Program implementation of large-integer modular multiplication based on Montgomery space

Natalia Kryvinska[1,†], Ihor Prots'ko[2,*,†] and Oleksandr Gryshchuk[3,†]

[1] *Comenius University Bratislava, Odbojarov, 10, Bratislava 25, Slovakia*
[2] *Lviv Polytechnic National University, S. Bandery, 12, Lviv, 79013, Ukraine*
[3] *LtdC "SoftServe", Sadova, 2d, Lviv, 79021, Ukraine*

Abstract

Modular multiplications and modular exponentiation over large integers is very computationally expensive. Many cryptosystems of smart networks use the Montgomery modular multiplication method, which reduces the latency of software and hardware implementations. The main directions of software development and outlines of the parts of Montgomery modular multiplication for the implementation are presented. The methods of the software class Montgomery Arithmetic over large integers for performing modular Montgomery multiplication are described. Improved method of reduction of operand transfer from Montgomery area that uses pre-computation. A comparison of the execution time of the Montgomery modular multiplication library functions over large integers was made. The developed software implementation of modular multiplication provides faster computation compared to functions of the MPIR, OpenSSL, Crypto++ libraries over large integers with 1024 and 2046 bits.

Keywords

Modular multiplication, Montgomery reduction, large numbers, speedup computation

1. Introduction

Problems of modern asymmetric cryptography and theoretical-numerical transformations widely use modular arithmetic. The most critical from the point of view of computational implementation is the operation of modular multiplication. Modular multiplication over numbers of a large bit size, which significantly exceeds the bit size of processors, is especially relevant. The increase in the number of digits leads to the complication of performing calculations on general-purpose computers, the slowing down of data exchange, and the possibility of unauthorized access to computer systems. Currently, to achieve a level of security acceptable for most applications, the necessary length of numerical data of modular multiplication can be 1024 - 4096 bits. This circumstance requires the development of effective software tools aimed at reducing the calculation time of computer arithmetic operations despite the increase in the bit rate of their operands [1].

Research on the efficient implementation of modular multiplication as a basic operation is under the constant attention of many developers of software libraries and hardware modules. After all, this basic operation is used in such areas of informatics as graphics and computer vision, communication networks, artificial intelligence, computer games and information security. It is these areas that use modular computing to model various processes, calculate activation functions in neural networks, cryptographic encryption and decryption, and generate pseudorandom data.

The scientific problem of speeding up modular reduction is relevant to the current stage of development of information and computer technologies. An analysis of the implementation of modular multiplication, taking into account many technical publications and textbooks, shows that

CIAW-2024: Computational Intelligence Application Workshop, October 10-12, 2024, Lviv, Ukraine
* Corresponding author.
† These authors contributed equally.
natalia.kryvinska @fm.uniba.sk (N. Kryvinska); ihor.o.protsko@lpnu.ua (I. Prots'ko); ocr@ukr.net (O. Gryshchuk)
0000-0003-3678-9229 (N. Kryvinska); 0000-0002-3514-9265 (I. Prots'ko);0000-0001-8744-4242 (O. Gryshchuk)

this basic operation is studied in detail and implemented in various computing software and hardware tools.

The article [2] proposes a method of accelerating the operation of modular multiplication of long numbers, which is important for cryptographic applications, by combining multiplications of sections of numbers with symmetric indices, as well as alternating cycles of adding sectional products with the same weight and Montgomery group reduction.

The methodology is relevant due to the use of matrix and vector modular methods of finding the remainder, modular multiplication and exponentiation, finding the inverse element based on the addition of the module [3]. The non-least positive form based modular multiplication method that combines naive multiplication and Karatsuba is applied in Montgomery modular multiplication [4]. In the article [5] new algorithm for the reduction of the Montgomery in the system of residues RNS, the main part of which is double matrix multiplication, is proposed. This makes it possible to remove some multiplication steps from the conventional algorithms, and thus the new algorithms are simpler and have higher regularity compared to the conventional ones.

In the paper [6], three modular reduction algorithms and one modular multiplication algorithm are implemented. The usage of Montgomery Multiplication and Montgomery Reduction can be done using Verilog as the description language in the Synopsys Design Compiler. In the article [7], a flexible and pipelined hardware implementation of modular Montgomery multiplication is proposed.

Thus, increasing the efficiency of modular multiplication is achieved in the directions of the development of algorithmic methods and hardware using modern information technologies for the implementation of computations.

The purpose of the study is the software implementation of modular multiplication based on Montgomery reduction to speed up computations over large numbers. The paper describes the software implementation of efficient modular algorithms, both Montgomery reduction and modular Montgomery multiplication. A comparison was made with functions for computing multi-bit modular multiplication of freely available software libraries.

2. Modular multiplication

Modular multiplication (multiplication of the large number T1 by the large number T2 modulo M) consists of calculating the product of T1 and T2 with a large modulus M, in form,

$$D = T_1 \cdot T_2 \bmod M , \tag{1}$$

it is believed that $2n-1 \le M < 2n$, $T1 < M$, $T2 < M$ and T1 , T2 , $M \in Z$, $D \in Zm$.

There are methods of accelerated multiplication, such as the Karatsuba method, the Fuhrer method, the Toom-Cook method, and FFT-based methods [8] in which the number of multiplications is less than n2.

In the case of calculation (1) with large integers T1 , T2 , M, use the property

$$T_1 \cdot T_2 \bmod M = [(T_1 \bmod M) \cdot (T_2 \bmod M)] \bmod M , \tag{2}$$

which allows you to perform calculations with smaller numbers. That is, if we multiply a large integer by another large integer, and the result of their multiplication is so large that it cannot be written in the computer, then using the property allows us to reduce the first two operands before starting the multiplication.

Thus, the basic operation of modular multiplication is the modular reduction of the numbers T1 and T2 concerning the module M. Effective implementation the modular reduction is the key to the high performance of numerical-theoretic algorithms for large numbers intended for information protection.

The classical calculation algorithm has no limit on the size of the numbers and can be easily adapted to a general-purpose division algorithm that gives both a quotient and a remainder. The

operation of integer division requires significant computing resources for its implementation. There are two ways of speeding up calculations: the use of recalculations and the replacement of processor division, which is inefficient for reduction, with processor multiplication. In order to avoid the operation of dividing multi-bit numbers when implementing modular reduction, several effective algorithms have been proposed, the main idea of which is to move from the operation of division to multiplication and shift. Algorithms are the most common among them Barrett Reduction, Mod without Division, Montgomery Reduction.

Barrett's modular reduction technology involves certain recalculations [9]. For example, there is a product P=T1· T2, module M. It is necessary to calculate P mod M. According to Barrett's method, the smallest q is searched for, for which: (P − q·M) < M in the form of a product:

$$q = P / M = P / 2^n \cdot w / 2^n , \quad w = 2^{2n} / M . \tag{3}$$

The numerical value of w depends only on the modulus M and can be calculated once. Division by power 2 is carried out by shifting. Barrett's reduction is implemented by two multiplication operations q · M and (P/2n) · (w/2n) with recalculation of w.

The Mod without division algorithm [10] is effective for calculating large numbers modulo. The condition of the algorithm is that the value of the input data must be greater than the value of the module. The algorithm also has pre-computation of the power of two modulo and certain predefined operations. In the algorithm, shifts are performed, while if the most significant bit is not zero, a correction is performed by setting the most significant bit to zero and adding to it a previously calculated value of the power of two modulo. The Mod without division algorithm for calculating the module operation ensures minimal memory access and is efficiently implemented at the hardware level.

Montgomery reduction [11] is used for modular reduction and is currently the most common algorithm. A unique property of the Montgomery reduction is that the algorithm does not compute the modulus directly, but instead, the modulus is multiplied by a constant. There are various modifications to the Montgomery modular reduction implementation.

The Montgomery reduction of the T number is defined as

$$T R^{-1} \bmod M , \tag{4}$$

where 0 < T < M-1; R > M; R, M ∈ Z .

To compute the Montgomery reduction (4), it is necessary to determine the value of R−1 that meets the condition R · R−1 mod M =1. To find R−1 the inverse modulo, you can use the extended Euclidean algorithm. Montgomery reduction is an algorithm with pre-computation that simultaneously computes the product by R-1 and reduces modulo M more quickly than the naive method.

For working with large numbers is common to write the Montgomery radix R = rn = 2k. This algorithm is based on scanning the bit of a large number T from the right (the least significant bit) to the left (the most significant bit). The algorithm Montgomery Reduction (Figure 1) of a number T for radix 2 what not require some pre-computation. In this algorithm, the resource-intensive division operation is replaced by simple shift operations by conversing the operands into the reduce number system domain before the operation and re-conversing the result after the operation.

Addition and subtraction in Montgomery form are the same as ordinary modular addition and subtraction because correspond to the distributive law:

$$(T_1 R^{-1} \bmod M) \pm (T_2 R^{-1} \bmod M) = (T_1 \pm T_1) R^{-1} \bmod M) , \tag{5}$$

where T1 , T2 ∈ Z .

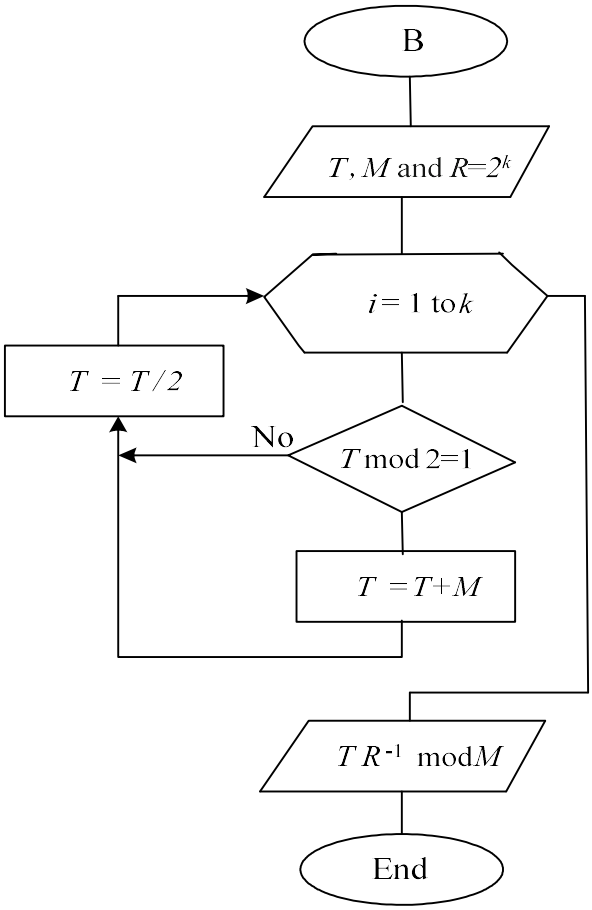

Figure 1: Algorithm of computation the Montgomery reduction for radix 2.

However, the reduction multiplication (T1R−1 mod M · T2R−1 mod M) is more complicated. The ordinary product of T1R−1 mod M and T2R−1 mod M does not represent the product of T1 and T2 because it has an additional factor of R-1.

3. Montgomery multiplication

Mathematician Peter L. Montgomery in 1985 proposed a method of computing modular multiplication, which does not require any division by M. This method is called Montgomery Modular Multiplication, which is the combined operation of multiplication and reduction.

The Montgomery reduction of the multiplication T1'· T2', where T1' = T1 R mod M and T2' = T2 R mod M, is

$$T_1^{'} \cdot T_2^{'} R^{-1} \bmod M \equiv (T_1 R \bmod M \cdot T_2 R \bmod M) / R^{-1} \bmod M = T_1 \cdot T_2 R \bmod M. \quad (6)$$

The inverse R-1 can be find using the extended Euclidean algorithm. Indeed, if gcd (M, R) =1, then the following integers will be found R -1 and M -1, that

$$R \cdot R^{=1} - M \cdot M^{-1} = 1. \quad (7)$$

Montgomery multiplication involves: first conversion of operands into Montgomery space, multiplication and then after result is re-conversed into Montgomery space.

Analysis of the Montgomery algorithm showed that its computational complexity of modular multiplication, which is performed without taking into account additional transformations, is close to the theoretical minimum. With a constant modulus, the computational complexity of the algorithm can be reduced. An analysis of numerous publications devoted to the development of the ideas of the Montgomery algorithm largely confirms this assessment.

There are various algorithms for bit-serial, digital-serial, and bit-parallel Montgomery multiplication by binary expansion fields to implement multi-precision modular multiplication. The modular multiplication method and various improvements to reduce latency for software implementation on devices are discussed in detail in the article [12]. In practice, Montgomery multiplication is the most efficient method when the common modulus is used and has a very regular structure, which speeds up the implementation [13].

The implementation of Montgomery multiplication can be applied both to a previously calculated product and as part of a combination of multiplication and reduction. The main advantage of separate execution of multiplication is that it is possible to apply existing methods of accelerated multiplication [8], such as the Karatsuba method, the Fuhrer method, the Toom-Cook method, and FFT-based methods.

The classic Montgomery modular multiplication algorithm (Figure 2) combines the execution time of multiplication and reduction, where one operation, the shift operation, is performed for the actual multiplication and reduction.

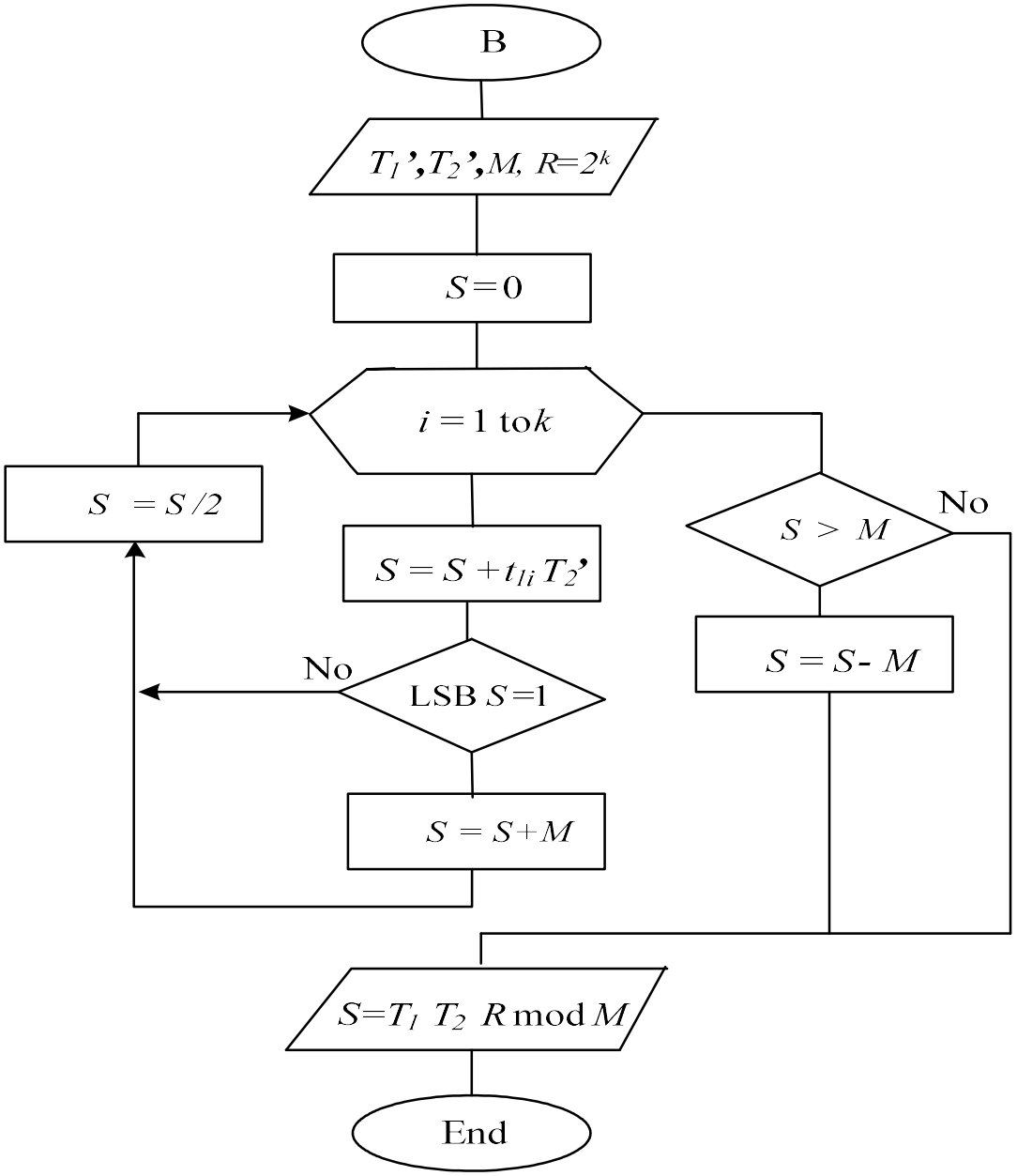

Figure 2: Algorithm of Computation the Montgomery modular multiplication, where T1' = (t1(k-1) … t1i ... t12 t11 t10), LSB S is last significant bit of S.

The form T1T2 R mod M means that, after we normally multiply two numbers in the Montgomery space, we need to reduce the result by multiplying it by R−1 and taking the modulo M.

4. The software implementations of Modular Multiplication

The operation of modular reduction is performed much more slowly compared to the operation of multiplying large numbers. This is due to the fact that obtaining the remainder requires significantly more computing resources, since it is impossible to use the built-in processor division operation for this purpose. Montgomery technology significantly accelerates modular multiplication, however, in the modern conditions of information technology development, there is a need to increase the efficiency of theoretical numerical algorithms, which are based on modular arithmetic operations.

Modern software libraries are used to perform calculations of modular multiplication of large numbers. The software implementation of the calculation of modular multiplication is included in the software libraries Crypto++, OpenSSL, MPIR, designed for working with large numbers. Software-implemented and modified algorithms of modular multiplication using Montgomery reduction require an experimental study of their advantages and disadvantages.

The increase in the bit rate of operands up to 4096 bits requires further development of efficient modular multiplication functions. The software implementation of modular multiplication in the C++ language allows you to perform modular multiplication of numbers with a variable bit using Montgomery reduction.

The basis of this program is the implementation of accelerated modular multiplication with a combination of multiplier addition and modular reduction according to the Montgomery algorithm using class-based pre-computations

```
class MontgomeryArithmetic
{
public:
   explicit MontgomeryArithmetic(const mpz_class& mod);
   mpz_class init(const mpz_class& x) const;
   void new_reduce(mpz_class& x) const;
private:
   const mpz_class mod_;
   mpz_class inv_;
   const size_t limbs_;
   const size_t bits_;
   mpz_class mip_2_;
   mpz_class mip_n_;
   };
```

The developed class MontgomeryArithmetic implements the Montgomery modular multiplication and reduction using the Multiple Precision Integers and Rationals library (MPIR) [14], which is a fork of the famous GNU Multiple Precision Arithmetic library (GMP). Accordingly, in the MPIR library, the data type mpz_t represents large numbers of arbitrary length, which are selected with the number from 256 to 4096 bits.

The constructor MontgomeryArithmetic(const mpz_class& mod) computes the modular inverse of the Montgomery reduction using the function mpz_sizeinbase(mpz_class(bits_).get_mpz_t(), 2) and initializes the other member variables

```
mpz_class power = mpz_sizeinbase(mpz_class(bits_).get_mpz_t(), 2);
for (size_t i = 0; i < power; i++)
{
   mpz_class expr = 2 - mod_ * inv_;
   mpz_tdiv_r_2exp(expr.get_mpz_t(), expr.get_mpz_t(), bits_);
   inv_ *= expr;
   mpz_tdiv_r_2exp(inv_.get_mpz_t(), inv_.get_mpz_t(), bits_);
}
```

The search for a solution is carried out by constructing successive approximations and is based on the principles of simple iteration (we start with inv_=1,as the inverse of m modulo 21). This method calls the mpz_tdiv_r_2exp() function provided by MPIR, which returns the least significant bits of a number, i.e., calculates the remainder of division by 2k. This algorithm uses only the shift, subtraction, and multiplication of multi-bit numbers on each iteration.

Implementation of Montgomery multiplication (Fig. 3) for the computed product of two factors a and b uses methods mpz_class init(const mpz_class& x) const and void new_reduce(mpz_class& x) const converting them to Montgomery space and their product a_inv · b_inv out Res of Montgomery form.

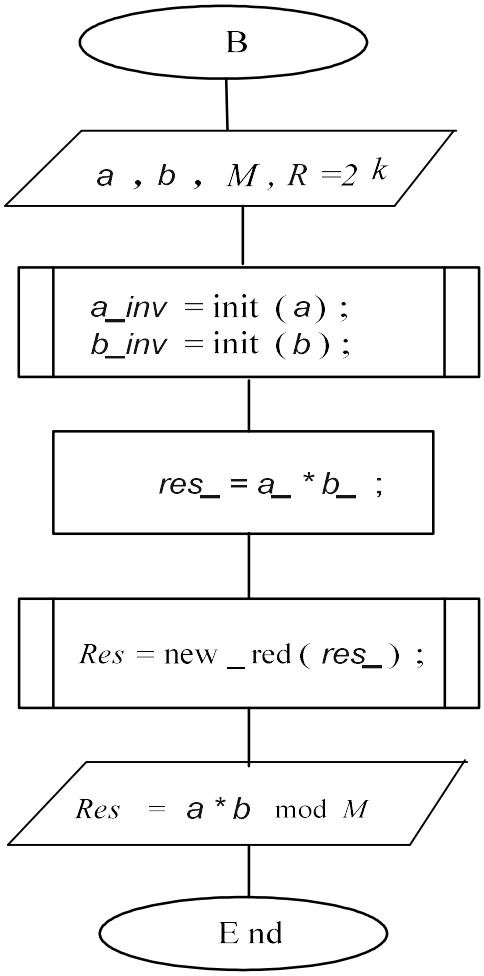

Figure 3: The scheme for computation the Montgomery modular multiplication.

Before use at the stage of the multiplication operation (Fig. 3), you need to convert all numbers to the Montgomery area. The method init mpz_class init(const mpz_class& x) const converts the number x to the Montgomery space. A fast conversion algorithm implemented for multi-bit numbers is described in [15].

The product of two numbers a_inv and b_inv converted to the Montgomery space uses the usual multiplication provided by the MPIR library, optimized with AVX2 SIMD instructions, and is performed by sequential execution of instructions

```
a_inv = mod_arithmetic.init(a);
b_inv = mod_arithmetic.init(b);
res_ = a_ * b_;
```

and then the Montgomery reduction is performed

```
mod_arithmetic_.new_reduce(res_);
```

This is one of the most performance-critical techniques. Method new_reduce(res_) with the product argument res_ for which the Montgomery reduction is computed. This method is a modernization of the mod_arithmetic_.reduce_1(res_) function of the MPIR library, which performs the basic algorithm based on the Montgomery reduction. As a result of the analysis of the function code, it was found that in reduce_1(res_) arithmetic transfers are stored in the variable tp and at the end are added to the result using the mpn_addmul_1() function. As a result of the development of the new_reduce(res_) function, an improvement has been achieved by taking into account transfers, which are immediately added to the result. The mpn_addmul_1() function essentially consists of many calls, so this simplification based on the class MontgomeryArithmetic provided faster computation for large-bit data.

The developed Montgomery modular multiplication algorithm is faster than the usual modular multiplication, which performs division by m. However, calculating the modular inverses of R-1, m-1 and converting the numbers to the Montgomery region and vice versa are time-consuming operations. The importance of speeding up the software computation of modular multiplication leads to new software algorithmic solutions and their implementations. We will test the developed software modular multiplication and compare it with similar functions from the Crypto++, OpenSSL, MPIR software libraries designed for working with large numbers.

5. Experiments and discuss the implementation of Montgomery modular multiplication

To test the effectiveness of the proposed method of accelerated modular multiplication with a combination of multiplier addition and modular reduction according to the Montgomery algorithm using pre-computations 5 different library functions were used. The library contains a set of available primitives for theoretical and numerical operations, such as generation and verification of prime numbers, arithmetic over a finite field, operations on polynomials.

The production-grade OpenSSL library contains a set of tools for cryptography that implements the Secure Sockets Layer (SSL v2/v3) and Transport Layer Security (TLS v1) network protocols, as well as the corresponding cryptography standards [16]. The OpenSSL library includes two functions to calculate the modular multiplication based on Montgomery space:

BN_mod_mul_montgomery(res_, a_, b_, mont_ctx_, ctx_);

BN_from_montgomery(res_, res_, mont_ctx_, ctx_);

Crypto++ is a library implemented in the C++ language, which is optimized for various platforms and allows performing high-performance cryptographic operations. The library supports not only basic cryptographic primitives, but also more advanced functions. [17]. The Crypto++ library contains a function to compute modular multiplication based on Montgomery space:

mod_arithmetic_.Multiply(a_, b_);

A highly optimized modification of the well-known GMP or GNU Multiple Precision Arithmetic Library the MPIR library, which is written in C and the assembler, is used to implement the algorithm for computing the modular multiplication. Accordingly, in the MPIR library, the mpz_t data type represents long numbers that are selected for the number of binary digits from 256 to 2048 bits for testing. For the implementation of the developed algorithm the MPIR library functions mpz_init_set(mul, base), mpz_sizeinbase(exp, 2), mpz_tstbit(exp, i), mpz_mul(r, r, mul) with long binary digit data are used. The MPIR library [14] offers a few low-level implementations of multiplication, which can be further optimized for specific use cases. There is no separate modular multiplication function in the MPIR library. Instead, you need to combine ordinary multiplication and Montgomery reduction. There is no separate modular multiplication function in the MPIR library. Instead, you need to combine ordinary multiplication and Montgomery reduction. Thre are three reduction functions in the MPIR library:

mpn_redc_1() function is used for numbers less than 2048 bits;

mpn_redc_2() function is used for numbers less than 4096 bits;

mpn_redc_n() function is used for larger numbers.

Experimental testing of 5 library functions marked as crypto++, openssl, mpir_redc_1, mpir_redc_2, mpir_redc_n and developed mpir_redc_new() using MPIR library to implementation, is presented in Figure 4. A computer system with a multi-core microprocessor with shared memory in 64-bit Windows was used for numerical experiments.Testing was performed on computer systems with processors an Intel Core i9-13900K (24 cores, 32 threads, 3.0GHz). The results are presented in Figure 4, which contains the values of average execution time (ns nanoseconds) of computing 20 executions of the modular multiplication for pseudo-random data a, b, mod for 1024 bits of trials 100000 and 2048 bits of trials 50000.

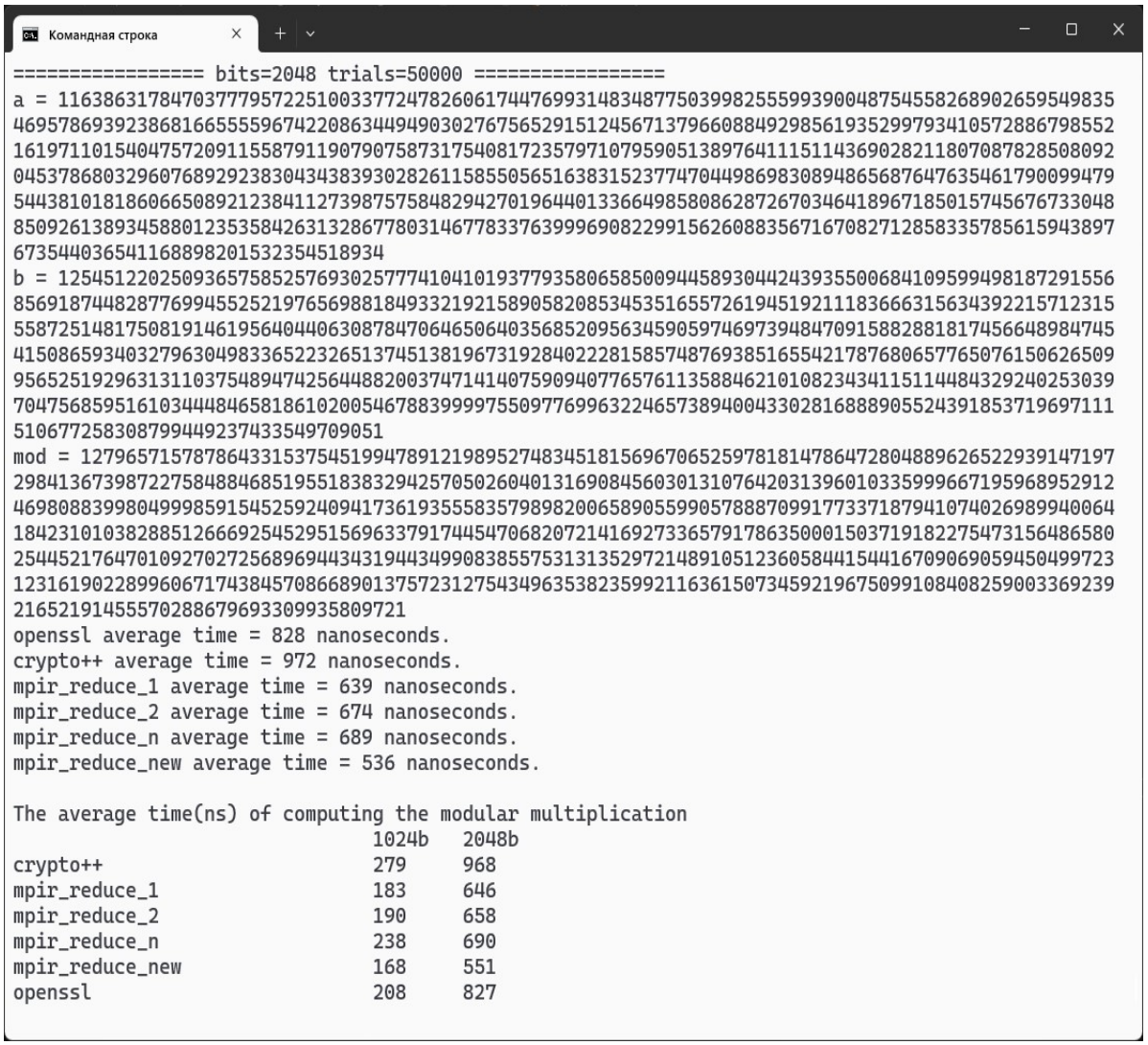

Figure 4: The result of testing the functions of computing the modular multiplication on a computer system with an Intel Core i9-13900K processor.

Analysis of the execution time of modular multiplication revealed that the execution time of the developed function mpir_redc_new() is the smallest than the execution time of the library functions. Montgomery modular multiplication mpir_redc_new() is the fastest in computing a reasonably long series of modular reductions, for instance in computing exponential functions of large numbers [18, 19].

Among 5 other functions, the best average time for computing the modular product (Fig. 4) is in the openssl library function. In relation to openssl, the developed function mpir_redc_new() has a better computation time by 24% for operands of size 1024 bits and by 50% for operands of size 2048 bits. Thus, with the increase in the number of operands, the average time for computing the modular product is reduced based on the use of the mpir_redc_new() function.

In the process of solving many number-theoretic transformations, and especially in the problems of cryptographic protection, the use of efficient computation of modular multiplication is the most used operation. An important component of these calculations is the implementation time of the modular multiplication, which has a significant impact on the performance of the computation. Application of modular Montgomery multiplication requires additional operations performed before and after the actual multiplication. In practice, modular multiplication using the Montgomery algorithm is suitable in performing modular exponentiation in number-theoretic algorithms using multi-digit numbers. For example, the obtained results of the implementation of the modular exponentiation function with precomputation of the residues for a fixed basis and the use of the developed Montgomery modular multiplication are the best among the available modular exponentiation functions from the Crypto++, OpenSSL, and MPIR libraries for large numbers with the bit size of more than 1Kbit [20]. Therefore, based on the developed software, the further implementation of the computation of modular multiplication using promising computing technologies will provide the possibility of effective solutions to applied problems.

6. Conclusions

The work compares and analyzes the use of the developed software implementation of the Montgomery Arithmetic class for calculating modular multiplication. The constituent parts of the modular Montgomery multiplication are outlined and the main features of the implementation of the methods of the Montgomery Arithmetic class are given. The developed function of modular multiplication and 5 functions of known software libraries were tested. As a result, the developed function provides faster computations of Montgomery modular multiplication compared to using other modular multiplication functions. We have shown that the developed Montgomery modular multiplication in general-purpose computers speeds up the computations by an average of 1.5 times compared to functions of modular multiplication for 2k bits numbers taken from famous software libraries.

The scientific novelty of the obtained results lies in the implementation of the analysis and software improvement of the Montgomery modular multiplication calculation, which obtained the best time characteristics among the well-known functions of the Crypto++, OpenSSL and MPIR libraries for large numbers over 1K bits.

The practical significance of the work lies in the fact that the obtained results can be successfully applied for efficient computation of number-theoretic transforms, for modern asymmetric cryptography and other computational applications. An especially effective use of the developed software implementation of the Montgomery Arithmetic class for calculating fixed-base modular exponentiation, which application is important in Diffie-Hellman key agreement and elliptic curve digital signature algorithm verification.

Prospects for further research are the development of a program function of Montgomery's modular multiplication and its implementation for large integers of size 4096 bits.

References

[1] V. Zadiraka, A. Tereshchenko, Computer arithmetic of multi-bit numbers in serial and parallel computing models, Scientific opinion, Kyiv. 2021. 152 p.

[2] O. Markovskyi, A. Jalil, Method of accelerated modular multiplication for efficient implementation of public key cryptographic protection mechanisms, Interdepartmental scientific and technical collection "Adaptive systems of automatic control", 1.44 (2024): 142-152. doi:10.20535/1560-8956.44.2024.302429.

[3] M. Kasianchuk, M, Karpinski, S. Kazmirchuk, Methodology of processing multi-digit numbers in asymmetric cryptosystems, Ukrainian Information Security Research Journal, 21.2 (2019): 65-73. doi: 10.18372/2410-7840.21.13764.

[4] J. Ding, S. Li, A low-latency and low-cost Montgomery modular multiplier based on NLP multiplication. IEEE Trans. Circuits Syst. II Express Briefs, 67(2019): 1319–1323. [CrossRef]

[5] doi: 10.1109/TCSII.2019.2932328.
[6] S. Kawamura, Y. Komano, H. Shimizu, T. Yonemura, RNS Montgomery reduction algorithms using quadratic residuosity. Journal of Cryptographic Engineering, 9 (2019): 313–331. doi:10.1007/s13389-018-0195-8.
[7] M. J. Ferrao, K. Kumar, N. Megha, Implementation of Modular Reduction and Modular Multiplication Algorithms, IOSR Journal of VLSI and Signal Processing (IOSR-JVSP), 8.6.I (2018): 34-38. doi: 10.9790/4200-0806013438.
[8] Z. Zhang,P. Zhang, A Scalable Montgomery Modular Multiplication Architecture with Low
[9] Area-Time Product Based on Redundant Binary Representation, Electronics 11.3712 (2022). doi:10.3390/electronics11223712.
[10] A. Tereshchenko, V. Zadiraka, Implementation of multi-bit multiplication operation based on discrete cosine and sine transformations, Cybernetics and Computer Technologies, 4 (2021): 61–79. doi:10.34229/2707-451X.21.4.7.
[11] Z. Cao , Z. Chen, R. Wei, L. Liu, Run-based Modular Reduction Method, International Journal of Network Security, 22.2 (2020): 331-336. doi: 10.6633/IJNS.202003 22(2).17.
[12] M. A. Will, R. K. L. Ko, Computing Mod Without Mod, in: Proceedings of 4th International Symposium Security in Computing and Communications, SSCC 2016, Jiapur, India, 21-24 September 2016, pp. 3-17. doi: 10.1007/978-981-10-2738-3.
[13] J.-C. Bajard, J. Eynard, N. Merkiche, Montgomery reduction within the context of residue number system arithmetic. Special Issue on Montgomery Arithmetic, Journal of Cryptographic Engineering, 8 (2018): 189–200. doi:10.1007/s13389-017-0154-9.
[14] J. W. Bos, P. L. Montgomery, Topics in Computational Number Theory Inspired by Peter L. Montgomery, in: J. W. Bos, A. K. Lenstra (Eds.), Montgomery Arithmetic from a Software Perspective. October 2017, 276 p. URL: www.cambridge.org/9781107109353
[15] S. Srinitha, S. Niveda, S. Rangeetha, Kiruthika, A High Speed Montgomery Multiplier used in Security Applications. in: Proceedings of the 2021 3rd International Conference on Signal Processing and Communication (ICPSC), Coimbatore, India, 13–14 May 2021, pp. 299–303.
[16] MPIR: Multiple Precision Integers and Rationals. 2021. URL: http://mpir.org/.
[17] Montgomery Multiplication, 2022. URL: https://cp-algorithms.com/algebra /montgomery_multiplication.html#implementation
[18] OpenSSL. Cryptography and SSL/TLS Toolkit, 2021. URL: http://www.openssl.org/
[19] Crypto++, 2021. URL: https://www.cryptopp.com/docs/ref/class_modular_arithmetic. html
[20] I. Prots'ko, A. Gryshchuk, V. Riznyk, Efficient Multithreading Computation of Modular Exponentiation with Pre-computation of Residues for Fixed-base in: Proceedings of Sixth International Workshop on Computer Modeling and Intelligent Systems (CMIS 2023), Zaporizhzhia, Ukraine, 3 May 2023. pp. 224-234. doi:10.32782/cmis/3392-19.
[21] I. Prots'ko, N. Kryvinska, O. Gryshchuk, The Runtime Analysis of Computation of Modular Exponentiation, Radio Electronics, Computer Science, Control 3 (2021). doi: 10.15588/1607-3274-2021-3-4.
[22] I. Prots'ko, A. Gryshchuk, Implementing Montgomery Multiplication to Speed-up the Computation of Modular Exponentiation of Multi-bit Numbers, Cybernetics and Systems Analysis 60.5 (2024). doi: 10.1007/s10559-024-00720-4.