UDC 004.421

THE MODULAR EXPONENTIATION WITH PRECOMPUTATION OF REDUSED SET OF RESIDUES FOR FIXED-BASE

Prots'ko I. – Dr. Sc., Associate Professor, Department of Automated Control Systems, Lviv National Polytechnic University, Lviv, Ukraine.

Gryshchuk O. – Software Developer, LtdC "SoftServe", Lviv, Ukraine.

ABSTRACT

Context. Modular exponentiation is an important operation in many applications that requires a large number of calculations Fast computations of the modular exponentiation are extremely necessary for efficient computations in theoretical-numerical transforms, for provide high crypto capability of information data and in many other applications.

Objective – the runtime analysis of software functions for computation of modular exponentiation of the developed program that uses the precomputation of redused set of residuals for fixed-base.

Method. Modular exponentiation is implemented using of the development of the right-to-left binary exponentiation method for a fixed basis with precomputation of redused set of residuals. To efficient compute the modular exponentiation over big numbers, the property of a periodicity for the sequence of residuals of a fixed base with exponents equal to an integer power of two is used.

Results. Comparison of the runtimes of five variants of functions for computing the modular exponentiation is performed. In the algorithm with precomputation of redused set of residuals for fixed-base provide faster computation of modular exponentiation for values larger than 1K binary digits compared to the functions of modular exponentiation of the MPIR and Crypto++ libraries. The MPIR library with an integer data type with the number of binary digits from 256 to 2048 bits is used to develop an algorithm for computing the modular exponentiation.

Conclusions. In the work has been considered and analysed the developed software implementation of the computation of modular exponentiation on universal computer systems. One of the ways to implement the speedup of computing modular exponentiation is developing algorithms that can use the precomputation of redused set of residuals for fixed-base. The software implementation of modular exponentiation with increasing from 1K the number of binary digit of exponent shows an improvement of computation time with comparison with the functions of modular exponentiation of the MPIR and Crypto++ libraries.

KEYWORDS: modular exponentiation, big numbers, exponentiation algorithm, fixed-base exponentiation, residual set.

ABBREVIATIONS

GMP is a GNU Multiple Precision Arithmetic library;

ME is a modular exponentiation;

MPIR is a Multiple Precision Integers and Rationals library.

NOMENCLATURE

A is a base integer value;
b is a binary representation of the exponent x;
Base is an identifier of a base;
e_i is a part of binary representation x;
exp is an identifier of an exponent;
$ind_R A$ is an index of residue;
k is a bitlength of a value x
m is a number of the parts of binary representation x;
mod is an identifier of modulo;
N is an integer value of modulo;
P is an odd prime;
q is a positive integer;
r is a bitlength of a part of binary representation x;
$r_{i,}$ is a residue;
R is a primitive root;
T' is a period of the residues;
u is an offset of a period of the residues;
x is an integer value of an exponent;
x_i is an bit value of an exponent;
y is an integer value of modular exponentiation;
$\varphi(N)$ is the Euler's function.

INTRODUCTION

The task of developing an effective computational algorithm for ME for big numbers is relevant enough to solve the problems of modern asymmetric cryptography, for efficient computation of number-theoretic transforms, digital signatures and other applications [1].

The object of study is the process of analysis the developed software implementation of the computation of ME. To efficient compute the ME over large numbers the property of the periodicity of the sequence of residuals for the exponent of the fixed-basis equal to the integer power of two are used.

The subject of study is the computation of ME based on the use the bits of the binary exponent with the precomputation of redused set of residuals for fixed-base.

The purpose of the work is to increase the speed of computation of ME based of computer systems in comparison with the function of ME of the MPIR and Crypto ++ libraries.

1 PROBLEM STATEMENT

The ME and the discrete logarithm are important operations that require a large number of calculations. The problem of discrete logarithm [1] is formulated so that for known integers A, N, y find the integer x, $(A, N) = 1$; $A, N, y, x \in Z$) such that

$$x = \log A^y, (0 \le x \le N-1). \qquad (1)$$

DOI 10.15588/1607-3274-2022-1-7

The number $x > 0$ is called the discrete logarithm of the number y based on A and modulo N according to formula (1).

The solution of the discrete logarithm problem can be the solution of the equation

$$A^x \bmod N = y. \quad (2)$$

That is, determining the number x, which is the solution of equation (2), we find the discrete logarithm. Thus, the problem of the discrete logarithm is reduced to the computation of the ME in the form (2). The discrete logarithm is considered to be a unidirectional function (1), because it is difficult to calculate it in a relatively acceptable time, for example, to break the cryptographic code. The development of an efficient computational algorithm for integer power of a modulo number for large numbers is relevant for solving problems of modern asymmetric cryptography, for the effective implementation of theoretical and numerical transformations and other applied problems. Therefore, it is very important to build algorithmic schemes that provide fast calculation of the ME.

2 REVIEW OF THE LITERATURE

Many effective methods of ME have been proposed [2, 3]. Among them are called: right-to-left k-ary exponentiation, left-to-right k-ary exponentiation, sliding window exponentiation, Montgomery ladder, simultaneous multiple exponentiation and their modifications. Considerable attention is paid to their software or hardware implementation [4–6] aimed at the effective definition of the discrete logarithm x.

One of the ways to accelerate the computation of modular elevation to the power is to parallelize calculations using modern technologies in universal computer systems [4–6].

Mathematical software libraries are used to implement the computation of ME. For example, the Pari/GP software library [7] contains a large set of programs for efficient computations of mathematical functions. The Pari/GP library also includes computation of the ME function for long numbers and other special numbers. A highly optimized modification of the well-known GMP or GNU Multiple Precision Arithmetic Library the MPIR library [8] contains the function of the realization the computation of ME. The library of cryptographic algorithms and schemes Crypto ++ is implemented in C ++ and fully supports 32 and 64-bit architectures of many operating systems and platforms [9]. 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.

3 MATERIALS AND METHODS

The general-purpose exponentiation algorithms referred to as repeated square-and-multiply algorithms. The papers of Knuth [10], Bach and Shallit [11] describe the right-to-left binary exponentiation method. Cohen [12] provides a more comprehensive treatment of the right- to-left and left-to-right binary methods along with their generalizations to the k-ary method.

The central idea to calculate A^x mod N is to use the binary representation of the exponent x

$$x = (x_{(k-1)} x_{(k-2)} \dots x_2\, x_1 x_0)_b\,,$$

$$x = \sum_{i=0}^{k-1} 2^i x_i \quad \text{and } x_i \in \{0,1\}. \quad (3)$$

We write the exponent x as a set of m parts that are equal in binary length r. That is, the binary representation of the value of x consists of m, the bit length of each of them is equal to $r=k/m$. Then the binary representation of the exponent $\boldsymbol{x}$ will be

$$\begin{aligned} x &= (e_{(m-1)} \dots e_2\, e_1)_b = \\ &(x_{mr-1} \dots x_{(m-1)(r+2)}\, x_{(m-1)(r+1)} x_{(m-1)r}) \dots \\ &(x_{2r-1} \dots x_{r+2}\, x_{r+1} x_r)(x_{r-1} \dots x_2\, x_1 x_0)_b \end{aligned} \quad (4)$$

In this case, the x value will be

$$x = \sum_{i=0}^{k-1} 2^{i(k/m)} e_i\,. \quad (5)$$

Accordingly (4, 5), the computation of the ME takes the form

$$\begin{aligned} & y = A^x \bmod N = A^{(2^{(m-1)r} e_{(m-1)} \cdot \ldots \cdot 2^{2r} e_2 \cdot 2^r e_1 \cdot 2^0 e_0)_b} \bmod N = \\ & = (A^{2^{(m-1)r} e_{(m-1)}} \bmod N * A^{2^{(m-2)r} e_{(m-2)}} \bmod N * \ldots \\ & * A^{2^{2r} e_2} \bmod N * A^{2^r e_1} \bmod N * A^{2^0 e_0} \bmod N) \bmod N = \\ & = ((A^{e_{(m-1)}} \bmod N)^{2^{(m-1)r}} * (A^{e_{(m-2)}} \bmod N)^{2^{(m-2)r}} * \ldots \\ & * (A^{e_2} \bmod N)^{2^{2r}} * (A^{e_1} \bmod N)^{2^r} * (A^{e_0} \bmod N)^{2^0}) \bmod N. \end{aligned} \quad (6)$$

There are three types of exponentiation algorithms A^x mod N [13], which include:

1) basic techniques for exponentiation;
2) fixed-exponent x exponentiation algorithms;
3) fixed-base A exponentiation algorithms.

A fixed element of a group (generally z/qz) is repeatedly raised to many different powers in several cryptographic systems. A popular application of fixed-base exponentation is in elliptic curve cryptography, for instance for Diffie-Hellman key agreement and elliptic curve digital signature algorithm verification. Therefore, many research works have been focused on a fixed base of ME [14–16].

Compute, respectively (6), the value modulo N for a simple fixed-base A with exponents $x = 2^i = 1,2,4,8,16\ldots$,

DOI 10.15588/1607-3274-2022-1-7

(i =0,1,2,…, r–1). Let A and N be relatively prime positive integers (A, N) = 1 and denote the least positive integer $x = \exp_N A$, in case

$$A^x \bmod N \equiv 1 \ . \quad (7)$$

Accordance of the theorem [17], if A and N relatively prime (A, N) = 1, positive integer $\mathbf{x}$ is solution of the congruence (7) if and only if

$$x = q \cdot \exp_N A, \quad (8)$$

Accordance the Euler's theorem, if A and N relatively prime (A, N) = 1, that $A^{\varphi(N)} \equiv 1$ (mod N). Consequently, we can do conclusion

$$\varphi(N) = q \cdot \exp_N A, \quad (9)$$

In case q=1, then $\varphi(N) = \exp_N R$, where R is the positive integer is called a primitive root modulo N. However the positive integer of modulo N, possesses a primitive root R if only if N=2, 4, P^k or $2P^k$, k is positive integer. The primitive root for modulo $N = P_1^{k1} P_2^{k2} \ldots P_m^{km}$ does not have, except крім if $\varphi(P_1^{k1})$, $\varphi(P_2^{k2})$,…, $\varphi(P_m^{ml})$ are relatively prime.

Thus, calculating $(R)^i$ mod N (i = 0,1, 2 ,,… N–1), we form a sequence of residuals ($r_0, r_1, r_2, \ldots, r_i, \ldots, r_{N-1}$), which periodically repeated for $x > (N–1)$ exponents. For all values of $A \in Zp$, the sequence A^i mod P is cyclic for a non-primitive element.

The unique integer x with $1 \le x \le \varphi(N)$ and R^x mod $N \equiv A$ is called $\mathrm{ind}_R A$ index (or discrete logarithm) of A to base R modulo N. The properties of indeces, where a, b, k a positive integer and (a, N)=1, (b, N)=1, are

1) ind_R 1 mod $\varphi(N) \equiv 0$,
2) $\mathrm{ind}_R (ab)$ mod $\varphi(N) \equiv \mathrm{ind}_R (a) + \mathrm{ind}_R (b)$ mod $\varphi(N)$,
3) $\mathrm{ind}_R (a^k)$ mod $\varphi(N) \equiv k\, \mathrm{ind}_R (a)$ mod $\varphi(N)$.

For example, for a primitive element R = 7, the sequence of residual values $r_i = (7^i)$ mod 11,

$$(r_0, r_1, r_2, r_3, r_4, r_5, r_6, r_7, r_8, r_9) = (1,7,5,2,3,10,4,6,9,8).$$

The maximum period of repetitions is equal to $\exp_{11} 7$ = 10, because 7^{10} mod 11=1, i=0,1,2,…,9. Then the sequence of indeces is equal (0, 1, 2, 3, 4, 5, 6, 7, 8, 9) = = ($\mathrm{ind}_7 1$, $\mathrm{ind}_7 7$, $\mathrm{ind}_7 5$, $\mathrm{ind}_7 2$, $\mathrm{ind}_7 3$, $\mathrm{ind}_7 10$, $\mathrm{ind}_7 4$, $\mathrm{ind}_7 6$, $\mathrm{ind}_7 9$, $\mathrm{ind}_7 8$).

Accordingly of the property of indeces

1) ind_7 1= 0,
2) ind_7 6 = ind_7 (2*3)= ind_7 (2)+ +ind_7 3 mod 10= 3+4=7;
3) ind_7 9 = ind_7 (3^2) = 2 * ind_7 3 mod 10 =2*4=8.

In the case of calculating (7^x) mod 11 with index x = 32, the index will be equal to (32 mod ind117) = 2, and accordingly $\mathrm{ind}_7 5$. In the case of determining ($7^{2\,\wedge\,6}$) mod 11, we find the number of the residue in the sequence with the index $\mathrm{ind}_7 3$, which is equal to (2^6) mod 10 = 4. After all, the value of the ME for 2 elements in the sequence of residual values r_4 = 3 = ($7^{2\,\wedge\,6}$) mod 11.

For computations according to formula (6), we determine the residuals for exponents 2^i, (i =2,3,4,…). As a result of computations r_i = ($7^{2\,\wedge\,i}$) mod 11, (i =2,3,4,…) we obtain the values of the residuals given in Table 1.

Table 1 – Periodic repetition of residual values 7 ^ 2^i mod 11

7 ^ 2^i	7^0	7^1	7^2	7^4	7^8	7^{16}	7^{32}
T'=4 (**1**,**7**,5,3,9, 4)	1	7	**5**	**3**	**9**	**4**	5

7^{64}	7^{128}	7^{256}	7^{512}	7^{1024}	7^{2048}	7^{4096}
3	9	4	5	3	9	4

That is, in the process of computing (7^2^i) mod 11, starting with the exponent 2^1 = 2, we obtain periodic repetition of the values of the residuals $r_0, r_1, \mathbf{r_2}, \mathbf{r_4}, \mathbf{r_8}, \mathbf{r_6}, r_2, r_4, r_8, r_6, \ldots$ with period T'= 4 and offset u = 0, because 2^0 = 1.

The value of T' is found by the condition

$$A\,\hat{}\,2^i \bmod N \equiv A\,\hat{}\,2^{(i+T'+u)} \bmod N, \, i > u \ . \quad (10)$$

Therefore, for a fixed-basis A of the ME of the computation of formula (6), which is equal to the product of the residuals of the exponent (A^2^i) mod N, (i = 2,3,4,…), you can speed up the process of computing the ME by precomputing the sequence of residuals what repetitions with the period T' after the offset u.

4 EXPERIMENTS

Mathematical software libraries are used to implement the computation of the ME. For example, the Pari/GP software library [7] contains a large set of programs for fast computations of mathematical functions. The Pari/GP library also includes computations of the Mod (a, n) ^ m function for multi-bit numbers, while using a small amount of memory in the process of performing computations. To work with numbers for modulo, the library uses a separate type *t_INTMOD*. Its feature is to represent the number in a special form (Montgomery reduction), which simplifies the computation of division by modulo. The Pari / GP library can be used in Linux or Mingw operating systems.

The library of cryptographic algorithms and schemes Crypto ++ is implemented in C ++ and fully supports 32 and 64-bit architectures of many operating systems and platforms [9]. 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. Each of the Crypto ++ library primitives includes a function set.

The function mod_arithmetic.Exponentiate (*base_crypto*, *exp_crypto*) raising the number to the power by modulo. The result of the function is written to the variable *actual_result_crypto*, and the computation time is fixed and averaged with the output value "*crypto++ average time*" in nanoseconds.

Compared to the Pari/GP library, the well-known MPIR library [8] is easier in use and can be compiled in

DOI 10.15588/1607-3274-2022-1-7

Windows easily. Therefore, to implement the algorithm for computing the integer power of a number modulo, we used the MPIR library, which is written in C and assembler, and provides the ability to compile its functions in Visual Studio C ++. Accordingly, in the MPIR library, the data type *mpz_t* represents large numbers of arbitrary length, which are selected for the power *exp* of the number *base* and the *mod* module with the number of bits from 256 to 2048 bits for testing.

The function mpz_powm (*expected_result*, *base*, *exp*, *mod*) performs raising the number to the power by modulo from the MPIR library, implementing the algorithm of the sliding window ("Sliding Window") with the use of Montgomery multiplication [14]. The result of the function is written to the variable *expected_result*, and the computation time is fixed and averaged with the output value "*mpz_powm average time"* in nanoseconds.

The function period_mod_exp (*remainders_data, exp*) has been developed, which performs the basic iterative algorithm "Right-to-left binary exponentiation" [13]. To implement the algorithm, the library functions mpz_init_set (*mul, base*), mpz_sizeinbase (exp*,* 2), mpz_tstbit (*exp, i*), mpz_mul (*r, r, mul*) from the MPIR library are used, the parameters of which are multi-bit data up to 2048 bits. The algorithm is executed without dividing the exponent into parts, according to formulas (3–6) with $m = 1$, in one main stream. The function period_mod_exp () computes products modulo using precomputed residuals. The organization of the computation of the ME is performed respectively (11) and the scheme for computing A^x mod N in Fig. 1.

$$\begin{aligned} y &= A^{x_{(k-1)}x_{(k-2)}\ldots x_2 x_1 x_0} \bmod N = \\ &= (A^{2^{k-1}} \bmod N * A^{2^{k-2}} \bmod N * \ldots \\ &* A^{2^2} \bmod N * A^{2^1} \bmod N * A^{2^0} \bmod N) \bmod N; \end{aligned} \qquad (11)$$

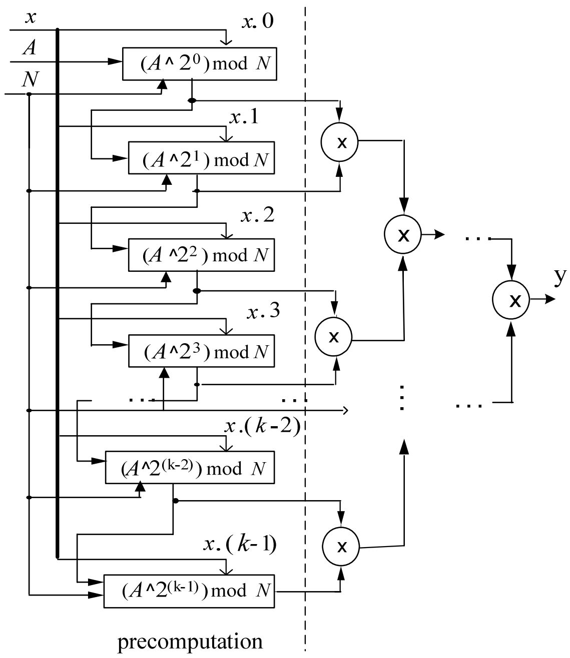

Figure 1 – The scheme for computing A^x mod N

DOI 10.15588/1607-3274-2022-1-7

In the software implementation, the function period_mod_exp (*remainders_data, exp*) computes the products modulo (11) over the precomputed values of the residuals (A ^ 2^i) mod N, which are read using the function get_remainder (*const RemaindersData & data, size_t power*). In the cycle of the function mpz_tstbit (exp*, i*) binary bits *x.i* of exponent *exp* are analyzed to determine to perform or not a multiplication operation modulo (Fig. 2). The computation of the value of the ME ends by writing the result in the variable *period_mod_exp_result*.

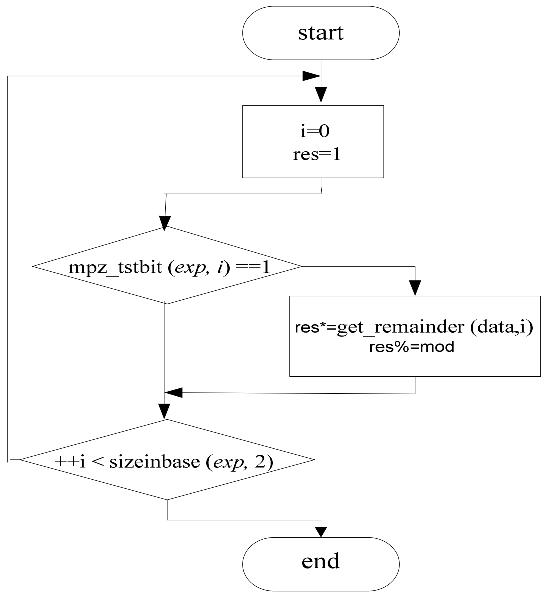

Figure 2 – The chart of the algorithm for determining to perform or not a multiplication under modulo in the function period_mod_exp() to compute the value of the ME

The precomputation includes finding the sequence of residuals for fixed numbers *Base* and *mod* for exp = 2^i (i = 0,1,2,…) and analysis of periodicity. In the program for computing the sequence of residuals is performed by the function find_remainders (*const mpz_class & base, const mpz_class & mod, size_t max_exp_bits*), which contains the function bool find_period (*const std::vector<mpz_class> & remainders*) to set the indication of finding the period. The function update_remainders (*RemaindersData & data*), shortens the length of the sequence of residuals to the end of the first periodicity. This function writes the offset *period_offset* beginning of the period and the length of the period *period_size* in the corresponding fields of the structure RemaindersData {*mpz_class base; mpz_class mod; std::vector <mpz_class> remainders; size_t period_offset; size_t period_size;*} also.

The precomputation have been made in a separate function find_remainders () to optimize multiple residual searches (A^2^i) mod N. The peculiarity of the large values of *Base*, *mod* and *Exp* is also taken into account for which the residuals must be calculated, in case when the value of the period T' is many orders of magnitude greater than the number of bits of the Exp exponent.

5 RESULTS

To compare the computation efficiency of the developed ME function for a fixed basis with precomputation, two ME functions implemented from the Crypto ++ 8.2 and MPIR libraries are used. The comparison is performed with previously developed functions Single (), which performs in one main thread without taking into account the periodicity, and Parallel (), which performs in using two threads [18] computation of the ME.

Numerical experiments were carried out on a computer system with a multi-core microprocessor with shared memory in a 64-bit Windows. Testing was performed on computer systems with processors an Intel Core i9-10980XE (18 cores, 36 threads, 3.0GHz) and AMD Ryzen 3600(6 cores, 12 threads, 3.0GHz).

The average time data of the test with prime numbers *P* for *Base* and *mod*, that are

Base = *P*=131071,

Base = $P_1*P_2*P_3$=131080=8*5*3277,

mod =*P* =6700417, mod= P^2=$(5\ 039)^2$=25391521,

mod =P^3=$(5039)^3$=127947874319,

mod =$P_1*P_2*P_3$=(641*809*5039)=2613069191 are shown in Table 2.

Table 2 – The average execution time (ns) of the function period_mod() of computing the ME

Release/x86	Release/x86 Intel Core i9-10980XE, trials=2000					
Base *Exp* *mod*	131071 11039 263374721	131080 11039 263374721	131071 263375000 263374721	131080 263375000 263374721	131071 6039 127947874319	131080 6039 127947874319
period_mod()	745	762	1091	1162	710	798
Base *Exp* *mod*	131071 5039 6700417	131071 6039 6700417	131071 6700500 6700417	131071 11039 25391521	131080 11039 25391521	131071 26391521 25391521
period_mod()	791	833	1019	838	853	1110
Base *Exp* *mod*	131071 6700500 127947874319	131080 6700500 127947874319	131071 6039 2613069191	131080 6039 2613069191	131071 6700500 2613069191	131080 6700500 2613069191
period_mod()	1163	1133	729	731	1050	1060

To compute the ME with a given number of trials the values of exponent *Exp*, numbers *Base* and *mod* were given by pseudo-random numbers with number of binary digit to 2048 bits. To reduce the total computation time on increasing the number of digits of big numbers the number of trials of latch-up of the computation time is decrease correspondent. The results are presented in Table 3, which contains the values of average execution time (ns nanoseconds) of computing the ME for pseudo-random data *Base*1, *Exp*1, *mod1* for 1024 bits and *Base*2, *Exp*2, *mod*2 for 2048 bits, that are

Base1 =
155925877528394482614610625993674588019100771066359218078554587162570881239567576804461126115887903799308419845098579180815670096035521874870908961799638269196068560103752308669818128860677719460381304397587862593607968358286567068579479763671817955144283945749615768573725580291910494735428411976050787788916;

Exp1 =
1428352097864899971708732555079465735564482140846285246054211882240996873229846735436141768520710535474156982526225738456830754529824749225178413672786090891369885834477564794839184417957332157713350938927468516042522797303684115973975776261194473923550803446318750817487083947368197108361761563379253499951164;

mod1 =
5891572246297868253412624759745406795585333619437698636102450190718985181854272934901524353228514225430991703852776048028896537550292368120372032210211051014369063473209609243694640800275752961327153630792783722354532277724001895425274132047428375298344510292277365376189309328346658848802247873952610445828 8;

and

Base2 =
256773876049791747456501134392418703199486921699875869870642170952808629559929090723006531686316217942866914409024816653331169514458883444161809664073451110710653136235607137432150724953185446158678719720295928259781123638183830596292580376934671270834776657789712993784966788640286174086177056566697844654876749702991335128692371495759781694921170827273202040085199072418372290679936844100387846101852154889031934611438558681610821715124734828847448121160578454206154924267974589088650928312748724335173725158853105514943013486113643404436304687646801817005256929894904468321901418919444734072243769450781284602123 40;

Exp2 =
206971186674602942893291319268428623712608587189616225423903941285779658834620654647264762656215005844221010903248805904788942050645268534371871257651724912857624853913319544665818453970774205805936276513235167804757330671171360229336910074202766840819523964084411554460264006668359867024199348668244418903647742281408991589590100597574944920059811530558721874424954824117451346461205848045559295541835156979224291886237220605698948712689724650080897444104535423282766208959387777042971769880327762033155857980482303238918889333926985203629914823135222855353547016859173882001780159272297308256145856605458847223736 80;

DOI 10.15588/1607-3274-2022-1-7

mod2 =
1575172313457203559599568743681259608254440573656861701382162511418168603526306451419569115886878055308740967587739361731922128494702349823709785322693518660273267146847449124775067704340487870135582678102049951096446593419054681899518334410792921307143499954265358560545893952695502234824684729370866539585264690942334837436559095193843277113108303325186274650168082850048905318634729938537417490687299729788885279263013200339077021629960904568618885515772917923280644659754459311463103183288771606668121786492047222814542774350966063675771760977395343458836197101195888587251900933188447377466402318085762388758106 8.

Testing for the average execution time of computation of ME (Table 3) was performed by the functions: mpz_powm () from the MPIR library, crypto++ () from the Crypto ++ library. The comparison is performed with previously [22] developed functions Single () and Parallel (). The developed function period_mod () performs the computation of ME by forming an reduced sequence of residuals. The precomputation time to determine of the sequence of residuals is not taken into account.

Table 3 – The average execution time (ns)of the functions of computing the ME

Release/x86	AMD Ryzen 3600		Intel Core i9-10980XE	
Data	*Base1, Exp1, mod1*	*Base2,Exp2,mod2*	*Base1,Exp1, mod1*	*Base2, Exp2, mod2*
bits / trials	1024 / 1000	2048 / 500	1024 / 1000	2048 / 500
Single()	1993761	12916466	2032243	13445459
Parallel()	1678701	9129259	1938135	11366590
crypto++()	2484181	10668126	2607767	10915908
mpz_powm()	1167370	8264648	1196241	8969671
period_mod()	739048	4827014	724754	4927932

The results of the calculation of ME with all functions are compared for the accuracy of their implementation, which confirms the possibility of using the property of periodicity of the sequence of residuals for powers equal to integers of degree two.

6 DISCUSSION

The period_mod () function of the ME reduces the computation time relative to other functions with increasing bit size, starting from data values from 512 bits. Reducing the computation time of the period_mod () function as well as Single () and Parallel () depends on the number of logical one in the binary representation of the *Exp* exponent, which determines the number of multiplication operations in the main stream. The periodicity of the sequence of residues has its own characteristics and depends on the specific values of *Base*, *mod* and *Exp*, because they can differ by many orders of magnitude bits. In the Table 2 shows the cases when *Base* and *mod* are relatively prime (*Base*, *mod*) = 1. The results of the average execution time for the given relatively prime data are consistent with the basic properties that are well studied in number theory.

The software implementation period_mod () through a single-threaded computation shows a slight reduction in the time of determination of the modular exponent with an increase throughput of microprocessors (Table 3). Therefore, based on of the developed software the further implementation of the computation of ME using multithreaded technologies will provide an opportunity the efficient computation of discrete logarithm.

CONCLUSIONS

The work compares and analyses the developed software implementation of the computation of ME and the software implementation of the functions of Crypto++ and MPIR libraries. The computational scheme of the ME, the software implementation of the algorithm using single thread for computing of ME, the run time results of the computation on multi-core microprocessors of universal computer systems have been described. As a result, has developed the function period_mod() of the computation, what speedups the execution of the computations of ME for fixed-base with precomputation. The execution time of the algorithms depends on the specific values of the *Base, mod* and *Exp* of modular exponentiation. The software implementation with increasing the number of binary digits of data shows a reduction of computation time near two times with regard to the MPIR function of computing modular exponentiation.

The scientific novelty of obtained results lies in the implementation of the algorithm of computing the modular exponentiation based on the use of a reduced set of residuals and the fundamental property of modularity.

The practical significance of the work lies in the fact that the obtained results can be successfully apply in the modern asymmetric cryptography, for efficient computation of number-theoretic transforms and other computational problems.

Prospects for further research are that the developed function period_mod() can be used for the organization of multithreading computations of ME.

ACKNOWLEDGEMENTS

The authors are grateful to Roman Rykmas of the team leader of Uniservice LtdC for participation in testing and discussing the results obtained by the functions of computing the ME.

DOI 10.15588/1607-3274-2022-1-7

REFERENCES

1. Studholme C. The Discrete Log Problem [Electronic resource]. Department of Computer Science, University of Toronto, 2002, 57 p. Access mode: http://www.cs.toronto.edu/~cvs/dlog/research_paper.pdf
2. Jakubski A., Perliński R. Review of General Exponentiation Algorithms, *Scientific Research of the Institute of Mathematics and Computer Science*, 2011, Vol. 2, No. 10, pp. 87–98
3. Marouf I., Asad M. M., Al-Haija Q. A. Comparative Study of Efficient Modular Exponentiation Algorithms, *COMPUSOFT, An international journal of advanced computer technology,* August-2017, Vol. 6, Issue 8, pp. 2381–2392.
4. Lara P., Borges F., Portugal R., Nedjah N. Parallel modular exponentiation using load balancing without precomputation, *Journal of Computer and System Sciences*, 2012, Vol. 78, No. 2, pp. 575–582. https://doi.org/10.1016/j.jcss.2011.07.002
5. Nedjah N., Mourelle Ld. M. Three hardware architectures for the binary modular exponentiation: Sequential, parallel, and systolic, *Circuits and Systems I: Regular Papers, IEEE Transactions,* 2006. Vol. 53, Issue 3, pp. 627–633. https://doi.org/ 10.1109/TCSI.2005.858767.
6. Vollala S., Ramasubramanian N., Tiwari U. Energy-Efficient Modular Exponential Techniques for Public-Key Cryptography. Springer Nature, Singapur, Pte Ltd. 2021, 255 p. https://doi.org/10.1007/978-3-030-74524-0
7. PARI/GP home. [Electronic resource]. Access mode: http://pari.math.u-bordeaux.fr/
8. MPIR: Multiple Precision Integers and Rationals. [Electronic resource]. Access mode: http://mpir.org/
9. Crypto++ Library 8.6 Electronic resource]. Access mode: https://www.cryptopp.com
10. Knuth D. E. The art of computer programming. 3d ed. Reading (Mass), Addison-Wesley, cop. 1998, 712 p.
11. Bach E., Shallit J. Algorithmic Number Theory, Volume I, Efficient Algorithms. Cambridge, USA: MIT Press. 1996, 516 p.
12. Cohen H. A course in computational algebraic number theory. Berlin, Heidelberg, Springer. 1993, 536 p. https://doi.org/10.1007/978-3-662-02945-9
13. Menezes A. J., Oorschot van P. C., Vanstone S. A.. Handbook of Applied Cryptography, 5th printing, Boca Raton. CRC Press, 2001, 816 p.
14. Sorenson J. P. [Electronic resource] A sublinear-time parallel algorithm for integer modular exponentiation, 1999. pp. 1–8. Access mode: https://www.researchgate.net/publication/2274099_A
15. Robert J.-M., Negre C., Plantard T. Efficient Fixed Base Exponentiation and Scalar Multiplication based on a Multiplicative Splitting Exponent Recoding, *Journal of Cryptographic Engineering, Springer*, 2019, Vol. 9, Issue 2, pp. 115–136. https://doi.org/10.1007/s13389-018-0196-7.
16. Joye M. and Tunstall M. Exponent Recoding and Regular Exponentiation Algorithms, *Conference on Cryptology in Africa (Africacrypt 2009): Second International Conference*. Gammarth, Tunisia, 2009, proceedings. Published by Springer, 2009, pp. 334–349.
17. Rosen K. H. Elementary number theory and its applications 6th ed., China: Pearson/Addison Wesley, 2011, 721 p.
18. Prots'ko I. Kryvinska N., Gryshchuk O. The Runtime Analysis of Computation of Modular Exponentiation, *Radio Electronics, Computer Science, Control*, 2021, No. 3, pp. 42–47. DOI: https://doi.org/10.15588/1607-3274-2021-3-4

Received 20.12.2021.
Accepted 15.01.2022.

УДК 004.421

ОБЧИСЛЕННЯ МОДУЛЬНОЇ ЕКСПОНЕНТИ ДЛЯ ФІКСОВАНОЇ ОСНОВИ З ПЕРЕДОБЧИСЛЕННЯМ СКОРОЧЕНОГО НАБОРУ ЗАЛИШКІВ

Процько І. – д-р техн. наук, доцент, кафедра автоматизованих систем управління, Національний університет «Львівська політехніка», Львів, Україна.

Грищук О. – розробник програмного забезпечення, ТОВ «СофтСерв», Львів, Україна.

АНОТАЦІЯ

Актуальність. Модульне піднесення до степеня є важливою операцією в багатьох застосуваннях, що вимагає великої кількості обчислень. Швидкі обчислення модульної експоненти вкрай необхідні для ефективних обчислень у теоретично-числових перетвореннях, для забезпечення високої криптостійкості інформаційних даних та в багатьох інших завданнях.

Мета – аналіз часу виконання програмних функцій розрахунку модульної експоненти з розробленою програмою, що використовує попереднє обчислення зменшеного набору залишків для фіксованої бази.

Метод. Модульне піднесення до степеня реалізовано з використанням методу двійкового зсуву справа наліво для фіксованого базису з попереднім обчисленням зменшеного набору залишків. Для ефективного обчислення модульної експоненти великих чисел використовується властивість періодичності послідовності залишків фіксованої бази з експонентами, що дорівнюють цілочисельній степені двійки.

Результати. Проведено порівняння часу виконання п'яти варіантів функцій для обчислення модульного піднесення до степеня. В алгоритмі з попереднім обчисленням зменшеного набору залишків для фіксованої бази забезпечується більш швидке обчислення модульної експоненти для значень даних, що перевищують 1K двійкових розрядів, порівняно з функціями модульного піднесення до степеня бібліотек MPIR і Crypto++. Бібліотека MPIR з цілочисельним типом даних з кількістю двійкових розрядів від 256 до 2048 біт використовується для розробки алгоритму обчислення модульного піднесення до степеня.

Висновки. У роботі розглянуто та проаналізовано розроблену програмну реалізація обчислення модульної експоненти на універсальних комп'ютерних системах. Одним із способів реалізації прискорення обчислення модульного піднесення до степеня є розробка алгоритмів, які можуть використовувати попереднє обчислення зменшеного набору залишків для фіксованої бази. Програмна реалізація модульного піднесення до степеня зі збільшенням від числа 1K двійкових розрядів даних показує покращення часу обчислень у порівнянні з функцією модульного піднесення до степеня бібліотек MPIR та Crypto++.

КЛЮЧОВІ СЛОВА: модульне піднесення до степеня, великі числа, алгоритм зведення до степеня, фіксована базова ступінь, множина залишків.

DOI 10.15588/1607-3274-2022-1-7

УДК 004.421

ВЫЧИСЛЕНИЕ МОДУЛЬНОЙ ЭКСПОНЕНТЫ ДЛЯ ФИКСИРОВАННОЙ ОСНОВЫ С ПРЕДВЫЧИСЛЕНИЕМ СОКРАЩЕННОГО НАБОРА ОСТАТКОВ

Процько И. – д-р техн. наук, доцент кафедры автоматизированных систем управления, Национальный университет «Львивська политэхника», Львов, Украина.

Грищук О. – разработчик программного обеспечения, ООО «СофтСерв», Львов, Украина.

АННОТАЦИЯ

Актуальность. Возведение в степень – важная операция во многих приложениях, требующая большого количества вычислений. Быстрые вычисления модульного возведения в степень необходимы для эффективных вычислений в теоретико-численных преобразованиях, для обеспечения высокой криптостойкости информационных данных и во многих других приложениях.

Цель – анализ времени выполнения программных функций расчета модульной экспоненты с разработанной программой, использующей предварительные вычисления сокращенного набора остатков для фиксированной базы.

Метод. Модульное возведение в степень реализовано с использованием разработки метода двоичного сдвига справа налево для фиксированного базиса с предварительным вычислением уменьшенного набора остатков. Для эффективного вычисления модульной экспоненты больших чисел используется свойство периодичности последовательности остатков фиксированной базы с экспонентами, равными целочисленной степени двойки.

Результаты. Проведено сравнение времени выполнения пяти вариантов функций для вычисления модульной экспоненты. В алгоритме с предварительным вычислением сокращенного остатка набор для фиксированной базы обеспечивается более быстрое вычисление модульного возведения в степень для значений, превышающих 1K двоичных цифр, по сравнению с функциями модульной экспоненты библиотек MPIR и Crypto++. Библиотека MPIR с целочисленным типом данных с количеством двоичных разрядов от 256 до 2048 бит используется для разработки алгоритма вычисления модульного возведения в степень.

Выводы. В работе рассмотрена и проанализирована разработанная программная реализация вычисления модульной экспоненты на универсальных компьютерных системах. Один из способов реализации ускорения вычисления модульного возведения в степень является разработка алгоритмов, которые могут использовать предварительное вычисление сокращенного набора остатков для фиксированной базы. Программная реализация модульного возведения в степень с увеличением с 1024 числа двоичных разрядов экспоненты показывает улучшение времени вычислений по сравнению с функциями модульной экспоненты библиотек MPIR и Crypto++.

КЛЮЧЕВЫЕ СЛОВА: модульное возведение в степень, большие числа, алгоритм возведения в степень, возведение в степень с фиксированной основой, множество остатков.

ЛІТЕРАТУРА / ЛИТЕРАТУРА

1. Studholme C. The Discrete Log Problem [Electronic resource] / C. Studholme // Department of Computer Science, University of Toronto. – 2002. – 57 p. Access mode: http://www.cs.toronto.edu/~cvs/dlog/research_paper.pdf
2. Jakubski A. Review of General Exponentiation Algorithms / A. Jakubski, R. Perliński // Scientific Research of the Institute of Mathematics and Computer Science. – 2011. – Vol. 2, № 10. – P. 87–98
3. Marouf I. Comparative Study of Efficient Modular Exponentiation Algorithms. / I. Marouf, M. M. Asad, Q. A. Al-Haija // COMPUSOFT, An international journal of advanced computer technology. – August-2017. – Vol. 6, Issue 8. – P. 2381–2392.
4. Parallel modular exponentiation using load balancing without precomputation / [P. Lara, F. Borges, R. Portugal, N. Nedjah] // Journal of Computer and System Sciences. – 2012. – Vol. 78, № 2. – P. 575–582. https://doi.org/10.1016/j.jcss.2011.07.002
5. Nedjah N. Three hardware architectures for the binary modular exponentiation: Sequential, parallel, and systolic / N. Nedjah, Ld. M. Mourelle // Circuits and Systems I: Regular Papers, IEEE Transactions. – 2006. – Vol. 53, Issue 3. – P. 627–633. https://doi.org/ 10.1109/TCSI.2005.858767.
6. Vollala S. Energy-Efficient Modular Exponential Techniques for Public-Key Cryptography. / S. Vollala, N. Ramasubramanian, U. Tiwari. – Springer Nature, Singapur: Pte Ltd. 2021. – 255 p. https://doi.org/10.1007/978-3-030-74524-0
7. PARI/GP home. [Electronic resource]. – Access mode: http://pari.math.u-bordeaux.fr/
8. MPIR: Multiple Precision Integers and Rationals. [Electronic resource]. – Access mode: http://mpir.org/
9. Crypto++ Library 8.6 Electronic resource]. – Access mode: https://www.cryptopp.com
10. Knuth D. E. The art of computer programming / D. E. Knuth. – 3d ed. Reading (Mass): Addison-Wesley, cop. 1998. – 712 p.
11. Bach E. Algorithmic Number Theory / E. Bach, J. Shallit. – Volume I: Efficient Algorithms. – Cambridge, USA: MIT Press. 1996. – 516 p.
12. Cohen H. A course in computational algebraic number theory / H. Cohen. – Berlin, Heidelberg : Springer, 1993. – 536 p. https://doi.org/10.1007/978-3-662-02945-9
13. Menezes A.J. Handbook of Applied Cryptography / A. J. Oorschot, S. A. Vanstone. – 5th printing, Boca Raton : CRC Press. 2001. – 816 p.
14. Sorenson J. P. [Electronic resource] A sublinear-time parallel algorithm for integer modular exponentiation, 1999. – P. 1–8. – Access mode: https://www.researchgate.net/publication/2274099
15. Robert J.-M. Efficient Fixed Base Exponentiation and Scalar Multiplication based on a Multiplicative Splitting Exponent Recoding / J.-M. Robert, C. Negre, T. Plantard // Journal of Cryptographic Engineering, Springer. – 2019. – Vol. 9, Issue 2. – P. 115–136. https://doi.org/10.1007/s13389-018-0196-7.
16. Joye M. Exponent Recoding and Regular Exponentiation Algorithms / M. Joye and M. Tunstall // Conference on Cryptology in Africa (Africacrypt 2009): Second International Conference, Gammarth, Tunisia, 2009: proceedings. – Published by Springer, 2009. –P. 334–349.
17. Rosen K. H. Elementary number theory and its applications / K. H. Rosen. – 6th ed., China : Pearson/Addison Wesley, 2011. – 721 p.
18. Prots'ko I. The Runtime Analysis of Computation of Modular Exponentiation / I. Prots'ko, N. Kryvinska, O. Gryshchuk / Радіоелектроніка, інформатика, управління. – 2021. – № 3. – С. 42–47. DOI: https://doi.org/10.15588/1607-3274-2021-3-4

DOI 10.15588/1607-3274-2022-1-7