Algorithm Designs and Complexity Analyses for Information Security

Chia-Long Wu
Professor and Director of Aviation Communication Electronics, Air Force Institute of Technology

***Abstract* —Modular exponentiation is composed of repetition of modular multiplication and is often the dominant part of modern cryptographic method. Taking the RSA cryptosystem for example, both the encryption and decryption operations are accomplished by modular exponentiation. It is performed by using successive modular multiplications. Exponentiation is to compute M^E for a positive integer E and modular exponentiation is to compute M^E mod N for positive integers E and N. When the lengths of the operators are at least 512 up to 2048 binary representations, modular exponentiation can be time-consuming and is often the dominant part of the computation in many computer systems. In this paper, I will describe some methods, which use some software. Most importantly, I will analyze the computational complexities for these methods respectively.**

I. INTRODUCTION

In many computation problems, the modular exponentiation is a common operation for scrambling secret data and is used by several public-key cryptosystems. We know modular arithmetic is the most dominant part of the computation which is performed in the RSA encryption system. Modular exponentiation is the most important operation in RSA public-key cryptosystem. Modular arithmetic is the most dominant part of the computation which is performed in both the public-key cryptosystems and key exchange scheme for modern cryptographic applications. The operation is time-consuming for large operands. The motivation of studying high-speed and space-efficient algorithms for modular exponentiation usually comes from the applications in modern cryptography. For RSA cryptosystem, the public and private keys are functions of a pair of large prime numbers, and the encryption and decryption operations are calculated by modular exponentiation. Fast modular exponentiation algorithms are often considered of practical significance in RSA cryptosystem. Computational complexity theory is a branch of the theory of computation in theoretical computer science. The modular exponentiation is composed of repetition of modular multiplications. Therefore, modular exponentiation can be time consuming, and is often the dominant part of modern cryptographic algorithms for key exchange, electronic signature, and authentication. As the "signed-digit recoding" has less occurrence probability of nonzero digit than binary number representation. Taking this advantage, we can effectively decrease the amount of modular multiplications. By using the technique of recording the common parts in the folded substrings, the "folding algorithm" can further improve the efficiency of the binary algorithm, thus can decrease the computational complexity of modular exponentiation. In this paper, we will concentrate on the first approach to effectively reduce the number of modular multiplications in modular exponentiation.

The modular exponentiation is a common operation for most cryptosystems and smart card systems. Most of cryptographic systems based on modular exponentiation. Generally, modular exponentiation is represented using a chain of modular multiplications. The performance of such cryptosystems is primarily determined by the implementation efficiency of the multiplication and the exponentiation. Many relevant papers about computer security are issued in many reports and journals to describe how to reduce the computational complexities in the cryptosystems. In the rest of this paper, some methods which reduce the number of multiplications by using some modular algorithm such as binary method, sliding-window and division for residue number are presented in Section 2. In Section 3, we present other techniques (Montgomery and CMM methods) which accelerate the multiplication itself by reducing the redundant computations. In Section 4, we will detailed depict for computational complexity analyses. Finally, some concise conclusions and future works are given in Section 5.

II. MODULAR ARITHMETIC

Modular exponentiation is important in the modular arithmetic, in this section, we will describe binary method [1-3], sliding window method [4], and division for residue numbers [5].

A. Binary Method

The most commonly used algorithms for computing α^r are the binary methods [6]. The binary methods (also called square-and-multiply methods) scan the bits of exponent r either from right to left or from left to right (Algorithm 1 and Algorithm 2). The right-to-left (RL) algorithm is based on the observation that $\alpha^r = (\alpha^{2^1})^{r_1}(\alpha^{2^2})^{r_2}...(\alpha^{2^m})^{r_m}$, while the left-to-right (LR) algorithm follows from $\alpha^r = (...(((\alpha^{r_m})^2\alpha^{r_{m-1}})^2\alpha^{r_{m-2}})^2...\alpha^{r_2})^2\alpha^{r_1}$.

Algorithm 1: (RL algorithm)
INPUT: α, $r = (r_m, \ldots, r_1)_2$
OUTPUT: $M=\alpha^r$
M=1; $S=\alpha$
  for i ***from*** 1 **to** m do
    if (r_i=1) ***then*** $M=M*S$
    $S=S*S$
  end for;
return M;
end.

Corresponding author:chialongwu@gmail.com
DOI:10.6159/IJSE.2014.(4-1).01

Algorithm 2: (LR algorithm)
INPUT: α, $r = (r_m, \ldots, r_1)_2$
OUTPUT: $M=\alpha^r$
M=1
for i ***from*** m ***downto*** 1 **do**
$M=M*M$
if $(r_i=1)$ ***then*** $M=M*\alpha$
end for;
return M;
end.

We remark that the LR algorithm (Algorithm 2) requires two registers (for α and for M) and that the RL algorithm (Algorithm 1) requires one more register (for S). However, we note that S can be used in place of α if the value of α is not needed thereafter. The RL algorithm presents the advantage to be parallelizable: one multiplier performs the multiplications $M=M*S$ and another one performs the squarings $S=S^2$. However, if only one multiplier is available, the LR algorithm may be preferred because the multiplications are always done by the fixed value α, $M=M*\alpha$. So, if α has a special structure, these multiplications may be easier than multiplying two arbitrary numbers.

B. Sliding Window Method

Let $E=(E_kE_{k-1}E_{k-2}\ldots E_2E_1)$ be the binary expansion of the exponent E, where n is the number of bits in the binary expansion of E. This representation of E is partitioned into n words of length d, such that $nd=k$. The exponent is padded with at most d-1 zeros, if d does not divide k. We define

$$F_i = (E_{id+d-1}E_{id+d-2}\ldots E_{id}) = \sum_{j=0}^{d-1} E_{id+j}\,2^j \qquad (1)$$

such that $0\le F_i\le 2^d-1$ and $E = \sum_{i=1}^{n} F_i\,2^{id}$. The sliding window method first computes the values of X^W for W=2, 3, …, 2^d-1. The exponent E is then scanned d bits at a time from the most significant to the least significant. At each step, the partial result is raised to the 2^d power and multiplied with X^{F_i} where F_i is the current nonzero word. The sliding window method is described in Algorithm 3.

Algorithm 3 (The Sliding Window Method)
Input: X, E
Output: $y=X^E$
Compute and store X^w for all w=2, 3, 4, …2^d-1.
Decompose E into d-bit words F_i for i=1, 2, …, n.
$y = X^{F_{k-1}}$
for i=n-1 ***downto*** 1
{ $y = y^{2^d}$
if $F_i\neq 0$ ***then*** $y=y*X^{F_i}$ }
return y

C. Division for Residue Numbers

In this section, we shall present division operation algorithm in the RNS, inspired by Gamberger's research [5]. The advantage of this method is the residues rather than transforming to be a binary number. The dividend and the divisor are in the RNS representation. The multiplication and subtraction in the method can then be performed in parallel. Every processor unit computes with two corresponding residues in the RNS representations of the dividend and the divisor.

Algorithm 4 (Division for Residue Numbers)
Step 1 Let the moduli be $q_1, q_2, \ldots, q_m$, the dividend be $(a_1, a_2, \ldots, a_m)$, and the divisor be $(b_1, b_2, \ldots, b_m)$.
Step 2 Determine auxiliary moduli set $(q_1', q_2', \ldots; q_m')$, where $\gcd(q_i, q_j')=1$ for i=1, 2, ..., m and j=1, 2, ..., m.
Step 3 Compute $(p_1, p_2, \ldots, p_m)$, where $p_i = q_1'*q_2'* \ldots * q_m' \bmod q_i$.
Step 4 Convert $(a_1, a_2, \ldots, a_m)$ modulo to $q_1, q_2, \ldots, q_m$ into $(a_1', a_2', \ldots, a_m')$ modulo to $q_1', q_2', \ldots, q_m'$.
Step 5 Convert $(b_1, b_2, \ldots, b_m)$ modulo to $q_1, q_2, \ldots, q_m$ into $(b_1', b_2', \ldots, b_m')$ modulo to $q_1', q_2', \ldots, q_m'$.
Step 6 Compute $(b_1^{-1}, b_2^{-1}, \ldots, b_m^{-1})$ modulo to $q_1, q_2, \ldots, q_m$, $(b_1'^{-1}, b_2'^{-1}, b_m'^{-1})$ modulo to $q_1', q_2', \ldots, q_m'$, and $(p_1^{-1}, p_2^{-1}, \ldots, p_m^{-1})$ modulo to $q_1, q_2, \ldots, q_m$.
Step 7 Compute $(c_1', c_2', \ldots, c_m') = (a_1', a_2', \ldots, a_m')* (b_1'^{-1}, b_2'^{-1}, b_m'^{-1})$.
Step 8 Convert $(b_1'^{-1}, b_2'^{-1}, b_m'^{-1})$ modulo to $q^{1'}, q^{2'}, \ldots, q_m'$ into $(b_1''^{-1}, b_2''^{-1}, b_m''^{-1})$ modulo to $q_1, q_2, \ldots, q_m$ and $(c_1', c_2', \ldots, c_m')$ modulo to $q_1', q_2', .., q_m'$ into $(c_1, c_2, \ldots, c_m)$ modulo to $q_1, q_2, \ldots, q_m$.
Step 9 Compute $(a_1, a_2, \ldots, a_m)$ $[(a_1, a_2, \ldots, a_m)*(b_1''^{-1}, b_2''^{-1}, b_m''^{-1})-(c_1, c_2, \ldots, c_m)]*p_1^{-1}, p_2^{-1}, \ldots, p_m^{-1}$
Step 10 Compute $(b_1, b_2, \ldots, b_m)=[(b_1, b_2, \ldots, b_m)*(b_1''^{-1}, b_2''^{-1}, b_m''^{-1})-(1, 1, \ldots, 1)]*(p_1^{-1}, p_2^{-1}, \ldots, p_m^{-1})$.
Step 11 If $(b_1, b_2, \ldots, b_m) = (1, 1, \ldots, 1)$ then $(a_1, a_2, \ldots, a_m)$ is the quotient; otherwise Goto Step 4.

III. MONTGOMERY AND CMM ALGORITHMS

In this section, we present four hardware implementations for computing modular exponentiation using Montgomery's multiplication [8] and CMM method.

A. Montgomery's Modular Reduction

Let the modulus M be an integer within the range $[2^{n-1}, 2^n]$ and let R be 2^n. Montgomery's multiplication algorithm 2 requires R and M to be relatively prime, i.e. $\gcd(R, M)=\gcd(2^n, M)=1$, which is satisfied if M is odd as is required by the algorithm. By exploiting this property, the Montgomery reduction algorithm introduces an efficient multiplication scheme, which computes the modular product, P, of two given integers, A and B, as Equation (2).

$$P = < A\,B\,R^{-1} >_M, \qquad (2)$$

where R^{-1} is the inverse of R modulo M. The computation of the Montgomery multiplication is carried out as follows.

Algorithm 5: (Montgomery's Multiplication)
{
$T=ABP=(T+<TM'>_R M)/R$
if $P \geqq M$ then $P=P-M$
}

B. Montgomery Exponentiation Algorithm

To perform the exponentiation process in the RSA algorithm, we started with the RL binary method and adapted it to be used in the Montgomery algorithm. RL binary exponentiation method is used as shown in Algorithm 7. C is the ciphertext, M is the plaintext and E is the public/private exponent that is k-bit in length. E_i shows the i^{th} bit of the E exponent. N shows the modulus and Z stores the squaring results. For each round squaring is performed, but multiplication is performed if and only if E_i = 1: As seen from Algorithm 2, C_{i+1} does not depend on Z_{i+1}; so that squaring is performed in parallel with the multiplication in the loop [12]. Since Montgomery's modular multiplication algorithm is used for the exponentiation. These calculations increase the encryption/decryption time, but in terms of the whole exponentiation process, it is not so crucial.

Algorithm 6. ModEXP(*M*, *E*, *N*)
Input: ***A, B, N***
Output: ***t***
{
s[0]=0;
for (*i* = 0, l, ..., *n*-1)
{p_i=(*s*[i]+a$_i$*B*) mod 2;
$s[i+1] = \frac{s[i] + a_i B + p_i N}{2}$ }
if *s*[*n*]<*N* ***then*** *t*=*s*[*n*]
else *t*=*s*[*n*]-*N*;
return *t*;
}

C. Common-Multiplicand Multiplication (CMM) Method

In 1993, Yen and Laih [13] developed the method of common-multiplicand multiplication (CMM) to improve the performance of the binary exponentiation method. Here we focus on the computations of $\{X*Yi \mid i = 1, 2, \cdots, t; t \geq 2\}$, where t is the number of division group. An arithmetic method is presented and applies to the binary exponentiation method for the efficient evaluation of exponentiation.

The basic idea of CMM method is to extract the common parts of multiplicands and then save the number of binary additions for the computation of common parts. By using the above method, the computations $\{X*Y_1, X*Y_2\}$ can be represented as $\{X*Y_{1,c}+ X*Y_{common}, X*Y_{2,c}+ X*Y_{common}\}$.

Let both X and Y_i be m-bit integers. On average, the Hamming weights of Y_i, Y_{common} and $Y_{i,c}$ are $\frac{m}{2}$, $\frac{m}{2^t}$, and $\frac{m}{2}-\frac{m}{2^t}$, respectively. The total number of binary additions for the common-multiplicand multiplication evaluation of $\{X*Yi \mid i = 1, 2, \cdots, t; t\geq 2\}$ is $\frac{m}{2^t} + t*(\frac{m}{2}-\frac{m}{2^t})$. Without the CMM algorithm, the evaluation of $\{X*Yi \mid i = 1, 2, \cdots, t; t\geq 2\}$ are computed one after another independently using total $t*\frac{m}{2}$ binary addition. The performance improvement of CMM method can be denoted as:

$$\frac{\frac{m}{2}*t}{\frac{m}{2^t}+t*(\frac{m}{2}-\frac{m}{2^t})} = \frac{t}{t+(1-t)*2^{1-t}}. \quad (1)$$

Based on the Eq. (5) shown above, the optimal performance can be obtained as $\frac{4}{3}$ when t = 2, which implies we need 1.5m (=2m*$\frac{3}{4}$) multiplications by using the CMM method for evaluating $X*Y_1$ and $X*Y_2$.

D. Improved Montgomery Algorithm

Algorithm 7: Montgomery Reduction (MR) Method
Input: ***A, B, N***
Output: X /*$X = ABR^{-1} \bmod N$*/
begin
X = 0
for i = 1 ***to*** m ***do***
begin
$X = X + A_iB$
$Y = X_0 N_0$' mod 2
$X = \frac{X + Y*N}{2}$
end
if ($X \geq N$) **then** $X = X - N$
end

Algorithm 8: (Proposed Method)
Input: M, E_{SD}, N, R /*$E_{SD} = (e_m e_{m-1} \ldots e_2 e_1)_{SD2}$, $R = 2^m$ */
Output: $C_1 = M^{E_{common[1]}}$, $C_2 = M^{E_{1,c[1]}}$, $C_3 = M^{E_{2,c[1]}}$,
$D_1 = M^{E_{common[\bar{1}]}}$, $D_2 = M^{E_{1,c[\bar{1}]}}$, $D_3 = M^{E_{2,c[\bar{1}]}}$
begin
C_1=C_2=C_3=D_1=D_2=D_3=2^m
S=M*R mod N /* R=2^m */
for i = 1 ***to*** m ***do***
begin
if (e_{1i}'=1) ***then*** C_1=MMR(SC_1)
if (e_{1i}'=$\bar{1}$) ***then*** D_1=MMR(SD_1)
if (e_{2i}'=1) ***then*** C_2 = MMR(SC_2)
if (e_{2i}'=$\bar{1}$) ***then*** D_2 = MMR(SD_2)
if (e_{3i}'=1) ***then*** C_3 = MMR(SC_3)
if (e_{3i}'=$\bar{1}$) ***then*** D_3 = MMR(SD_3)
S = MMR(SS)
end
end

Assume we have two n-bits exponents E_1 and E_2, $M^{E_{SD}}$ can be depicted as:

$$M^E = M^{E_1 \| E_2} = M^{E_2 * 2^n} * M^{E_1} \quad (\text{"}\|\text{"} \text{ is the concatenation operator}) \quad (2)$$

Let the decomposition segments E_1 and E_2 be expressed as $E_{1i} = \sum_{i=1}^{m} e_{1i} * 2^i$ and $E_{2i} = \sum_{i=1}^{m} e_{2i} * 2^i$, where e_{1i} and e_{2i} are signed-digits and e_{1i}, $e_{2i} \in \{0, 1, \bar{1}\}$. Moreover, the exponentiations $M^{E_1[1]}$ and $M^{E_2[1]}$ are evaluated for handling positive signed-digit in E_1 and E_2, respectively.

IV. COMPLEXITY ANALYSES

On average, by applying the CMM method and the binary method, exponentiation can be evaluated by using $\frac{(1.5+1)}{2} * m = 1.25m$ multiplications. For evaluating M^E mod N, where both M and E are 512-bit, the binary exponentiation method and the CMM method combined with the binary exponentiation method require 768 multiplications and 640 multiplications, respectively. In the proposed method, based on the computational analyses of Montgomery reduction method, the probability of executing modular multiplication MMR(SC_1), MMR(SC_2), or MMR(SC_3) in the proposed method is all equivalent to the occurrence probability of signed-digit "1" in E_{SD}. Therefore, the total operations MMR(SC_1), MMR(SC_2), and MMR(SC_3) require$(3/6)*[1.5m*(n-2)*(n+1)] = (3/4)*m*(n^2-n-2)$ multiplications, where m is the bit-length of the exponent E and $n=(m/2)$. Similarly, operations MMR(SD_1), MMR(SD_2), and MMR(SD_3) total require $(3/6)*[1.5m*(n-2)*(n+1)]=(3/4)*m*(n^2-n-2)$ multiplications. The operation MMR(SS) requires $(2/3)*[0.5m*(n-2)*(n+1)]=(1/3)*m*(n^2-n-2)$ multiplications.

The Ha-Moon algorithm takes $0.625m^3+m^2$ multiplications and the Dusse-Kalisk algorithm takes $0.75m^3+0.75m$ multiplications [14] for calculating an m-bits exponent exponentiation. However, the proposed method takes $(3/4)*m*(n^2-n-2)+ (3/4)*m*(n^2-n-2)+ (1/3)*m*(n^2-n-2) \approx 0.458m^3-0.916m^2-3.666m$ multiplications. Take a 512-bit exponent E for example to evaluate M^E, the number of the multiplications by using the proposed method is 61,263,141 multiplications, the number of the multiplications by using the Dusse-Kaliski algorithm [10-24] is 100,663,680 multiplications, and the number of the multiplications by using the Ha-Moon algorithm is 84,148,224 multiplications.

V. CONCLUSION AND FUTURE WORKS

We know modular arithmetic is the cornerstone computations performed in public-key cryptosystem. An efficient computation of the modular exponentiation is very important and useful both in smart card and public-key cryptosystems. Now there are still many novel methods issued in many computer security journals] and reports for computer arithmetic operations and theoretical analyses. In the future, we can incorporate modular arithmetic and some advanced techniques to effectively reduce the number of multiplications or accelerate the multiplication itself respectively for modern cryptographic applications [10-25].

REFERENCES

[1] C.-C. Chang and Y.-P. Lai, "A division algorithm for residue numbers," *Applied Mathematics and Computation*, vol. 172, no. 1, pp. 368-378, Jan. 2006.

[2] Steve Wright, "Quadratic residues and the combinatorics of sign multiplication," J*ournal of Number Theory*, vol. 128, no. 4, April 2008, pp. 918-925.

[3] M. Joye and S.-M. Yen, "Optimal left-to-right binary signed-digit recoding," *IEEE Transactions on Computers*, vol. 49, no. 7, pp. 740-748, July 2000.

[4] C. K. Koç, "Analysis of sliding window techniques for exponentiation," *Computers & Mathematics with Applications*, vol. 30, no. 10, pp. 17-24, Nov. 1995.

[5] D. Gamberger, "New approach to integer division in residue number systems," *Proceedings of 10th IEEE symposium on Computer Arithmetic*, Grenoble, France, pp. 84-91, 1991.

[6] D. E. Knuth, *The Art of Computer Programming, vol. II: Seminumerical Algorithms*, 3rd Edition, MA: Addition-Wesley, 1997.

[7] A. J. Menezes, P. C. Oorschot, and S. A. Vanstone, *Handbook of Applied Cryptography*, CRC Press, 1997.

[8] Lee, C.-Y., Lu, E.-H. and Lee, J.-Y., "Bit-parallel systolic multipliers for GF(2_m) fields defined by all-one and equally spaced polynomials," *IEEE Trans. Computers*, vol. 50, no. 5, pp. 385–393, 2001.

[9] L. M. Mourelle and N. Nedjah, "Reconfigurable hardware for addition chains based modular exponentiation," *International Conference on Information Technology: Coding and Computing*, vol. 1, April 2005, pp. 603-607.

[10] N. Nedjah and L. M. Mourelle, "Efficient pre-processing for large window-based modular exponentiation using genetic algorithms," *Proceedings of the 16th International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems (IEA/AIE 2003)*, Loughborough, UK, LNAI 2718, Springer-Verlag, 2003, pp. 625-635.

[11] C. K. Koc, "*High-speed RSA Implementation, Technical report, RSA Laboratories*, Redwood City, California, USA, Nov. 1994.

[12] D. Pearson, "A parallel implementation of RSA selected areas in cryptography," *(SAC '96)*, Kingston, Ontario, August 1996.

[13] C.-L. Wu and W.-L. Wang, "Information superiority infrastructure and information security architecture mechanisms," *3rd Conference of Aviation Technology and Flight Safety*, Sep. 27, 2013, pp. 1-8.

[14] S. R. Dusse and B. S. Kaliski, "A cryptographic library for the Motorola DSP 56000,"*Proceedings of EUROCRYPT'90 on Advance in Cryptology*, LNCS, Springer-Verlag, 1990, vol. 473, pp. 230-244.

[15] J.-C. Ha and S.-J. Moon, "A common-multiplicand method to the Montgomery algorithm for speeding up exponentiation," *Information Processing Letters*, vol. 66, no. 2, pp. 105-107, April 1998.

[16] C.-L. Wu, "Modern defense information technology and information security theories," *Journal of Crisis Management*, vol. 10, no. 2, pp. 1-6, Sep., 2013.

[17] Chia-Long Wu, "Efficient modular algorithm design for modern cryptosystems and complexity analyses on information security applications," *Journal of AFIT*, vol. 12, no. 1, pp. 15-22, Aug. 2013.

[18] S.-M. Yen, "Improved common-multiplicand multiplication and fast exponentiation by exponent decomposition," *IEICE Transactions on Fundamentals*, vol. E80-A, no. 6, pp. 1160-1163, June 1997.

[19] J. Chung and A. M. Hasan, "Low-weight polynomial form integers for efficient modular multiplication," *IEEE Transactions on Computers*, vol. 56, no. 1, pp. 44-57, Jan. 2007.

[20] C.-L. Wu, D.-C. Lou, J.-C. Lai and T.-J. Chang, "Fast modular multi-exponentiation using modified complex arithmetic," *Applied Mathematics and Computation*, vol. 186, no. 2, pp. 1065-1074, March 2007.

[21] G. Chen, B. Guoqiang, and H. Chen, *A new systolic architecture for modular division.* IEEE Transactions on Computers, vol. 56, no. 2, Feb. 2007, pp. 282-286.

[22] W. N. Chelton and M. Benaissa, "Fast Elliptic Curve Cryptography on FPGA," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 16, no. 2, pp. 198-205, Feb. 2008.

[23] D.-Z. Sun, Z.-F. Cao, and Y. Sun, *How to compute modular exponentiation with large operators based on the right-to-left binary algorithm.* Applied Mathematics and Computation, vol. 176, no. 1, May 2006, pp. 280-292.

[24] C.-L. Wu, Complexity analyses and design for cryptographic modular algorithm," 2011 Symposium on Communication Information Technology on Management and Application, A2: Communication Theory, pp. 1-6, 2011.

[25] C.-L. Wu, *Fast Montgomery binary algorithm for information security*, 2011 International Symposium on NCWIA, D6: Information Systems and Innovative Computing, pp. 1-5, 2011.