UDC 004. 004.4 (043.2)

ORGANIZATION OF PARALLEL EXECUTION OF MODULAR MULTIPLICATION TO SPEED UP THE COMPUTATIONAL IMPLEMENTATION OF PUBLIC-KEY CRYPTOGRAPHY

I. Boiarshyn, O. Markovskyi, B. Ostrovska

*The article theoretically substantiates, investigates and develops a method for parallel execution of the basic operation of public key cryptography - modular multiplication of numbers with high bit count. It is based on a special organization of the division of the components of modular multiplication into independent computational processes. To implement this, it is proposed to use the Montgomery modular reduction. The described solution is illustrated with numerical examples. It has been theoretically and experimentally proven that the proposed approach to parallelization of the arithmetical process of modular multiplication makes it possible to speed up this important for cryptographic tasks operation by 5-6 time*s.

Key words: *modular multiplication, Montgomary modular reductions, open key cryptography, parallel computation, multiplicative operations of modular arithmetic*.

Introduction

The process of modular exponentiation, which is executed on numbers whose bit count significantly exceeds the bit count of the processor, is the fundamental operation for a wide range of cryptographic algorithms which are based on irreversible problems of number theory. In particular, this operation is the basis of computational implementation of RSA, El-Gamal, digital signature standard, FESIS scheme of strict identification of remote users [1].

The protection level of cryptographic security mechanisms, which are based on the operation of modular multiplication, is fully determined by the bit count of the module [2]. To date, 2048 bits have been enough for most practical tasks.

At the same time, an analysis of the dynamics of the improvement of applied problems in which public key data protection mechanisms are practically used shows that a significant part of them is performed in real time and requires fast implementation of the corresponding calculations. Another vital feature of the use of modular arithmetic at the present stage of development of public key cryptography is the increase in the number bit count used. The dynamics of the improvement of cloud technologies potentially provides attackers with the ability to remotely access large computing power, which can be used to break cryptographic protection mechanisms. This catalyzed the need for an adequate increase in the level of security, which for cryptographic mechanisms with a public key can be achieved by increasing the bit count. This leads to a noticeable increase in the time of computational implementation of cryptographic data security mechanisms.

Therefore, the task of scientific research is to speed up calculations that implement public key cryptographic mechanisms by using the multiprocessor capabilities of modern computer systems. The main way of solving this problem is the parallelization of the basic operation - the modular multiplication.

The scientific problem of speeding up modular multiplication for cryptographic information security systems is relevant for the present stage of development of information and computer technologies.

Problem statement and review of methods for its solution

Modern public key cryptography was founded at the operation of modular exponentiation. The problem of rapid implementation of this operation is of key importance for the development of information protection complexes and information security.

For personal computers and powerful systems, this problem can be solved by including crypto processors in the hardware. To date, a significant range of cryptoprocessors [4] is commercially produced, almost all of which implement the modular exponentiation operation at the hardware level.

But for a wide class of mobile computer devices, terminal microcontrollers of systems for remote control of real-world objects, in which the Internet is used as a data exchange medium, the problem of fast implementation of the modular exponentiation operation is very acute. For many critical applications, the use of crypto processors is unacceptable for information security reasons.

It is a widespread knowledge that the classical scheme of modular exponentiation is strictly sequential and practically cannot be parallelized [5]. Therefore, the main point for increasing the speed of calculating the modular exponent is the parallelization of its fundamental operation - modular multiplication.

The operation of modular multiplication of numbers of large bit count $A \cdot B \ mod \ X$ consists of two parts: multiplication of the components $Y=A \cdot B$ and finding the residue after dividing the product $A \cdot B$ by the module X. In public key cryptography, the module X is part of the public key, so it can be considered constant [6].

Algorithms for modular multiplication are divided into two groups: gradual, in which the multiplication $A \cdot B$ and the gradual calculation of the residue from division by the module are execute sequentially in time and alternating, in which the operations of multiplication and finding the remainder of the division are combined in time [7].

For modular multiplication algorithms of the first group, there are special possibilities to use the processor's built-in multiplication instructions. To do this, N numbers that take part in the modular multiplication operation are divided into K fragments, the length S of which is equal to the processor bit count. Accordingly, the procedure of modular multiplication is reduced to pairwise multiplication of fragments with gradual summation of the results obtained. This method makes it possible to use the hardware of modern processors with high efficiency, and, in particular, the built-in circuits for multiplication acceleration [8].

In the basic algorithm, modular reduction is executed using the operation of integer division of a 2·s-bit dividend by a s-bit divisor to obtain a quotient and a remainder [9]. Since the division of N-bit numbers on a s-bit processor ($N>>s$) is very inefficient, the reduction in the basic algorithm requires $k\cdot(k+2.5)$ multiplication operations and k integer division operations [10]. To date, a number of algorithms [11, 12] have been declared to improve the performance of the software implementation of the modular multiplication operation. Most of them implement an increase in the performance of modular multiplication due to the acceleration of modular reduction by eliminating the operation of integer division, which is used in the basic algorithm [9].

Two technologies for finding the modulo modulo remainder have received the widest practical use: Barrett's algorithm [9] and Montgomery algorithm [13]. The first one is funded in calculating the minimum value of m for which $A\cdot B - m\cdot X < X$. Accordingly, the remainder of the division is calculated as $R = A\cdot B - m\cdot X$. Thus, Barrett's algorithm is implemented with two s-bit multiplications, while modular multiplication $A\cdot B \ mod \ X$ using Barrett's algorithm is implemented with three.

Another technology for calculating the modulo product residue, the Montgomery algorithm, is well adapted to the universal processor architecture. The algorithm replaces the operation of division by a random modulus X with divisions by a power of 2, which are effectively implemented by shifts. The modular reduction operation in Montgomery's algorithm requires $k\cdot(k+1)$ multiplication operations.

The total computational complexity of the implementation of the algorithm for modular multiplication of N-bit numbers using the Montgomery algorithm on a s-bit processor is determined by $2\cdot k2$ + s processor multiplication operations and $4\cdot k2+4\cdot k+2$ processor addition operations. A significant advantage of the Montgomery algorithm is that it is relatively easy to combine in time with the multiplication process. This allows, in the process of modular multiplication, to limit the length of intermediate results to $N + 1$ and thereby reduce the amount of calculations compared to the sequential scheme that works with $2\cdot N$ -bit intermediate results [14].

Two approaches are most often used [15] to speed up modular multiplication:

- precalculations depending only on the value of the module, which are stored in a special table memory;
- simultaneous processing of several digits of the multiplier;
- parallelization of operations of summation of fragments of a modular product.

In practice, these three approaches are often used in combination.

The analysis of existing methods to speed up the execution of modular multiplication showed that with limits of computational processes that run on a single processor, the possibilities of obtaining new results are practically exhausted. This means that a further increase in the speed of the computational implementation of the modular multiplication operation, necessary for practical problems, can be achieved only by using the capabilities of multi-core processor architectures.

Purpose and objectives of research

The target of the research is to speed up the execution of the modular multiplication operation on numbers, which is important for cryptographic tasks, the bit count of which significantly exceeds the bit count of the processor, due to the organization of parallel calculation of fragments of the modular product on multi-core computers.

The following set of tasks is solved in the work to achieve the target goal:

- analysis of the computing process of modular multiplication due the point of view of its parallelization possibilities; description for choosing a scheme of modular reduction, which combining in time with the multiplication process;
- creating of a method of parallel modular multiplication using a multi-core architecture, which, due to the division of the computing process into loosely connected fragments, allows to organize their parallel processing, due to which acceleration of the computational implementation of modular multiplication is achieved;
- optimization of the structure of the parallel computation of the Modular multiplication according to the criterion of maximum exploitation of processor elements;
- theoretical evaluation of the effectiveness of the developed method of accelerated modular multiplication;
- software development and experimental evaluation of the effectiveness of the proposed method of parallel modular multiplication of numbers, the bit count of which significantly exceeds the bit count of the processor.

The object of research to which the article is devoted are the processes of calculating multiplicative operations of modular arithmetic, which are performed on numbers, the length of which is orders of magnitude greater than the bit capacity of processors.

The method of parallel calculation of the modular product on multicore processors

To achieve this target, is declared the following organization of parallel computation of the modular product $A \cdot B \bmod X$ in the form of s independent computational processes. Accordingly, these computational processes are performed on s cores. For this, the N-bit factor $A=a1+a1\cdot 21+a2\cdot 22+\ldots+aN\cdot 2N$, $\forall j\in\{1, 2, \ldots, N\}: aj\in\{0,1\}$, is decomposed into s partial factors A1, A2, …, As with N bits. Each i-th, $i \in\{1,2, \ldots, s\}$, N-bit partial multiplier Ai includes those r=N/s digits of the multiplier A, the residue of dividing by s numbers of which is equal to i, the remaining digits of the N-bit partial multiplier multipliers are zero. In other words, each i–th partial factor Ai can be represented as:

$$A_i = \sum_{j=i}^{(r-1)\cdot s} a_j \cdot 2^j \qquad (1)$$

The above can be illustrated by the following example: if the bit depth is N=12 and the number of independent computing processes is s=3, then the factor $A = 0011\ 1001\ 1100_2 = 924_{10}$, is divided into 3 partial factors, each of which contains $r=N/s$=4 significant binary bits of the full multiplier A. The partial multiplier A_0 includes every fourth, starting from the least significant, that is, the 1st, 4th and 7th bits of the full multiplier A: A_1=0010 0000 1000_2. The second partial multiplier A_1 contains the 2-nd, 5-th and 8-th digits of the full factor A: A_2=0000 1001 0000_2.The last, third partial factor is: $A_3 = 0001\ 0100\ 0100_2$.

It is quite obvious that the disjunction of all partial factors is equal to the factor A of the modular product: $A_1 \cup A_2 \cup \ldots \cup A_s = A$, and the conjunction of partial products is equal to zero: $A_1 \cap A_2 \cap \ldots$

$\cap A_s = 0$. This means that the modular product $A \cdot B \bmod X$ is equal to the sum of modular products of partial factors and multiplier B:

$$A \cdot B \bmod m = (\sum_{i=1}^{s} A_i \cdot B \bmod m) \bmod m \quad (2)$$

Therefore, the separation of the significant digits of the factor A by partial factors $A_1, A_2, \ldots, A_s$, which are independently modularly multiplied by the multiplier B, provides an increase in the speed of calculating the modular product due to the parallelization of the process of modular multiplication. In addition, the predominance of zeros in each of the partial factors, which are independently multiplied by the multiplicand, creates conditions for the effective use of precalculations in the process of calculating modulo modulus residues using the Montgomery technology [2].

Montgomery's algorithm is funded on the idea of replacing the calculation of modular reduction modulo X with the calculation of reduction modulo M, which is a power of 2, so that division operations are reduced to shifts.

Montgomery's technology allows instead of calculating $Y \bmod X$ to calculate $R=Y \cdot M^{-1} \bmod X$ without the division operation, where M^{-1} is the modular inversion of M. After that, to obtain Y mod X the calculated value of R is multiplied by M mod X: $Y \bmod X = (Y \cdot M^{-1} \bmod X \cdot \text{M mod } X) \bmod X = (Y \cdot M \cdot M^{-1}) \bmod X = (Y \cdot 1) \bmod X$. Thus, to compute Y mod X one must compute M mod X. However, in practice, applying the Montgomery reduction $X < M < 2 \cdot X$, so that $M \bmod X = M - X$. That is, the calculation of M mod X is reduced to one operation of subtracting N-bit numbers. Usually $M=2^N$, and the module X is a number of length N binary digits, and the most significant bit of the binary representation of X equal to one: $X_{N-1}=2^{N-1}$.

When calculating $A \cdot B \bmod X$, the complexity of pre-computation and post-computation must be taken into account. If we take into account the complexity of calculating the modular product, taking into account the correction, which again requires the operation of modular multiplication by the modular inversion of 2^N modulo X, and multiplying it by the result using the Montgomery algorithm, then it turns out that the complexity will be $4 \cdot X \cdot (X+1)$**.** This means that when performing a single operation of modular multiplication, the Montgomery algorithm has no obvious advantages over the basic algorithm.

An analysis of these features allows us to formulate requirements for the organization scheme of partial calculations. The implementation of this computation, together with the preservation of the general principles on which the Montgomery algorithm is based (for example, minimization of all intermediate results by replacing them with smaller numbers congruent in a given modulo), allows us to obtain a working algorithm of parallel multiplication, which is more efficient. In this case, the organization of calculations must be single-pass: to obtain the result of modular multiplication, the calculation cycle must be performed only once. The classical Montgomery algorithm does not satisfy this requirement, because it is two-way. To form a general result based on the results of partial multiplications, the maximum load of processor cores is required.

As pointed, a characteristic feature of the declared variant of dividing the factor A into partial factors is the presence of local groups of zeros, the number of which is not less than s. This allows us to solve the problem of accelerating the calculation of the modular product by adding to the intermediate result not a module, but a linear combination of the module P(X) chosen in such a way that the lower s digits of the sum of the intermediate result and this linear combination $Y+P(X)$ are equal to zero. Accordingly, after that, the resulting sum is shifted to the right by s bits at once without loss of significant bits. It is quite obvious that such a solution makes it possible to speed up the reduction of the intermediate result by a factor of s at once. For the practical implementation of the proposed technology of accelerated calculation of the remainder of the division of the intermediate result by the module X it seems necessary to calculate in advance for each of the 2^s options for possible values of the lower s digits of the intermediate result Y the value of the linear combination of the module $P(X)$, the lower s digits of which are the algebraic complement of s lower digits of the intermediate result. The results obtained from such pre-calculations are stored in the form of a table T of pre-calculations. An example of the table T precomputation P(X) is given below in Table 1 for s=3 and the value of the module X= 23⊕29 = 667.

The full capacity of table memory for storing the results of precomputations $P(X)$ is $N \oplus 2^s$ bits. he way of storing P(X) in one table presented above can be considered as a special case of partitioned organization of tables of precomputation results. The use of multi-section tables can significantly reduce the amount of memory for their storage.

For example, under the conditions of the above example of the implementation of accelerated multiplication of 2048-bit numbers on an 8-bit microcontroller with two-section memory, the amount of memory required will be 8.57 times less than with a single-section organization of table memory. On the other hand, the use of a multi-section organization of table memory is associated with an increase in the execution time of modular reduction.

Table 1.
Linear Combination Precomputation Example $P(m)$
for module X= 23·29 = 667 and S=3

low-order Y	$P(X)$	low-order P(X)
$y_3\ y_2\ y_1$		$p_3\ p_2\ p_1$
0 0 1	$3335 = 4 \oplus X + X$	1 1 1
0 1 0	$1334 = 2 \oplus X$	1 1 0
0 1 1	$4669 = 4 \oplus X + 2 \oplus X + X$	1 0 1
1 0 0	$2668 = 4 \oplus X$	1 0 0
1 0 1	$667 = X$	0 1 1
1 1 0	$4002 = 4 \oplus X + 2 \oplus X$	0 1 0
1 1 1	$2001 = 2 \oplus X + X$	0 0 1

The presented method of modular multiplication with parallelization of calculations on s processor cores involves the simultaneous execution of procedures for calculating partial products on all processor cores, followed by their cascaded modular summation to reduce the time of generating the result of modular multiplication.

In this case, the procedure for calculating a partial modular partial product consists in performing the following sequence of actions:

1. The counter h of cycles is set to zero, as well as the Y code of the current result h=0; Y=0.
2. The partial factor A_r with the number r is shifted to the right by r-1 binary digits: A_r=A_r>>(r-1) with the high digits filled with zeros.
3. If the least significant digit of the partial product Y is equal to one: y_1 =1, then the multiplier B is added to the result code: Y += B.
4. If the value of the counter h of cycles is a multiple of the value s, then go to step 6.
5. To the code of the partial product Y the tabular code T[l], is added, addressed by s least significant digits of the result code $l = y_1 + 2 \oplus y_2 + \ldots + 2^{s-1} \oplus y_s$: Y += T[l].
6. Result code Y and multiplier B are shifted s bits to the right: Y= Y>>s; B = B>>s. The cycle counter h of the algorithm is increased by one: h++, the transition to the repeated execution of paragraph 3 of the algorithm is performed.
7. To the code of the partial product Y, the tabular code T[l] is added, addressed by s-r lower significant digits of the result code$l = y_1 + 2 \oplus y_2 + \ldots + 2^{s-r-1} \oplus y_{s-r+1}$: Y += T[l].
8. The Y result code is shifted s-r bits to the right: Y= Y >>s-r.
9. End.

After performing the described procedure, Y has generated a modular product code containing $A \cdot B \cdot M^{-1} \bmod X$, where M^{-1} is the multiplicative inversion of M=2^N modulo X. To obtain the correct result, the resulting Y code must be modularly multiplied by M: $Y' = M \cdot Y \bmod X$. However, when performing the operation of modular multiplication as a component of modular exponentiation, corrective multiplication by code M is performed only once, after all cycles of the classical algorithm of modular exponentiation have been executed.

Evaluation of the effectiveness of the method of parallel modular multiplication

It is expedient to estimate the efficiency of the proposed method of modular multiplication by means of the achieved acceleration of the computational implementation of this operation when using s processor cores. The numerical expression for the acceleration estimate can be the coefficient q, which is determined by the ratio of the time t_1 for performing modular multiplication in the form of a single process using the Montgomery reduction to the time t for performing this operation in the form of s parallel processes using the developed method:

$$q = \frac{t_1}{t_k} \tag{3}$$

The time t_1 of performing the operation of modular multiplication on one processor using the alternation of the multiplication cycle and the reduction of the precalculation result is determined by the execution time of n cycles by the number of digits of the numbers. Each cycle, depending on the value of the current digit of the multiplier A в, the addition of the multiplier B to the code of the precalculation result Y is performed or not performed. After that, depending on the value of the least significant digit of the received sum Y, the addition of the module X to the code of the precalculation result Y is performed or not performed. ends with shifting the precalculation result code to the right by one bit. Thus, the cycle, on average, contains two operations on n-bit numbers. If the execution time of these operations is denoted by t_N, then $t_1=2 \cdot N \cdot t_N$.

The developed procedure provides for the implementation of the multiplication of the multiplier by the partial factor in the form of N/s cycles, in each of which the following is performed: adding the multiplicand to the result if the least significant bit of the partial factor is equal to one, adding the code from the precalculation table to the result, as well as shifting the multiplier and the result to the right. Accordingly, the average number of operations on n-digit numbers is 3.5, and the value $t_X=3.5 \cdot N \cdot t_N/s$. Thus, the numerical value of the acceleration coefficient q is determined by the following formula:

$$q = \frac{t_1}{t_k} = \frac{2 \cdot N \cdot t_N}{3.5 \cdot t_N \cdot \frac{N}{s}} \approx 0.57 \cdot s \tag{4}$$

Experiments on multi-core processors using a specially developed program showed the role of the acceleration factor equal to $0.5 \cdot s$, close to the predicted theoretical estimate.

Conclusion

As a result of the research aimed at increasing the speed of computer implementation of modular multiplication - the basic operation of public key cryptography based on unsolvable mathematical problems of number theory, the following results were obtained:

Theoretically substantiated, developed and investigated a method for parallelizing the operation of modular multiplication in the form of s independent parallel processes that can be executed on the cores of modern processors, a distinctive feature of which is the division of significant digits of the multiplier into different processes, due to which parallelization is ensured, which allows achieving real acceleration performing this important operation for cryptographic applications. The processing of insignificant digits of partial factors is performed in the form of Montgomery group reduction, which is an additional acceleration factor. To implement group reduction, precomputation tables are used, which depend only on the module and practically do not change, since the module is part of the public key of cryptosystems.

It has been theoretically and experimentally proven that the presented method makes it possible to speed up the computational implementation of modular multiplication by $0.57 \cdot$ s times.

The developed method is focused on application in multi-core computer systems to accelerate the implementation of a wide range of cryptographic data protection protocols with a public key.

References

1. Kabir L.A. The method of accelerating modular multiplication according to Montgomery technology / L.A. Kabir, O.V. Rusanova, I.O. Humenyuk //Almanac of Science No. 1(52).- 2022.- P.44-46.
2. Tribunska K.E. The method of accelerating modular multiplication using group reduction /K.E. Trybunska // Current issues of the development of science and education: materials of the M International Scientific and Practical Conference in Lviv, March 30-31, 2022.-Lviv: Lviv Scientific Forum, 2022.- P.38-46.
3. Giorgi P. Parallel modular multiplication on multi-core processors. / Giorgi P., Imbert L., Izard T. // IEEE Symposium on Computer Arithmetic, Apr 2013, Austin, TX, United States. - R.135-142.
4. Boyarshin I. Method of accelerated modular multiplication with Montgomery group reduction / I. Boyarshin, O. Markovskyi, B. Ostrovska // Proceeding of International Conference Security, Fault Tolerance, Intelligence. ICSFTI-2022, - Kyiv. - P.40-45.
5. Buhrow B. Parallel modular multiplication using 512-bit advanced vector instructions: RSA fault-injection countermeasure via interleaved parallel multiplication / Buhrow B., Gilbert B., Haider C. // Journal of Cryptographic Engineering.-2021.- No. 2.- R. 46-53.
6. Osadchyy V. The Order of Edwards and Montgomery Curves «, / V.Osadchyy // WSEAS Transactions on Mathematics, - 2020.- Vol. 19.- No. 25, - P. 253-264.
7. Haches G. Montgomery multiplication with no final subtraction./ G. Haches, J.J. Quisquater // Cryptographic Hardware and Embedded System - CHES'2000. LNCS-1965, Springer-Verlag. — 2000.- R. 293-301.
8. Markovskyi O.P. The method of accelerating exponentiation using recalculations / O.P. Markovskyi, O.V. Rusanova, A.A. Olievskyi, V.M. Cherevyk // Telecommunication and information technologies. - 2018.- No. 1(58). - P.31-39.
9. Elfard S. Justification of Montgomery Modular Reductions / S. Elfard //Advanced Computing.- 2012. - No. 11. – P.41-45.
10. Anisimov A.V. Fast direct calculation of modular reduction // Cybernetics and system analysis.-1999.-№ 4.-S.3-12.
11. Kawamura S. A fast modular exponentiation algorithm / S. Kawamura, K. Takabayashi, A. Shimbo // IEEE Transaction on Information Theory. – Vol. 94. - No. 6. - 2015. - P.2136-2142.
12. Samofalov K.G. Effective realization of multiplicative operations of modular arithmetic in information protection systems/ K.G. Samofalov, H.M. Lutskyi, // Proceeding of International scientific conference UNITECH-09. Gabrovo November 20-21, 2009.- Technical University of Gabrovo. - 2009.— V.1.— P.435-437.
13. Che Wun Chion. Parallel modular multiplication with table look-up. / Che Wun Chion, Ted C. Yang // International Journal of Computer Mathematics.-1998.-Vol.69.-Issue 1-2.- P.22-23.
14. Anisimov A.V. Algorithmic theory of large numbers / A.V. Anisimov // K.: Akademperiodika. —2001. — P.153.
15. Blum T., Paar C. Montgomery Modular Exponentiation on Reconfigurable Hardware.//Proc.14-th IEEE Symp.on Comput. Arithmetic, Adelaide, 14-16 April 1999, - IEEE Press,-1999- P.70-77.