], methods of tamper-proof coding [7-8]. But this especially applies to the problems of cryptography [9-11], where the most common operations are multiplication, exponentiation [12], finding the greatest common divisor (GCD), use of the Chinese remainder theorem (CRT) [13-14], etc. The combination of the first two with the rest of the operations requires a strictly sequential implementation, which significantly reduces the speed of computer systems. An example can be the Rabin cryptosystem, in which the Euclidean algorithm is used to apply the CRT, and the same numbers must be multiplied to form the public key. Therefore, the question of the possibility of parallelization of similar operations [15], in particular, the use of the residual number system (RNS) [16, 17].

2. Related works

2.1. Rabin cryptosystem

To generate keys in the Rabin cryptosystem, two large prime numbers p and q are chosen, which act as a secret key. Their product $m=p\cdot q$ is a public key. The encryption of the plaintext block V, which must be maximal but less than m, occurs according to the formula $Z=V^2 \text{mod } m$.

Decryption is much more complicated and takes place in several stages. First, the residuals $z_1=Z \text{ mod } p$ and $z_2=Z \text{ mod } q$. are sought. Next, you need to determine the square roots modulo: $\sqrt{z_1} \text{ mod } p = z_{11}, z_{12} = -z_{11} = p - z_{11}$; $\sqrt{z_2} \text{ mod } q = z_{21}, z_{22} = -z_{21} = q - z_{21}$. After that, four pairs of residues are formed from the values of z_{ij}, where i, j=1, 2: (z_{11}, z_{21}), (z_{11}, z_{22}), (z_{12}, z_{21}), (z_{12}, z_{22}).The last stage of deciphering is a four-fold application of CRT. However, before that, it is necessary to find the values of parameters s i t in the fornula $s\cdot p+t\cdot q=1$ using the Euclidean algorithm and its consequence. This gives four variants of the plaintext block, one of which is correct:

1) $V_1=s\cdot p\cdot z_{21}+t\cdot q\cdot z_{11}$;
2) $V_2=s\cdot p\cdot z_{22}+t\cdot q\cdot z_{11}$;
3) $V_3=s\cdot p\cdot z_{21}+t\cdot q\cdot z_{12}$;
4) $V_4=s\cdot p\cdot z_{22}+t\cdot q\cdot z_{12}$.

Figure 1 shows the Rabin encryption scheme.

So, the complete cycle of Rabin cryptosystem involves the execution of Euclidean algorithm for two numbers (private keys) and their multiplication.

2.2. Greatest common divisor search methods

The mathematical notation of Euclidean algorithm looks like this: for any $X>Y=r_0$, where X and Y are integers, the system of equations is fulfilled.

$$\begin{aligned}
&X = r_0 \cdot q_1 + r_1,\ q_1 = Y,\ 0 \le r_1 < r_0; \\
&r_0 = r_1 \cdot q_2 + r_2,\ 0 \le r_2 < r_1; \\
&\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\
&r_{n-2} = r_{n-1} \cdot q_n + r_n,\ 0 \le r_n < r_{n-1}; \\
&r_{n-1} = r_n \cdot q_{n+1} + 0.
\end{aligned}$$

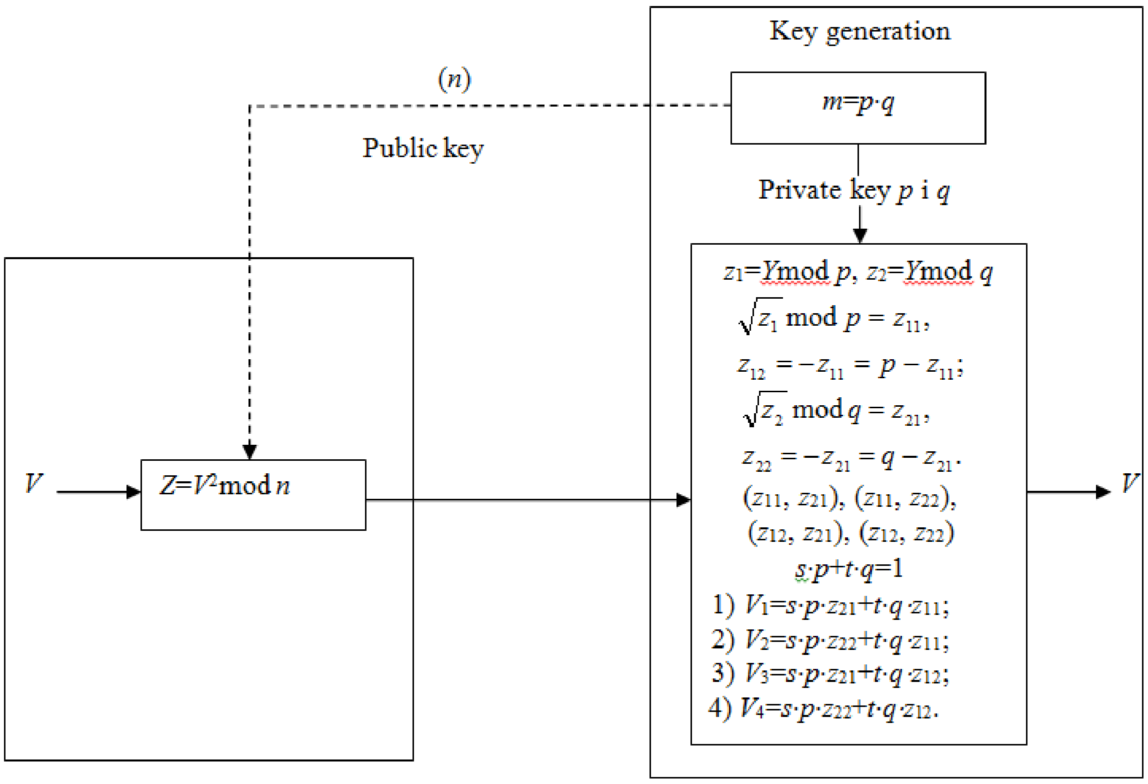
Key generation
$m=p\cdot q$
(n)
Public key
Private key p i q
$z_1=Y\bmod p$, $z_2=Y\bmod q$
$\sqrt{z_1} \bmod p = z_{11}$,
$z_{12}=-z_{11}=p-z_{11}$;
$\sqrt{z_2} \bmod q = z_{21}$,
$z_{22}=-z_{21}=q-z_{21}$.
(z_{11}, z_{21}), (z_{11}, z_{22}),
(z_{12}, z_{21}), (z_{12}, z_{22})
$s\cdot p+t\cdot q=1$
1) $V_1=s\cdot p\cdot z_{21}+t\cdot q\cdot z_{11}$;
2) $V_2=s\cdot p\cdot z_{22}+t\cdot q\cdot z_{11}$;
3) $V_3=s\cdot p\cdot z_{21}+t\cdot q\cdot z_{12}$;
4) $V_4=s\cdot p\cdot z_{22}+t\cdot q\cdot z_{12}$.
V
$Z=V^2 \bmod n$
V

Figure 1: Rabin encryption scheme

GCD (X, Y), calculated using the Euclidean algorithm, will be equal to r_n, that is, the last non-zero term of the sequence r_i. When the numbers X and Y are mutually prime, which is important for asymmetric cryptography, then r_n=1.

In addition, in [18], entitled Optimized GCDSAD, proposed a new method that computes the GCD of two 32-bit numbers using an absolute difference sum block.

2.3. Methods of performing the multiplication operation

Many algorithms have been developed for multiplication (standard, binary, Montgomery, Karatsuba-Ofman, etc.), which are characterized by different computational complexity. It should be noted that multiplication, in particular, modular, is the most critical operation when using asymmetric cryptosystems. For example, in [19] they present new optimal methods of modular multiplication, which are based on interleaved Montgomery multiplication on 16-bit MSP430X microprocessors. Meanwhile, a part of the multiplication is performed in the hardware multiplier, and part - in the basic arithmetic logic device. In [20] it is presented the implementation of multiplication and squaring for 32-bit ARM Cortex-M4 microcontrollers. Implementation and research of fast modular exponentiation on FPGAs using multiplication is presented in [21]. B [22-23] a hardware architecture for efficient modular exponentiation in asymmetric cryptography is presented. In [24] prospects for the use of modern software in Montgomery arithmetic are given.

However, theoretical studies do not always show the real picture regarding the speed of calculations, as they do not take into account all the factors that affect the computer's operation, in particular, memory access, features of the processor load, its bit rate and parallelization of calculations, etc.

Therefore, the aim of our work is to find, compare and analyze the time of simultaneous execution of the Euclid algorithm and multiplication of two multi-bit numbers performed using the classical and proposed methods for numbers of different bits on a computer with given parameters.

3. Proposed model

3.1. The method of searching for GCD using parallelization

To parallelize the search for GCD, it is advisable to use the proposed algorithm based on RNS, which involves working with residues [25]. It consists of the following stages.

1. Numbers are represented in the binary numbering system:

$$X = x_{n-1} \cdot 2^{n-1} + x_{n-2} \cdot 2^{n-2} + \mathrm{K} + x_i \cdot 2^i + \mathrm{K} + x_0 \cdot 2^0\;;$$

$$Y = y_{n-1} \cdot 2^{n-1} + y_{n-2} \cdot 2^{n-2} + \mathrm{K} + y_i \cdot 2^i + \mathrm{K} + y_0 \cdot 2^0\;.$$

Where n is the bit rate of the input data.

2. Binary codes of numbers *X* and *Y* are represented in the delimited system of residual classes in the form of residual vectors according to the following expressions:

$$(x_i \cdot 2^i) \bmod p_j = a_{ij}\,,\ (y_i \cdot 2^i) \bmod p_j = b_{ij}\,, \tag{1}$$

where *i*=*n*-1, *n*-2, …, 1, 0; *j*=0, 1, …, *k*.

As a result, the codes of numbers X, Y in the system of simple modules p_j can be represented in the form of residual vectors that form the corresponding matrices:

$$\begin{vmatrix} a_{n-1,1} & a_{n-2,1} & \Lambda & a_{i,1} & \mathrm{K} & a_{0,1} \\ a_{n-1,2} & a_{n-2,2} & \mathrm{K} & a_{i,2} & \mathrm{K} & a_{0,2} \\ \Lambda & \Lambda & \Lambda & \Lambda & \Lambda & \ldots \\ a_{n-1,j} & a_{n-2,j} & \mathrm{K} & a_{i,j} & \mathrm{K} & a_{0,j} \\ \Lambda & \Lambda & \Lambda & \Lambda & \Lambda & \Lambda \\ a_{n-1,k} & a_{n-2,k} & \mathrm{K} & a_{i,k} & \mathrm{K} & a_{0,k} \end{vmatrix} \qquad \begin{vmatrix} b_{n-1,1} & b_{n-2,1} & \Lambda & b_{i,1} & \mathrm{K} & b_{0,1} \\ b_{n-1,2} & b_{n-2,2} & \mathrm{K} & b_{i,2} & \mathrm{K} & b_{0,2} \\ \Lambda & \Lambda & \Lambda & \Lambda & \Lambda & \Lambda \\ b_{n-1,j} & b_{n-2,j} & \mathrm{K} & b_{i,j} & \mathrm{K} & b_{0,j} \\ \Lambda & \Lambda & \Lambda & \Lambda & \Lambda & \Lambda \\ b_{n-1,k} & b_{n-2,k} & \mathrm{K} & b_{i,k} & \mathrm{K} & b_{0,k} \end{vmatrix} \tag{2}$$

The given representation of X, Y in the form of residual vectors can be summarized as follows:

$X = \|a_{ij}\|$, $Y = \|b_{ij}\|$, де *i*=*n*-1, *n*-2, …, 1, 0; *j*=0, 1, …, *k*., $0 \le a_{ij}, b_{ij} \le p_j - 1$.

Then, for the multiplicative GCD (MGCD) *z*, the inequality must hold:

$$1 \le z \le \prod_{j=1}^{k} p_j\,. \tag{3}$$

3. Next, the elements of matrices (2) need to be represented in the form of two vectors:

$$a_j = \left(\sum_{i=1}^{n-1} a_{ij} \right) \bmod p_j ; b_j = \left(\sum_{i=1}^{n-1} b_{ij} \right) \bmod p_j \tag{4}$$

for all $j \in \overline{0, k}$.

From (4), two vectors are obtained that represent numbers X, Y in the integer system of residual classes:

$$\begin{aligned} X &= (a_1 \quad a_2 \quad \ldots \quad a_j \quad \ldots \quad a_k); \\ Y &= (b_1 \quad b_2 \quad \ldots \quad b_j \quad \ldots \quad b_k). \end{aligned} \tag{5}$$

To determine the components (modules) of the MGCD *z*, it is necessary to perform a comparison $a_j = b_j = 0,\ \forall j \in \overline{1, k}$. Then, the desired MGCD is obtained from the following expressions:

$$z = \prod_{j=1}^{k} p_j\,,\ p_j = \begin{cases} p_j, a_j = b_j = 0 \\ 1, a_j \ne b_j \end{cases}. \tag{6}$$

4. In order to find the GCD *Z*, the input numbers *X* and *Y* are represented in the delimited numbering system by modules p_j that meet the condition (6). Then, according to (2), matrices of residues are created by modules $p_j^{\,m}$, where m, is the exponent of the degree for which condition (6) is fulfilled:

$$\begin{bmatrix} a_j^{(2)} = \left(\sum_{i=1}^{k-1} a_{ij}\right) \bmod p_j^{2} \\ a_j^{(3)} = \left(\sum_{i=1}^{k-1} a_{ij}\right) \bmod p_j^{3} \\ \ldots\ldots\ldots\ldots\ldots\ldots \\ a_j^{(m)} = \left(\sum_{i=1}^{k-1} a_{ij}\right) \bmod p_j^{m} \end{bmatrix}; \quad \begin{bmatrix} b_j^{(2)} = \left(\sum_{i=1}^{k-1} b_{ij}\right) \bmod p_j^{2} \\ b_j^{(3)} = \left(\sum_{i=1}^{k-1} b_{ij}\right) \bmod p_j^{3} \\ \ldots\ldots\ldots\ldots\ldots\ldots \\ b_j^{(m)} = \left(\sum_{i=1}^{k-1} b_{ij}\right) \bmod p_j^{m}. \end{bmatrix} \tag{7}$$

The verification of condition (6) for powers is performed according to comparisons

$$\begin{aligned} X^{(m)} &= \ (a_j^{(2)} \quad a_j^{(3)} \quad \ldots \quad a_j^{(k)}); \\ Y^{(m)} &= \ (b_j^{(2)} \quad b_j^{(3)} \quad \ldots \quad b_j^{(k)}). \end{aligned} \tag{8}$$

Then the GCD is calculated according to the following multiplicative function:

$$Z = \prod_{j=1}^{k} p_j^{m} . \tag{9}$$

In comparison with the well-known Euclidean algorithm, the proposed algorithm for finding the GCD is characterized by the following advantages:

1. Formation of matrices can occur in parallel using several processors.
2. Simultaneously with the search for the GCD, its factorization takes place.

An improved method of searching for GCD using parallelization

The proposed algorithm can be significantly improved by removing the third step with the fulfillment of an additional condition:

$$\min j : a_j = b_j = 0 . \tag{10}$$

The search of the GCD for the components p_j that meet the condition (10) is carried out on the basis of the representation of the input numbers *X* and *Y* in the delimited RNS according to (2). Next, the matrices (10) of the residuals are formed by p_j^{m}, where m , is the exponent of the power at which the condition (10) is fulfilled. After that, the remainders are searched for products of modules, $p_{j,k}^{m_s} = p_j^{m} \cdot p_{j+k}^{s}$, where s is the exponent of the power at which the condition is fulfilled, $a_{j,k} = b_{j,k} = 0$, k is the next simple module after the j. Residue matrices are formed:

$$\begin{bmatrix} a_{j,k}^{(2)} = \left(\sum_{i=1}^{n-1} a_{ij}\right) \bmod p_{j,k}^{m_2} \\ a_{j,k}^{(3)} = \left(\sum_{i=1}^{n-1} a_{ij}\right) \bmod p_{j,k}^{m_3} \\ \ldots\ldots\ldots\ldots\ldots\ldots \\ a_{j,k}^{(m_s)} = \left(\sum_{i=1}^{n-1} a_{ij}\right) \bmod p_{j,k}^{m_s} \end{bmatrix} \quad \begin{bmatrix} b_{j,k}^{(2)} = \left(\sum_{i=1}^{n-1} b_{ij}\right) \bmod p_{j,k}^{m_2} \\ b_{j,k}^{(3)} = \left(\sum_{i=1}^{n-1} b_{ij}\right) \bmod p_{j,k}^{m_3} \\ \ldots\ldots\ldots\ldots\ldots\ldots \\ b_{j,k}^{(m_s)} = \left(\sum_{i=1}^{n-1} b_{ij}\right) \bmod p_{j,k}^{m_s} \end{bmatrix}. \tag{11}$$

Thus, the product of all simple modules to $\sqrt{Y}$ powers, for which the condition $a_{j,k} = b_{j,k} = 0$ is fulfilled, will be GCD, i.e.:

$$Z = \prod_{j=1}^{k} p_j^{m_j} . \tag{12}$$

The main advantage of this algorithm in comparison with the previous one is the avoidance of the search for MGCD, which will significantly increase the speed of operation.

3.2. Algorithmic support for the simultaneous execution of the Euclid algorithm and multiplication

In Rabin cryptosystem, the given numbers a and b are prime, so it is known that in this case GCD $(a,b)=r_n=1$. In contrast to the execution of the Euclidean algorithm followed by the multiplication of these same numbers, it is proposed, according to (1), to use the intermediate and final results of the Euclidean algorithm in the multiplication as follows:

$$a \cdot b = r_0^2 q_1 + r_1^2 q_2 + r_2^2 q_3 + \ldots + r_{n-1}^2 q_n + r_n^2 q_{n+1} . \tag{13}$$

It should be noted that the number of terms in (13) corresponds to the number of steps in Euclidean algorithm. Although this method involves the performance of a larger number of arithmetic operations, they will be performed on numbers of smaller digits. Having a table of squares in the computer's memory is an essential step to increase performance, although this leads to an increase in the use of computer system resources. The block diagram of the algorithm is shown in Figure 2.

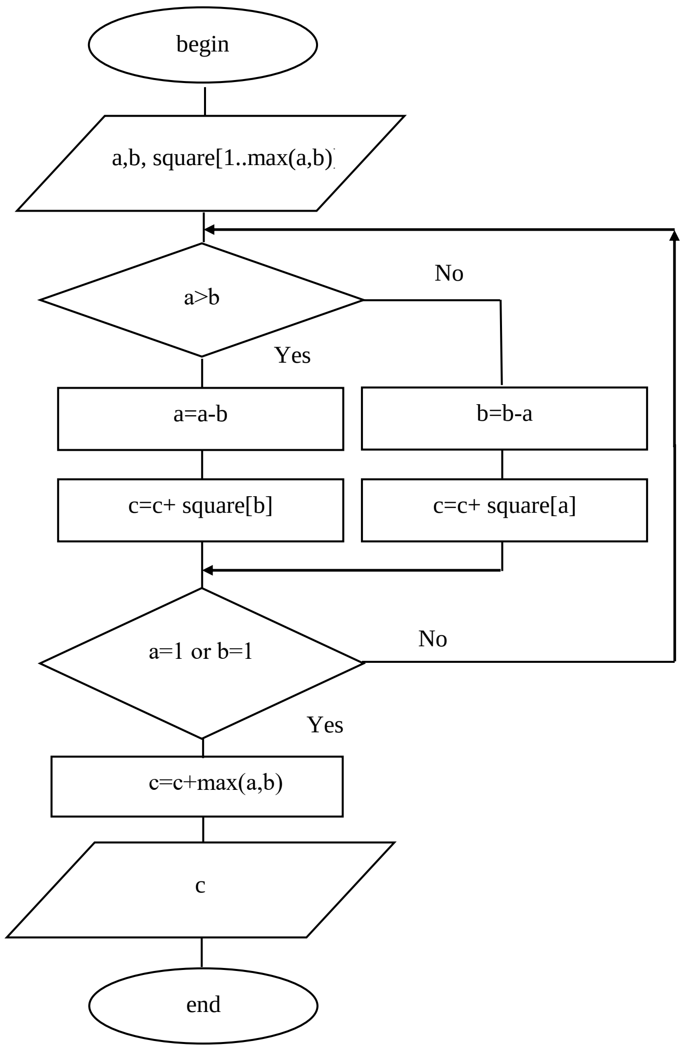

Figure 2: Block diagram of the proposed algorithm

Since in the Rabin cryptosystem the private key consists of simple numbers, it is appropriate to give an example with such numbers. Let *a*=757, *b*=131. Then:

757=131·5+102
131=102·1+29
102=29·3+15
29=15·1+14
15=14·1+1
14=1·14+0.
Obtained:
757·131=1312·5+1022·1+292·3+152·1+142·1+12·14=85805+10404+2523+225+196+14=99167.

4. Results and Discussions

4.1. Selection of hardware and software

A Lenovo IdeaPad 1 15IAU7 Cloud Gray laptop computer with an Intel Core i5-1235U processor (4.4 GHz) was chosen for experimental research. The amount of RAM in the device was 8 GB. When designing the software complex that provided calculations, the C++ high-level programming language was chosen. A special purpose library written by A. Lenstra was used to work with multi-bit numbers. This made it possible to provide flexible possibilities for working with multi-bit numbers, the size of which depends only on available system resources. The selected programming language and the cross-platform nature of the special library allow you to transform the codes for different architectures and operating systems. The most common method of software implementation of Euclidean algorithm is to subtract a smaller number from a larger number successively until the difference becomes less than the denominator. Then the same procedure must be performed with the subtractor and the difference. The subtraction process will continue until the numerator and the difference are equal.

4.2. Obtained results and their discussion

Table 1 presents the time of simultaneous execution of the Euclidean algorithm and multiplication of numbers by classical (t_1) and proposed (t_2) methods for b=2, 3, …, 70 if a=71, and in Figure 3 - the corresponding graphic dependence (t_1 and t_{1av} - dashed line, t_2 and t_{2av} – solid).

Table 1
The time of simultaneous execution of the Euclidean algorithm and multiplication of numbers by classical (t_1) and proposed (t_2) methods

b	2	3	4	5	6	7	8	9	10	11	12
t_1, s	0,011	0,028	0,033	0,035	0,034	0,017	0,036	0,035	0,037	0,038	0,034
t_2, s	0,030	0,033	0,037	0,019	0,034	0,036	0,034	0,033	0,026	0,029	0,034
b	13	14	15	16	17	18	19	20	21	22	23
t_1, s	0,036	0,027	0,044	0,042	0,035	0,033	0,049	0,035	0,059	0,034	0,030
t_2, s	0,030	0,014	0,026	0,014	0,027	0,029	0,030	0,044	0,029	0,027	0,014
b	24	25	26	27	28	29	30	31	32	33	34
t_1, s	0,033	0,035	0,047	0,049	0,053	0,036	0,044	0,037	0,047	0,049	0,036
t_2, s	0,015	0,027	0,043	0,045	0,027	0,029	0,028	0,026	0,029	0,030	0,014
b	35	36	37	38	39	40	41	42	43	44	45
t_1, s	0,034	0,036	0,036	0,049	0,060	0,049	0,047	0,035	0,053	0,047	0,060
t_2, s	0,014	0,015	0,030	0,043	0,030	0,030	0,044	0,043	0,031	0,045	0,030
b	46	47	48	49	50	51	52	53	54	55	56
t_1, s	0,049	0,052	0,035	0,036	0,049	0,052	0,052	0,051	0,050	0,036	0,043
t_2, s	0,031	0,030	0,030	0,043	0,031	0,030	0,034	0,014	0,027	0,027	0,029
b	57	58	59	60	61	62	63	64	65	66	67
t_1, s	0,035	0,049	0,036	0,052	0,035	0,037	0,038	0,036	0,034	0,037	0,035
t_2, s	0,014	0,015	0,026	0,014	0,026	0,028	0,025	0,026	0,032	0,014	0,024
b	68	69	70								
t_1, s	0,047	0,036	0,037	Average time: t_{1av} = 0,040333 s.							
t_2, s	0,030	0,031	0,016	Average time: t_{2av} = 0,028174 s.							

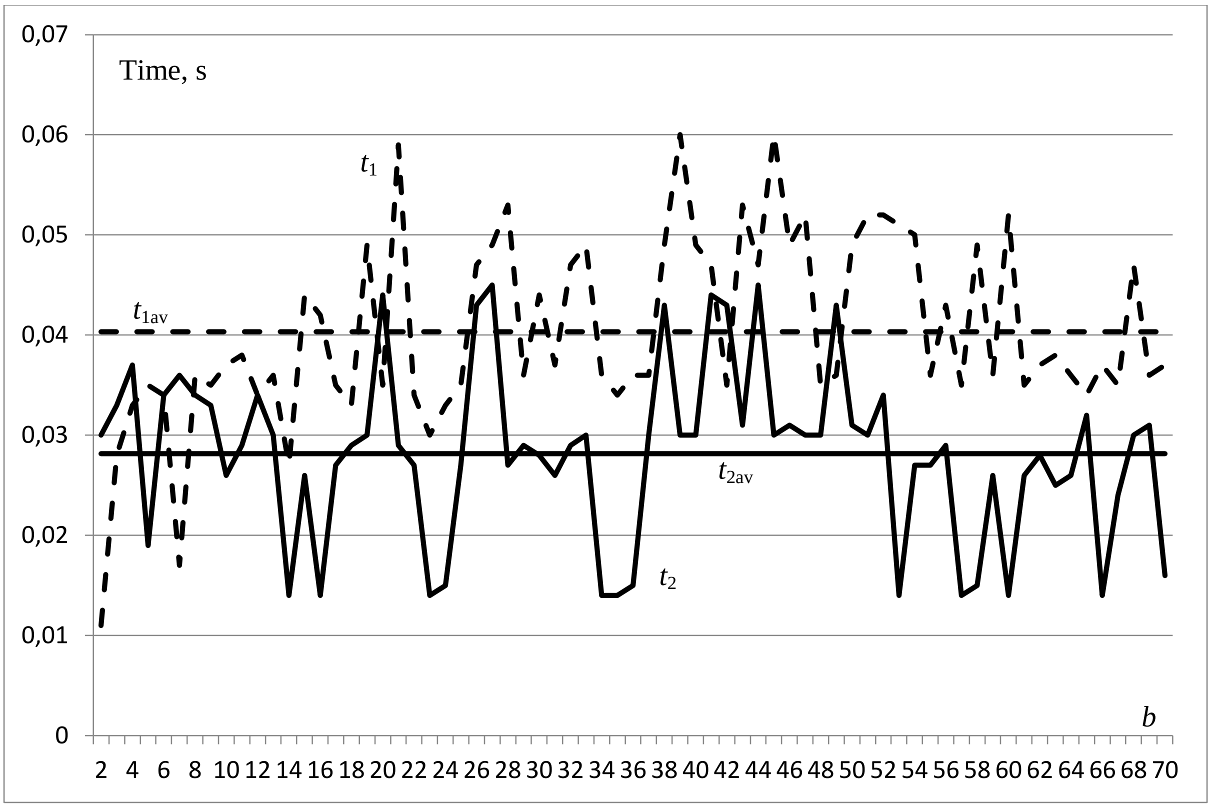

Figure 3: Graphic dependence of the time of simultaneous execution of the Euclidean algorithm and multiplication by classical (t_1) and proposed (t_2) methods

The graphs have an oscillating character, which is explained by the different number of steps in the Euclid algorithm for different numbers. In 60 cases out of 69, which is 87%, calculations using the proposed method are performed faster, in 7 (10%) - slower, in 2 cases (3%) the execution time of both methods is the same. The average time values are respectively t_{1av}=0,040333 s and t_{2av}=0,028174 s, which is also presented on the graph. Therefore, the speed increased by an average of 1.43 times.

Table 2 presents the average time of simultaneous execution of the Euclidean algorithm and multiplication by the classical (t_3) and proposed (t_4) methods in the case when the prime numbers *a* are within one bit from 67 to 127, and Figure 4 shows the corresponding graphs depending on the number. Meanwhile, *b* changes from 2 to *a*-1.

Table 2

Presents the average time of simultaneous execution of the Euclidean algorithm and multiplication by the classical (t_3) and proposed ($_{t4}$) methods

№	1	2	3	4	5	6	7
a	67	71	73	79	83	89	97
t_3, s	0,036378	0,037795	0,037279	0,038432	0,038529	0,038299	0,037149
t_4, s	0,026	0,029734	0,030091	0,029974	0,030903	0,030181	0,030925
№	8	9	10	11	12	13	
a	101	103	107	109	113	127	
t_3, s	0,038799	0,038494	0,040197	0,039259	0,039557	0,039558	
t_4, s	0,032202	0,031235	0,030418	0,030805	0,030545	0,03127	

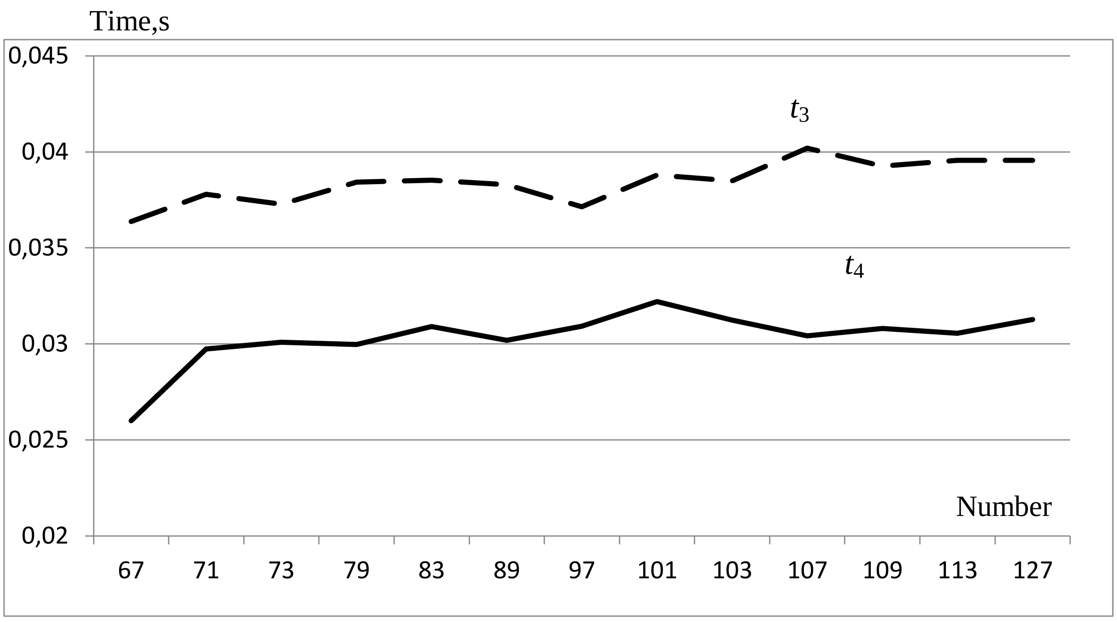

Figure 4: Graphical dependence of the average time of simultaneous execution of the Euclidean algorithm and multiplication by classical (t_3) and proposed (t_4) methods on the number of the number according to Table 2

It can be seen from Figure 4 that the average working time of the proposed method is in all cases less than that of the classical method. The general trend shows an increase in time when the given numbers increase, and the graph for the classical method grows more intensively.

Table 3 presents the average time of simultaneous execution of the Euclidean algorithm and multiplication by the classical (t_5) and proposed (t_6) methods in the case when the bit size of n prime numbers a is in the range from 7 to 16 bits, and in Figure 5 - the corresponding graphic dependencies in the logarithmic scale. The number b changes similarly to the previous case.

Table 3
The average time of simultaneous execution of the Euclidean algorithm and multiplication by classical (t_5) and proposed (t_6) methods

a	131	181	257	359	521	727
$\log_2 a$	7	7,5	8	8,5	9	9,5
t_5, s	0,040647	0,039552	0,041882	0,043543	0,044861	0,046143
t_6, s	0,031404	0,032203	0,032901	0,034511	0,034842	0,035637
a	1031	1447	2053	2897	4099	5791
$\log_2 a$	10	10,5	11	11,5	12	12,5
t_5, s	0,048351	0,049332	0,051031	0,051939	0,05343	0,054777
t_6, s	0,037071	0,037709	0,039201	0,039927	0,040637	0,041501
a	8209	11587	16411	23173	32771	46337
$\log_2 a$	13	13,5	14	14,5	15	15,5
t_5, s	0,056491	0,05767	0,059349	0,060891	0,062278	0,063533
t_6, s	0,042513	0,043564	0,044601	0,045537	0,04623	0,047939

It can be seen that the average working time increases almost linearly with the increase in the number of bits, and the graph for the classical method grows more intensively.

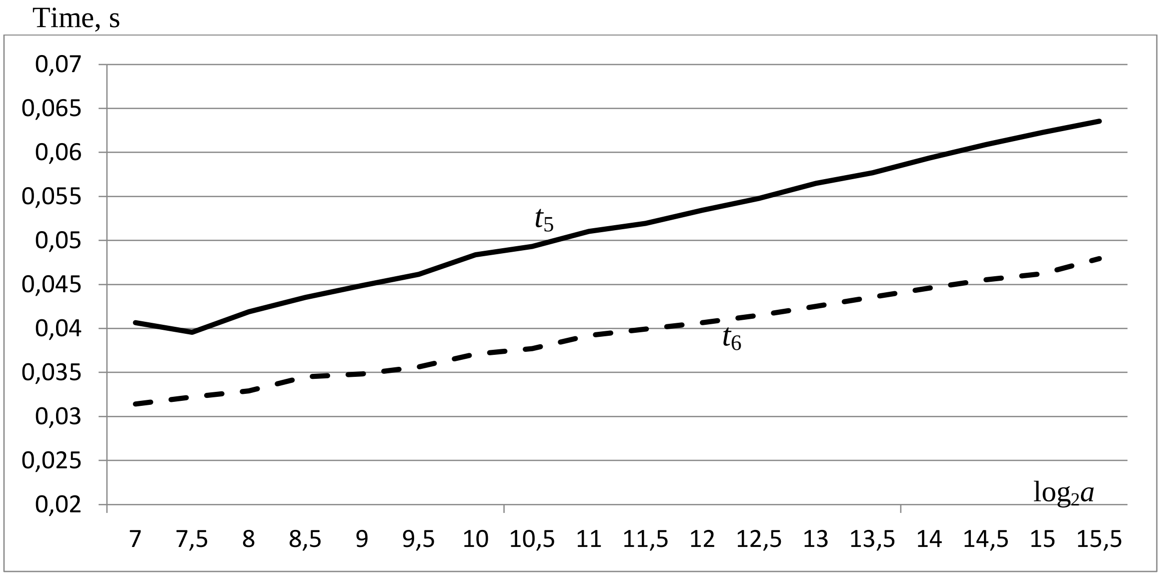

Figure 4: Graphical dependence of the average time of simultaneous execution of the Euclid algorithm and multiplication by the classical (t_5) and proposed (t_6) methods on the number of digits of *a*

Table 4 presents the average time of simultaneous execution of the Euclid algorithm and multiplication by the classical (t_7) and proposed (t_8) methods in the case when the digit number of *n* prime numbers a (parameter a was assigned the value of the smallest prime number that exceeded $2n$) is in the range from 16 to 44 bits, and Figure 6 shows the corresponding graphical dependencies in a logarithmic scale. The number b acquired 10,000 different values.

Table 4
The average time of simultaneous execution of the Euclid algorithm and multiplication by classical (t7) and proposed (t8) methods

$\log_2 a$	16	20	24	28	32	36	40	44
t_7, s	0,06741	0,07823	0,08839	0,09791	0,10651	0,11875	0,12629	0,13231
t_8, s	0,05003	0,05834	0,06712	0,07554	0,08627	0,09764	0,10509	0,10901

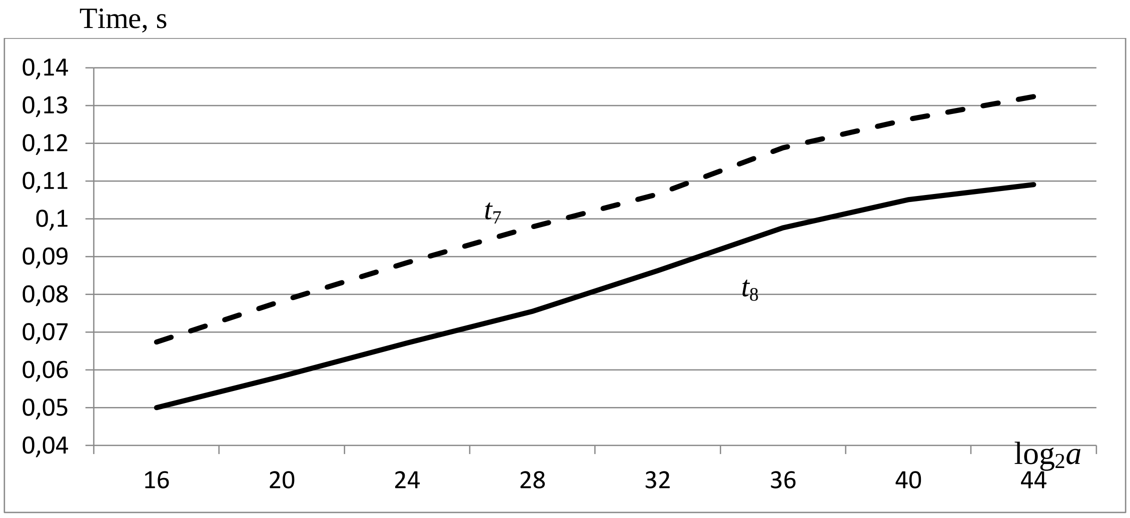

Figure 6: Graphical dependence of the average time of simultaneous execution of the Euclid algorithm and multiplication by classical (t_7) and proposed (t_8) methods on the number of digits of *a*

As it can be seen from the figure, the graphs are placed almost parallel. With large bits ($n \geq 40$), the intensity of t_8 growth decreases.

All calculations were repeated 5000 times to eliminate random effects on runtime.

5. Conclusion

The work deals with an experimental study of the time of simultaneous execution of the Euclidean algorithm and multiplication of two multi-bit numbers by classical and proposed methods. The C++ high-level programming language is used. The proposed method stipulates the use of intermediate results of the Euclidean algorithm and the table of squares available in the computer's memory. The research was conducted on numbers of different digits. It is shown that for the vast majority of the considered numbers, the proposed method requires less time to perform arithmetic operations. The average time of simultaneous execution of the Euclidean algorithm and multiplication decreases by approximately 1.43 times.

In addition, new methods of finding the greatest common divisor based on the formation of residual matrices are given. This makes it possible to parallelize the process of processing multi-bit numbers and, accordingly, increase the speed. It was established that the procedure for finding the greatest common divisor by the proposed methods is accompanied by its factorization.

References

[1] K.C. Fuson, The Best Multidigit Computation Methods: ACross-cultural Cognitive, Empirical, and Mathematical Analysis. Universal Journal of Educational Research, Vol. 8 (4), 2020, pp. 1299-1314. doi:10.13189/ujer.2020.080421

[2] J.S. Milne, Algebraic Number Theory. Version 3.08. MilneANT, 2020, 166 p.

[3] P. Pandey, R. Gupta, M. Khan, S. Iqbal, Multi-Digit Number Classification using MNIST and ANN. International Journal of Engineering Research & Technology (IJERT), 9 (05) (2020) 415-421.

[4] C. Luo, Y. Hao, Z. Tong, Research on Digital Image Processing Technology and Its Application. Proceedings of the 2018 8th International Conference on Management, Education and Information (MEICI 2018). Series: Advances in Intelligent Systems Research, 2018, pp.587-592. DOI: 10.2991/meici-18.2018.116

[5] K. Priya, B. Rasheeda Banu, T. Umme Habiba, A. Boosra, Digital image processing techniques-a review. International Journal of Emerging Technologies and Innovative Research, 6 (9) (2019) 56-61.

[6] L. Zhang, L. Zhang, L. Zhang, Application research of digital media image processing technology based on wavelet transform. J Image Video Proc., 138 (2018). doi: https://doi.org/10.1186/s13640-018-0383-6

[7] I.A.Aremu, K.A. Gbolagade, Redundant Residue Number System Based Multiple Error Detection and Correction Using Chinese Remainder Theorem. Software Engineering, Vol. 5 (5), 2017, pp. 72-80. doi: 10.11648/j.se.20170505.12

[8] H.Xiao, H.Garg, J.Hu, G.Xiao, New Error Control Algorithms for Residue Number System Codes. Electronics and Telecommunications Research Institute, 38 (2) (2016) 326-336. doi:https://doi.org/10.4218/etrij.16.0115.0575

[9] M.Kasianchuk, I.Yakymenko, Y.Nykolaychuk, Symmetric Cryptoalgorithms in the Residue Number System. Cybernetics and Systems Analysis, 57 (2) (2021) 329–336. doi:https://doi.org/10.1007/s10559-021-00358-6

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

[11] E.Y. Baagyere, P.A.-N. Agbedemnab, Z. Qin, M.I. Daabo, Z. Qin, A Multi-Layered Data Encryption and Decryption Scheme Based on Genetic Algorithm and Residual Numbers. IEEE Access, 4 (2016) 100438-100447. doi: 10.1109/ACCESS.2020.2997838

[12] J.-C. Bajard, J. Eynard, N. Merkiche, Montgomery Reduction within the Context of Residue Number System Arithmetic. Journal of Cryptographic Engineering, 8 (2018) 189-200. doi:https://doi.org/10.1007/s13389-017-0154-9

[13] M. Karpinski, S. Rajba, S. Zawislak, K. Warwas, M.Kasianchuk, S. Ivasiev, I. Yakymenko, A Method for Decimal Number Recovery from its Residues Based on the Addition of the Product Modules. Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications (IDAACS–2019): Proceedings of the 10th IEEE International Conference, Metz, France, 2019, pp.13–17. doi: 10.1109/IDAACS.2019.8924395

[14] E.A. Alhassan, K. Tian, O.J. Abban, I.E. Ohiami, M. Adjabui, G. Armah, S. Agyemang, On Some Algebraic Properties of the Chinese Remainder Theorem with Applications to Real Life. Journal of Applied Mathematics and Computation, 5(3) (2021) 219-224. doi: 10.26855/jamc.2021.09.008

[15] H. Al Daas, G. Ballard, P. Benner, Parallel Algorithms for Tensor Train Arithmetic. SIAM Journal on Scientific Computing, Vol. 44 (1), 2022, pp. C25-C53. DOI: 10.1137/20M1387158

[16] K. Anitha, T.S. Arulananth, R. Karthik, P. Bhaskara Reddy, Design and Implementation of Modified Sequential Parallel RNS Forward Converters. International Journal of Applied Engineering Research. 12 (16) (2017) 6159-6163.

[17] Ya.M. Nykolaychuk, I.Z. Yakymenko, N.Ya. Vozna, M.M. Kasianchuk, Residue Number System Asymmetric Cryptoalgorithms. Cybernetics and Systems Analysis, 58 (4) (2022) 611-618. doi: https://doi.org/10.1007/s10559-022-00494-7

[18] S. Nabipour, M. Gholizade, N. Nabipour, Efficeint FPGA Design of 32-bit Euclid’s GCD based on Sum of Absolute Difference. Technology in Electrical and Computer Engineering (ETECH-2021): Proceedings of the 5th IEEE Conference, Tafresh, Iran, 2021, pp. 2107-2113.

[19] H. Seo, K. An, H. Kwon, Z. Hu, Montgomery Multiplication for Public Key Cryptography on MSP430X. ACM Transactions on Embedded Computing Systems, Vol. 19 (3), 2020, pp. 1-15. doi:https://doi.org/10.1145/3387919

[20] H. Seo, Memory efficient implementation of modular multiplication for 32-bit ARM Cortex-M4. Applied Sciences, Vol. 10 (4), 2020, pp. 1539. https://doi.org/10.3390/app10041539

[21] S. Li, J. Tian, H. Zhu, Z. Tian, H. Qiao, X. Li, J. Liu, Research in Fast Modular Exponentiation Algorithm Based on FPGA. Measuring Technology and Mechatronics Automation (ICMTMA-2019): Proceedings of the 11th International Conference, Qiqihar, China, 2019, pp. 79-82. doi: 10.1109/ICMTMA.2019.00024

[22] S. Prasanth Kumar, K.J. Jegadish Kumar, B. Partibane, Efficient Modular Exponentiation Architectures for RSA Algorithm. International Journal of Engineering Research in Electronic and Communication Engineering (IJERECE), 3 (5) (2016) 230-234.

[23] S. Vollala, B.S. Begum, A.D. Joshi, N. Ramasubramanian, High-radix Modular Exponentiation for hardware implementation of Public-Key Cryptography, Computing, Analytics and Security Trends (CAST-2016): Proceedings of the International Conference, Pune, 2016, pp. 346-350. doi: 10.1109/CAST.2016.7914992

[24] J.W. Bos, P.L. Montgomery, Montgomery Arithmetic from a Software Perspective. Cryptology ePrint Archive Report 2017/1057, 2017, 30, URL: https://eprint.iacr.org/2017/1057.pdf

[25] 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. doi:https://doi.org/10.1109/ACITT.2019.8779899