Research Journal of Applied Sciences 10 (8): 442-446, 2015
ISSN: 1815-932X

Comparative Analysis of Module Reduction Calculation Algorithms

Dmitriy A. Dolgov
Kazan Federal University, Kremlevskaya Street 18, Kazan, Russia

Abstract: The most popular tasks in cryptography are information encryption\decryption tasks, the calculation of a digital signature, the calculation of the greatest common divisor for several numbers, the creation of safe (cryptographically resistant) data transmission protocols and so on. The protocols use the asymmetric cryptography for keys authentication, the symmetric encryption for confidentiality preservation, the message authentication codes to ensure the message integrity. But at the same time with the increase of local network capacity as well as internet traffic, the role of cryptographic protocol performance increases. Many cryptographic algorithms are performed over a field, so the acceleration of the basic arithmetic operations implementation (addition, multiplication, exponential calculation) will speed up the implementation of the whole algorithm. This study describes the effective versions of multiplication, exponent calculation and modular reduction algorithms for a wide range of cryptography issues. The detailed description of these algorithms is provided, combined with mathematical transformations necessary for their analysis, the field of their use is described, their comparative analysis is performed on the basis of which we may obtain for example, a more efficient version of asymmetric cryptography algorithms such as RSA, El Gamal and others and thus develop some faster protocols.

Key words: Modular reduction, module, montgomery reduction, the reduction of Barrett, table modular reduction, public key cryptography

INTRODUCTION

The public key cryptography is closely related to the concept of one-way function that is with such a function f(x) that may be obtained if x is known, when the calculation of inversion is the problem which is solved difficultly (Fig. 1).

The public key cryptography uses one-way functions with a secret. The secret may be presented by a private key which helps to decrypt a message. That is there is such an Y knowing that allows to calculate x if f(x) and Y is known (Fig. 2).

The most commonly used cryptosystems with a public key are based on modular arithmetic:

- The best known algorithm for public-key cryptography is RSA (Menezes *et al*., 1996). The basis of the cryptographic system is the problem of two large prime numbers factoring. The operation of exponentiation according to large number module is used for encryption. One has to decrypt Euler function within a reasonable time from this large number which is necessary for the knowing of a number decomposition into prime factors

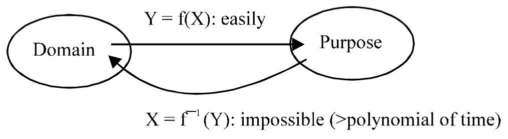

Fig. 1: The domain to purpose inversion with polynomial of time

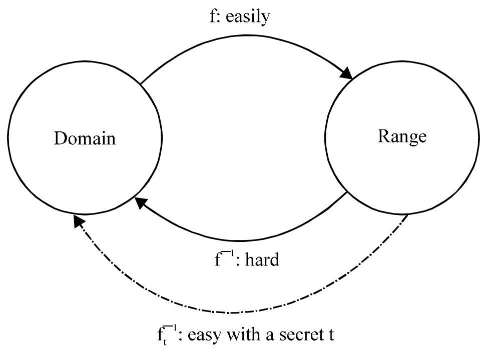

Fig. 2: The domain to range inversion with a secret (t)

- Diffie-Hellman, ElGamal and DSA algorithms are based on a simple module exponentiation
- ECC may be implemented by simple fields

The most important operation is the multiplication by module. A simple way of this operation performance is a simple division:

$$A \times B \bmod C = \frac{\left(\left(A - \left[\frac{A}{C}\right] \times C\right) \times \left(B - \left[\frac{B}{C}\right] \times C\right)\right) \bmod C}{D} = D - \left[\frac{D}{C}\right] \times C$$

But this method is inefficient because it depends on the module length:

- 1024 bits for RSA, Diffie-Hellman, ElGamal and DSA algorithms
- 160 bit for the algorithms based on elliptic cryptography

Another important operation is the calculation of the following exponent: $A^B \bmod C$. Typically it is calculated using the fast exponentiation algorithm (Seong-Min *et al.*, 2013). It amounts to the multiplicative analogue of Horner scheme. The algorithm helps to speed up the computation by reducing the operations of exponentiation, nevertheless the module calculation is performed via division (Diffie and Hellman, 1976).

Algorithm: The fast exponentiation algorithm (square and multiply alg.). Entry: a, b, n. Exit: $a^b \bmod n$:

- Obtain a binary expansion $b = (b_0 b_1 b_2)_2$, $b_i \in \{0, 1\}$
- $C_0 = a$, $C_{i+1} = \{M\ Mult\ (M\ Mult\ (c_i))$ for $i \geq 0$
- Return C_{i+1}

Amount of multiplications in an algorithm: $O(\log_2 n)$.

MONTGOMERY REDUCTION

The computation of the multiplication by module using the computational strategy "multiplication-residue" is used in the algorithm of fast exponentiation, involves an efficient computation of module. Since, we want to avoid the accurate division which is very "expensive" operation, the method is necessary that will help us to avoid it (Parhami, 1994). Montgomery proposed the algorithm capable of multiplication performance according to a certain module within the time which is almost equivalent to the multiplication of several numbers with high precision. If n is the number of digits 2 for multiplied numbers, then the computational complexity of Montgomery multiplication is as follows: $2n(n+1) = 2n^2 + 2n$ single-precision multiplications, when a multiplication precision requires n^2.

Montgomery proposed an efficient algorithm for computing a multiplication module. The idea is to move all calculations from the ring Z_N into a special residual domain, the domain of Montgomery which will run all operations.

Let N>0, we define the ring $Z_N = \{0,1,2,\ldots, N\text{-}1\}$. We define Montgomery transformation:

$$\mu: (Z_N, +, X) \rightarrow (\widetilde{Z_N}, +, *)$$

Multiplication in $\widetilde{Z_N}$ is performed faster, so if the algorithm has a long chain of multiplications by module in Z_N, then the optimal strategy is to transit the Montgomery $\widetilde{Z_N}$ domain operands, perform the multiplication within it and convert the result back to Z_N (Parhami, 1997; Lim *et al.*, 1997). If the original formula has also the operations of addition, subtraction, we will fulfill them between multiplications according to the rules in Montgomery $\widetilde{Z_N}$ domain and then bring the result back to Z_N (Montgomery, 1985).

Montgomery transformation: x the element of Z_N, positive R>N:

$$NOD(R,N) = 1 = R \times R' - N \times N'$$

where the coefficients R' and N' satisfy the following relations:

$$R' \equiv R^{-1} \bmod N, 0 \leq R' < N$$

$$N' \equiv -N^{-1} \bmod R, 0 \leq N' < R$$

Montgomery transformation μ is determined:

$$\tilde{x} = \mu(x) = x \times R \bmod N$$

$$x = \mu^{-1}(\tilde{x}) = \tilde{x} \times R' \bmod N$$

T.к R>N and R is mutually simple with N, then $R = 2^k$ is chosen as a rule (Barrett, 1986). In this case, the division into R would mean a simple shift to the right by k position which is performed very fast.

Further, all calculations are performed on the image. In order to transit to a natural representation of x it is necessary to divide by R.

Addition in montgomery domain: Initial elements $x, y \in Z_N$, images $\tilde{x}, \tilde{y} \in \widetilde{Z_N}$.

$$\tilde{x} \pm \tilde{y} = xR \bmod N \pm yR \bmod N = (x \pm y) R \bmod N = \widetilde{(x \pm y)}$$

Algorithm MRed(.); Montgomery reduction:
Entry: $0<x\leq RN$, N,R: R>N, gcd(R, N) =1 = RR'-NN'
Exit: $xR^{-1} \bmod N$

- t' = xN' mod R#1 modular multiplication mod R
- x = x+Nt`#1 addition according to module R
- x = x/R#Shift
- If x≥N then
- x = x-N
- Back x

The algorithm is able to calculate xR^{-1} mod N, using 1 multiplication, 1 addition, 2R module calculations. Using the algorithm MRed one may describe the operation of two numbers multiplication in the domain $\widetilde{Z_N}$.

MMult(.,.) algorithm; Montgomery multiplication:
Entry: $\tilde{x} = xR,\ \tilde{y} = yR \in Z_N$
Exit: $\tilde{z} = xyR \bmod N$
Back: $\tilde{z} = MRed(\tilde{x} \times \tilde{y})$

MMult returns the result from Montgomery domain:

$$\tilde{x} \times \tilde{y} \bmod N = (xR \times yR) \times R^{-1} \bmod N = \\ xyR^2R^{-1} \bmod N = \widetilde{xy} \bmod N$$

The advantage of this approach is that MMult uses 1 multiplication together with 1 multiplication, 1 addition, 1 MRed algorithm shift which is considerably faster, than the performance of regular reductions according to N module. We may express μ and μ^{-1}:

$$\mu : (Z_N, +, x) \rightarrow (\widetilde{Z_N}, +, *) : \tilde{x} = \mu(x) = \\ xR \bmod N = MMult(x, R^2 \bmod N)$$
$$\mu^{-1} : (\widetilde{Z_N}, +, *) \rightarrow (Z_N, +, x) : x = \mu^{-1}(\tilde{x}) = \\ \tilde{x}R' \bmod N = MMult(\tilde{x}, 1)$$

MExp algorithm (.,.); Montgomery exponent:
Entry: $\tilde{x} = xR,\ e = (e_0, e_1, e_2, ..., e_k)_b$
Exit: x^e mod N

- $C_0 = x$, C_{i+1} = {M Mult (M Mult (c_i)) for i≥0
- Return C_{i+1}

The algorithm demands $2\log_2 N$ multiplications, additions and shifts.

BARRETT REDUCTION

Barrett presented the algorithm that computes a modular reduction which he used for the implementation of RSA. At first glance, the modular reduction ia just the calculation an integer division remainder:

$$Z \bmod N = Z - \left\lfloor \frac{Z}{N} \right\rfloor N = Z - qN, q = \left\lfloor \frac{Z}{N} \right\rfloor$$

The Barrett's idea was that multiplication will be used instead of division on a precomputed constant which approximates the module inversion. Therefore, the direct calculation of is avoided, $\hat{q}$ is calculated instead:

$$\hat{q} = \left| \frac{\lfloor 2^{n-1} \rfloor \lfloor N \rfloor}{2^{n+1}} \right|$$

This equation appears to be complicated nevertheless the division by 2^{n-1} and 2^{n+1} is just a simple shift. The equation $\lfloor 2^{2n}/N \rfloor$ depends only on the module N and is a constant one, until the module is not changed. Thus, this constant may be calculated in advance. The calculation of the module will be reduced to two simple multiplications and rounding operation. It turns out that the result cannot be completely reduced, but it is in the range from 0 to 3N-1. Therefore, 1 or 2 N reduction may be necessary for an accurate result.

Barrett algorithm (Menezes *et al.*, 1996):

$$\text{Input: } x>0,\ m>0:\ x = (x_{2k-1}, ..., x_1, x_0)_b, \\ m = (m_{k-1}, ..., m_1, m_0)_b,\ m_{k-1} \neq 0, \\ \mu = \left[b^{2k}/m\right],\ b > 3$$

Output: r = x mod n

- $q_1 = [x/b^{k-1}]$, $q_2 = q_1 \times \mu$, $q_3 = [q_2/b^{k+1}]$
- $r_1 = x \bmod b^{k+1}$, $r_2 = q_3 \times m \bmod b^{k+1}$, $r = r^1 - r^2$
- If r<0 then $r = r+b^{k+1}$
- While r>m: r = r-m
- Return (r)

BARRETT AND MONTGOMERY ALGORITHM

Similarities:

- A preliminary calculation of a number of constants
- The reduction avoids expensive division operations

Differences:

- Montgomery algorithm requires the transfer into Montgomery domain, the Barrett algorithm works directly with numbers. This allows Montgomery algorithm to implement the exponentiation operation effectively
- Montgomery algorithm performs all operations with high precision, Barrett's algorithm is based on approximation of a real value with a limited accuracy
- Barrett's algorithm is applied to the same module during the calculation of multiple reductions

BASIC SEARCH METHOD BY TABLE

As during the calculation of a number exponential the module is fixed, you may use the table of precalculated values. Given: $n>0$: $2^{k-1}<n<2^k$. Let's create a precomputed Table 1:

$$r[l] \leq \left\lceil \frac{n}{2} \right\rceil, l = k,...,2k-1$$

Find: $z \bmod n \times z$ is composed of k bits, n is composed of t bits lim (k/t) = 2. The table of precalculated values is calculated:

$$n[j] = b^{k+j} \bmod n \,(o<j<k)$$

We compute:

$$Z = \sum_{j=0}^{2k-1} Z_j b^j$$

b is a computer word size, so this formula is the expansion of the number in the computation system b, here b = 2. It is equivalent to:

$$Z = Z[0{:}k] + \sum Z_{k+j} n[j]$$

Thus, z mod n is k+1 it is the least significant digits of the number within the calculation system b plus $[\log_2 k]$ of precalculated values.

If we have a target value and it is multiplied by a constant, the latter is entered into the table of precalculated values:

$$y = cx = \sum_{j=0}^{k-1} x_j \times c \times b^j \bmod n$$

$$c[j] = c \times b^j \bmod n \,(0 \leq j < k)$$

$$y = \sum_{j=0}^{k-1} x_j \times c[j] \bmod n$$

Table 1: The precomputer values

l	2k-1	2k-2	...	k
r[l]	2^{2k-1} mod n	2^{2k-2} mod n	...	2^k mod n

Let's consider the algorithm for fast exponentiation. The1 is better to use for the calculation of a module after finding a square, A is best of all to use for the obtaining of the remainder after the multiplication by a constant. In this case, we will have 2 tables of precalculated values. Let's define the basic algorithm?

Entry: n, k = BitLength(n), $0 \leq z < 2^{2k}$, T = {r[2k-1], r[2k-2],..., r[k]}.

Exit: z mod N

- If z<n, let's return (z)
- If BitLength(z) = k, then let's return (z-n)
- s = Binary(z), r = 0 for i from BitLength(z)-1 down to k do if s[i] = 1, r = r + r[i]
- $r = r + \sum_{j=0}^{k-1} s[j]2^j$
- While r≥n:r = r-n
- While r<0:r = r+n
- Let's return (r)

The number of additions in this method depends on the number of units in the left part of n number decomposition. If it consists of 2k bits, the average number of units in the left part is k/2. Since:

$$r[l] \leq \left\lceil \frac{n}{2} \right\rceil, l = k,..., 2k-1$$

then |r|<kn/4. Thus [k/4] deduction is necessary for the calculation of z mod n. Thus, the method demands [3k/4] additions.

SUMMARY

The abovementioned algorithms at the correct use may significantly reduce the computation time. The most promising method is the search method by table and Montgomery Method. Further improvements of Montgomery Method is associated with the search of a new domain and a transformation operation. The search method by the table depends on the efficiency of preliminary computations, their possible transformations and the search of the most effective number decomposition as the sum.

CONCLUSION

Barrett Method is the best one for any reduction. It is much superior to other methods. At a fixed module, such as the calculation of the exponent the search method according to the table shows the best results. Montgomery Method is slightly worse.

IMPLEMENTATION

Implemented and tested on the computer with AMD Phenom P820 processor Triple-Core 1.8 Ghz, 4GB RAM, Windows 7x64, the library GNU GMP was used.

The calculation of arbitrary reduction for 30-50 bit numbers. Because of the module variability it negates all possible preliminary calculations (Bosselaers and Vandewalle, 1993). The testing of reduction computation speed makes 500 numbers (in seconds) (Fig. 3).

If we fix the module, we get a significant acceleration of the Montgomery Calculation Method and the basic search method on the table due to the fact that all the operations of reduction calculation have been performed already, only the operations of addition and multiplication

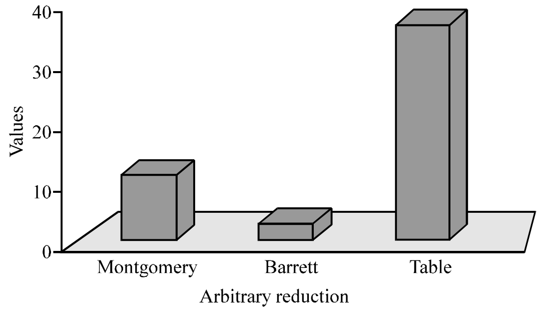

Fig. 3: The preliminary calculation

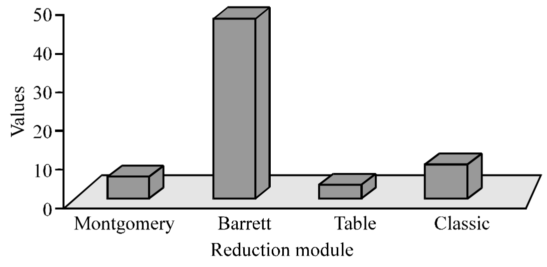

Fig. 4: The speed made reductions

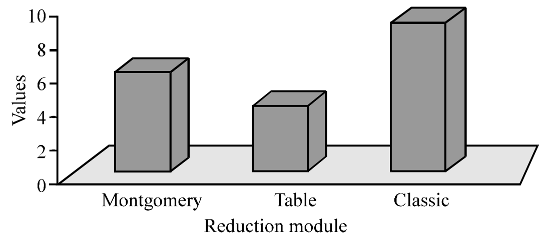

Fig. 5: The reduction according to the fixed module

remained basically (MacSorley, 1961). In view of the low-level implementation of Montgomery Method in GMP library, the classical approach was implemented manually. The checking of calculation speed made 100,000 reductions (in seconds) (Fig. 4): The same diagram but without Barrett reduction.

The calculation of a product or an exponent may also be reduced to the calculation of reduction. Therefore, the histogram of the exponent calculation at a fixed unit will have a similar appearance (Fig. 5).

ACKNOWLEDGEMENT

The research is performed according to the Russian Government Program of Competitive Growth of Kazan Federal University.

REFERENCES

Barrett, P., 1986. Implementing the Rivest Shamir and Adleman Public Key Encryption Algorithm on a Standard Digital Signal Processor. Proceedings CRYPTO, pp: 311-323.

Bosselaers, R.G. and J. Vandewalle, 1993. Comparison of Three Modular Reduction Functions. Proc. CRYPTO, pp: 175-186.

Diffie, W. and M.E. Hellman, 1976. New directions in cryptography. IEEE Trans. Computers, IT-22 (6): 644-654.

Lim, C., H. Hwang and P. Lee, 1997. Fast modular reduction with precomputation. In: Proc. of Korea Japan Joint Workshop on Information Security and Cryptology, pp: 65-79.

Menezes, A., P. van Oorschot and S. Vanstone, 1996. Handbook of Applied Cryptography [Tent]. CRC Press.

Montgomery, P.L., 1985. Modular Multiplication Without Trial Division. Mathematics of Computation 4A, 170: 519-521.

MacSorley, O.L., 1961. High-Speed Arithmetic in Binary Computers. Proceedings of the Institute of Radio Engineers, 49: 67-91.

Parhami, B., 1994. Analysis of tabular methods for modular reduction. In: Proc. 28th Asilomar Conf. Signals, Systems and Computers. Pacific Grove, CA, pp: 526-530.

Parhami, B., 1997. Modular reduction by multi-level table lookup. In: Proc. Midwest Symposium on Circuits and Systems, MWSCAS, pp: 381-384.

Seong-Min H., Sang-Yeop Oh and Hyunsoo Yoon, 2013. New Modular Multiplication Algorithms for Fast Modular Exponentiation. Department of Computer Science and Center for AI Research.