5th SASTech
Iran |Mashhad
May 12 - 17 | 2011
5th Symposium on Advances in Science & Technology

An Efficient Parallel CMM-CSD Modular Exponentiation Algorithm by Using a New Modified Modular Multiplication Algorithm

Abdalhossein Rezai and Parviz Keshavarzi
Electrical and Computer Engineering Faculty
Semnan University
Semnan, Iran
a_h_rezai@sun.semnan.ac.ir, pkeshavarzi@semnan.ac.ir
Paper Reference Number: 16
Name of the Presenter: Abdalhossein Rezai

Abstract

This paper presents a new modified Montgomery modular multiplication algorithm based on canonical signed-digit (CSD) representation, and sliding window method. In this modified Montgomery modular multiplication algorithm, signed-digit recoding technique is used in order to increase probability of the zero bits. Also sliding window method is used in order to reduce the multiplication steps considerably. In addition, a new efficient modular exponentiation algorithm based on this new modular multiplication algorithm, CMM method and parallel structure is presented. In this new CMM-CSD modular exponentiation algorithm, not only the common part of modular multiplication is computed once rather than several times but also the modular multiplication and modular squaring operations is performed in parallel. Using this new modular exponentiation algorithm, the security of the cryptosystem increased considerably. The results show that the average number of multiplication steps in the proposed CMM-CSD exponentiation algorithm is reduced considerably. Therefore, the efficiency of the proposed CMM-CSD modular exponentiation algorithm increased considerably in compare with Dusse-Kaliski's Montgomery algorithm, Ha-Moon's improved Montgomery algorithm, Wu's CMM-MSD Montgomery algorithm, Wu et al.'s CMM-SDR Montgomery algorithm and Wu's CMM-CSD Montgomery algorithm.

Key words: public-key cryptography, security, fast modular exponentiation, Montgomery modular multiplication, signed-digit recoding.

1. Introduction

In the recent years, the electronic communication and the internet service like email, e-commerce and e-banking have been widely used in our life. So information security becomes an increasing concern. The core technology used for information protection is cryptography. Public-key cryptography (PKC) is the important component of the cryptography (Xiangyan et al. 2009). The modular exponentiation with large modulus is a

crucial operation in many PKCs such as RSA and DSA. This operation is implemented by repeating modular multiplication. So the efficiency of the many PKCs is determined by the efficiency of modular multiplication algorithm and the number of modular multiplication which required in modular exponentiation algorithm (Nedjah and Murelle 2009a, Wu 2009a).

Montgomery modular multiplication algorithm (Montgomery 1985) is an efficient algorithm for modular multiplication because it avoids division by the modulus (Wu 2009a, Nedjah and Murelle 2009a). There are many research efforts in order to speed up the performance of the Montgomery modular multiplication algorithm such as high-radix design (Pinckney and Harris 2008, Tawalbeh et al. 2005), scalable design (Shieh and Lin 2010, Pinkey and Harris 2008), parallel calculation quotient and partial result (Keshavarzi and Harrison 98) and signed-digit recoding (Koc and Hung 92, Philips and Burgess 04).

Also, there are many research efforts in order to reduce the number of modular multiplication such as signed-digit recoding (Wu 2009 a, Wu et al. 2008), CMM method (Ha and Moon 98, Wu 2009a, Wu 2009b, Rezai and Keshavarzi 2011) and sliding window method (Nedjah and Murelle 2009a, Nedjah and Murelle 2009b).

In this paper, we present a new Montgomery modular multiplication algorithm based on sliding window method, multiple bits scan-multiple bits shift technique and signed-digit technique in order to increase the efficiency of the modular multiplication. In addition we proposed using this new modular multiplication in order to increase the efficiency of the modular exponentiation algorithm. In this new modular exponentiation algorithm, we used also CMM method and parallel structure in order to increase the efficiency of the modular exponentiation algorithm. So the efficiency and the security of the cryptosystem which use of the proposed exponentiation algorithm increase considerably.

The rest of this paper is organized as follows: section 2 describes the preliminaries of the proposed algorithms. The proposed modular multiplication algorithm and its application in CMM-CSD modular exponentiation algorithm is presented in section 3. Section 4 evaluates the proposed algorithms. Finally conclusion is given in section 5.

2. Preliminaries

2.1. The Montgomery modular multiplication algorithm

Montgomery (1985) proposed a modular multiplication algorithm which speeds up the modular multiplication and modular exponentiation algorithm by replacing the trial division by the modulus with a simple right shift (Wu 2009a). Algorithm 1 shows the radix-2 Montgomery modular multiplication algorithm.

Algorithm 1: The radix-2 Montgomery modular multiplication algorithm

Input: X,Y,M;
Output: $S(n)=XY2^{-n} \bmod M$
1. S (0):= 0;
   For i = 0 to n-1 **Do**
2. $q_i := (S(i) + x_i Y) \bmod 2$;
3. $S(i+1):= (S(i) + x_i Y + q_i M) / 2$;
4. **If** S (n) ≥ M **Then** S (n) – M

This algorithm computes $S(n) = XY2^{-n} \bmod M$ in n- loop iterations. So it is time-consume operation.

2.2. The modular exponentiation algorithm

As modular exponentiation consists of series of modular multiplications, the performance of the modular multiplication is determined by the efficiency of the implementation of the modular multiplication (Wu 09a, Nedjah and Murelle 2009a). The Montgomery modular exponentiation algorithm is shown in algorithm 2:

Algorithm 2: The Montgomery modular exponentiation algorithm

Input: A,E,R,N;
Output: $C:=A^E \bmod N$;
1. S:= AR mod N, C:=R mod N;
   For i = 0 to k-1 **Do**
2. **If** (e_i=1) **Then** {C:=Mont(S,C),S:=Mont(S,S)};
3. **Else** S:=Mont(SS);
4. C:=Mont(C,1);

In algorithm 2, when the exponent bit is nonzero, both Mont(S,C) and Mont(S,S), are performed. Ha and Moon (1998) proposed the common part in Mont(S,C) and Mont(S,S) can be computed once rather than twice. There are many attempts (Ha and Moon 98, Wu et al. 2008, Wu 2009a, Wu 2009b) in order to speed up the performance of modular exponentiation algorithm based on this idea. One of the recent attempts is the parallel CMM-CSD Montgomery algorithm (Wu 2009b) which is shown in algorithm 3:

Algorithm 3: The Parallel CMM-CSD Montgomery modular exponentiation algorithm

Input: M,E_{CSD},N,R;
Output: $C = M^{E_{CSD}} \bmod N$; $D = M^{E_{CSD}} \bmod N$;
1. S = MR mod N, C = D = R mod N;
   For i = 0 **to** m **Do**
   Parallel begin
2. **If** (e_i = 1) **Then** C = Mont(S,C);
3. **If** (e_i = 1) **Then** D = Mont(S,D);
4. S = Mont(S,S);
   Parallel End;
   Parallel begin
5. C = Mont(C,1);
6. D = Mont(D,1);
   Parallel End;

In algorithm 3, the modular squaring and modular multiplication operations are executed in parallel. Also the registers C and D are used in order to store the operation results of the positive digit and negative digits in exponent E_{CSD} respectively. In addition in this algorithm by using CMM method, the common part of three multiplication is computed just one.

3. The Proposed CMM-CSD Modular Exponentiation

In serial-parallel multiplication, partial result shifts one bit per iteration. Also multiplication by zero bit results in zero, but this multiplication by zero is performed and implemented per iteration. In this paper, we proposed a new modified Montgomery modular multiplication by recoding and then by partitioning the multiplier. This performs multiplication by zero partition with any length in only one-cycle instead of several cycles. The proposed modular multiplication algorithm is shown in algorithm 4.

Algorithm 4: The modified Montgomery modular multiplication algorithm

Input: X,Y, M;
Output: P:=$XY2^{-n}$ mod M;
1. P=0;
{recoding phase}
2. Compute D by performed canonical recoding on X;
parallel begin
{partitioning phase}
3. Building Π(D) using algorithm 5;
4. Let w=# Π(D) ;
{pre-computation phase}
5. Compute and store V_iY
Parallel End
{multiplication phase}
6. **For** i = 0 to w-1 **Do**
7. P:= P +V_iY;
8. m:= $P_0 M'_0 \mod 2^{l_i}$;
9. P:= (P+mM)/ 2^{l_i} ;
10. **If** (P>M) **Then** P=P-M;

In this algorithm, l_i is the length (i.e. the number of bits) of ith partition, #Π(D) is the number of partitioning in the multiplier and V_i is the corresponding partition value of Π(D).

In recoding phase of this new algorithm, the canonical recoding is performed on the multiplier. In partitioning phase, the partitioning is performed on the resulted signed-digit multiplier. The partitioning strategy instrumented in this algorithm scans the multiplier from the least significant digit to the most significant digit according to Algorithm 5. In this strategy, zero windows are allowed to have an arbitrary length, but the maximum length of nonzero windows should be the exacted value of d digit.

Algorithm 5: The Partitioning Algorithm

Input: D,d
Output: Π(D)
1. **ZP:** Check the incoming single digit;
2. **If** it is zero **Then** stay in ZP
3. **Else** go to NZP;
4. **NZP:** Stay in NZP until all d digits are collected;
5. Check the incoming single digit;
6. **If** it is zero **Then** go to ZP
7. **Else** go to NZP;

The transition probability graph of proposed modular multiplication is shown in Fig. 1. This graph is similar to the transition probability graph of the adaptive m-ary segmentation canonical recoding multiplication algorithm in (Koc and Hung 1992, Philips and Burgess 04).

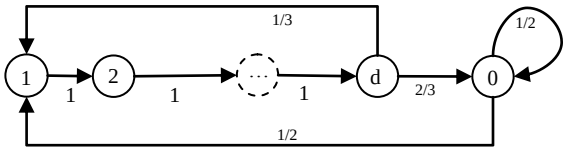

Fig. 1: Transition probability graph for the proposed modular multiplication algorithm

In pre-computation phase of algorithm 4, the least significant digit of nonzero partition is either 1 or $\bar{1}$, which implies that the nonzero partition value is always an odd number. So we require only pre-computation of V_iY for odd value of V_i. Note that the pre-computation phase and the partition phase are performed independently in parallel. This speeds up modular multiplication.

The multiplication phase of algorithm 4 is performed w times. Recall that w denote the number of partitioning in the signed-digit multiplier. In the each iteration of multiplication phase of algorithm 5, l_i bits of multiplier and n-bit multiplicand are processed.

We propose also using this new modular multiplication algorithm in order to speeding up the CMM-CSD Montgomery exponentiation algorithm (Wu 2009b) as shown in algorithm 6.

Algorithm 6: The proposed CMM-CSD Montgomery modular exponentiation algorithm

Input: M, E_{CSD}, N, R;
Output: $C := M^{E_{[1]}} \bmod N$; $D := M^{E_{[\bar{1}]}} \bmod N$;
Parallel begin
1. S: = MR mod N, C := D := R mod N;
2. Compute S_1 by performed steps 2-5 of algorithm 4 on S by one multiplication loop delay;
Parallel End
For i = 0 **to** m **Do**
parallel begin
3. **If** (e_i = 1) **Then** C := S_1C mod N; */* perform steps 6-10 of algorithm 4 for positive signed-digit */*
4. **If** (e_i = 1) **Then** D := S_1D mod N; */* perform steps 6-10 of algorithm 4 for negative signed-digit */*
5. *S:* = S_1S mod N; */* perform steps 6-10 of algorithm 4 */*
6. Compute S_1 by performed steps 2-4 of algorithm 4 on S by one multiplication loop delay;
Parallel End;
Parallel begin
7. C:= C×1 mod N; */* perform algorithm 4 */*
8. D:= D×1 mod N; */* perform algorithm 4 */*
Parallel end;

In this algorithm, for e_i=1, both S_1C mod N and S_1S mod N are computed in parallel. Also for $e_i=\bar{1}$, both S_1D mod N and S_1S mod N are computed in parallel. In addition by using CMM method, the common part of two operations is computed once rather than twice.

In step 1 of this algorithm, S is computed by using algorithm 4. In step 2, S_1 is computed by executing steps 2-5 of algorithm 4 on S by one multiplication loop delay in compare with step 1. In steps 3 and 4 of algorithm 6, $M^{E_{[1]}}$ and $M^{E_{[\bar{1}]}}$ are computed based on value of the e_i. These values are computed by executing steps 6-10 of algorithm 4. In step 5 of algorithm 6, the partial result, S, is computed by executing steps 6-10 of algorithm 4. Also in step 6 of this algorithm S_1 is computed by executing steps 2-4 of algorithm 4. This step is computed by one multiplication loop delay in compare with step 5. The outputs of this algorithm are $C := M^{E_{[1]}}$ and $D := M^{E_{[\bar{1}]}}$. In this algorithm, the exponentiation operation $M^{E_{CSD}}$ is depicted as $M^{E_{CSD}} = C \times D^{-1}$.

4. Evaluation

In the proposed modified Montgomery modular multiplication algorithm, sliding window method is performed on the signed-digit multiplier. So the Hamming weight of the multiplier is $\frac{3n}{3d+4}$. Therefore based on the computational analyses of Ha and Moon (1998), for n-bit modulus, both operations S_1C mod N, and S_1D mod N require

$$(\frac{6n}{3d+4}-2)(2n+1)+2(2n+1)=(\frac{6n}{3d+4})(2n+1)$$

multiplication steps. Also the operation S_1S mod N requires

$$(\frac{6n}{3d+4}-2)n+2(2n+1)=\frac{6n^2}{3d+4}+2n+2$$

multiplication steps. In the proposed CMM-CSD modular exponentiation for $e_i=1$, both S_1C mod N and S_1S mod N are performed in parallel. Also for $e_i=\bar{1}$, both S_1D mod N and S_1S mod N are performed in parallel. In addition we use of radix-2 signed-digit exponent. So the probability of digits "0", "1" and "-1" is 2/3, 1/6 and 1/6 respectively. Therefore the proposed modular exponentiation algorithm for k-bit exponent takes

$$\frac{2}{3}k[(\frac{6n}{3d+4})(2n+1)]+\frac{1}{3}k[(\frac{6n}{3d+4})(2n+1)]=k(\frac{12n^2}{3d+4}+\frac{6n}{3d+4})$$

multiplication steps, however the Dusse and Kaliski's exponentiation algorithm(Dusse and Kaliski 1990), the Ha-Moon's improved Montgomery exponentiation algorithm (Ha and Moon 1998), the Wu et al.'s CMM-MSD algorithm (Wu et al. 2008), Wu's improved CMM-MSD exponentiation algorithm (Wu 2009a) and Wu's parallel CMM-CSD exponentiation algorithm (Wu 2009b) require $1.5k(2n^2+n)$, $0.5k(5n^2+4n)$, $0.5k(2n^2+2n+0.75)$, $1.833k(n^2-n-2)$ and $0.5k(2n^2+2n+5.33)$ multiplication steps respectively. So, the proposed modular exponentiation algorithm reduces the overall number of multiplication steps on average at about

$$1-\frac{k(\frac{12n^2}{3d+4}+\frac{6n}{3d+4})}{1.5k(2n^2+n)}\approx 1-\frac{4}{3d+4}$$

$$1-\frac{k(\frac{12n^2}{3d+4}+\frac{6n}{3d+4})}{0.5k(5n^2+4n)}\approx 1-\frac{12}{2.5(3d+4)}$$

$$1-\frac{k(\frac{12n^2}{3d+4}+\frac{6n}{3d+4})}{1.833k(n^2-n-2)}\approx 1-\frac{12}{1.833(3d+4)}$$

$$1-\frac{k(\frac{12n^2}{3d+4}+\frac{6n}{3d+4})}{0.5k(2n^2+2n+0.75)}\approx 1-\frac{12}{3d+4}$$

$$1-\frac{k(\frac{12n^2}{3d+4}+\frac{6n}{3d+4})}{0.5k(2n^2+2n+5.33)}\approx 1-\frac{12}{3d+4}$$

in compare with (Dusse and Kaliski 1990, Ha and Moon 1998, Wu 2009a, Wu et al. 2008, Wu 2009b) respectively.

We summarize the multiplication steps improvement for the proposed CMM-CSD modular exponentiation algorithm over exponentiation algorithm in (Dusse and Kaliski 1990, Ha and Moon 1998, Wu 2009a, Wu et al. 2008, Wu 2009b) for various window widths in Table 1.

Window width	Improvement percentage			
	Dusse and Kaliski (1990)	Ha and Moon (1998)	Wu(2009a)	Wu et al.(2008), Wu (2009b)
d=3	69.2%	63.1%	49.6%	7.7%
d=4	75%	70%	59.1%	25%
d=5	78.9%	74.7%	65.5%	36.8%
d=6	81.8%	78.2%	70.2%	45.5%
d=7	84%	80.8%	73.8%	52%
d=8	85.7%	82.9%	76.6%	57.1%
d=9	87.1%	84.5%	78.9%	61.3%
d=10	88.2%	85.9%	80.7%	64.7%

Table 1: Multiplication step improvement of the proposed CMM-CSD modular exponentiation algorithm

These results are represented graphically in Fig. 2.

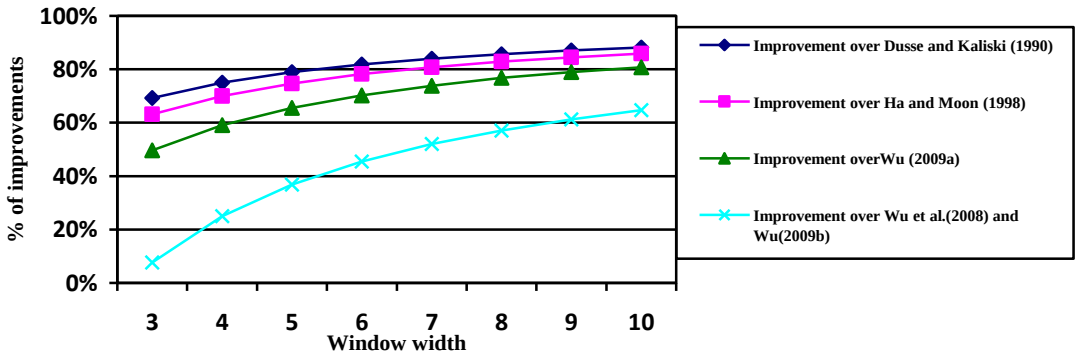

Fig. 2: Multiplication step improvement of the proposed CMM-CSD modular exponentaition algorithm

As it is shown in Table 1 and Fig. 2, the proposed modular exponentiation algorithm reduces the multiplication steps considerably.

The results show that this new CMM-CSD exponentiation algorithm reduces the number of multiplication steps on average at about 69.2%-88.2%, 63.1%-85.9%, 7.7%-64.7%, 49.6%-80.7% and 7.7%-64.7% in compare with Dusse and Kaliski (1990), Ha and Moon (1998), Wu et al. (2008), Wu (2009a) and Wu (2009b) respectively for d=3-10.

5. Conclusions

This paper presents a new efficient modular exponentiation algorithm based on a new modified Montgomery modular multiplication. Also this new modular exponentiation uses other techniques such as CMM method and parallel processing. In the proposed modular multiplication algorithm, by performing sliding window method on signed-digit multiplier, the Hamming weight of multiplier is reduced considerably. Also by skipping from zero digit multiplication and the following required addition, the efficiency of modular multiplication is increased considerably. In the proposed modular exponentiation algorithm, by using the CMM method, the common part of the modular multiplication is computed once rather than

several times. Also by using the parallel processing, the speed of the proposed modular exponentiation increases considerably. Using this new modular exponentiation algorithm, the security of the cryptosystem increased considerably. The results show that the average number of multiplication steps in the proposed CMM-CSD exponentiation algorithm is reduced at about 69.2%-88.2%, 63.1%-85.9%, 7.7%-64.7%, 49.6%-80.7% and 7.7%-64.7% in compare with Dusse and Kaliski (1990), Ha and Moon (1998), Wu et al. (2008), Wu (2009a) and Wu (2009b) respectively for d=3-10.

References

Dusse, S. R., Kaliski, B. S. (1990), A cryptographic library for the Motorola DSP 56000, *in Proceedings of the advance in cryptology UROCRYPT'90, LNCS*, 73,230-244.

Ha, J. C., and Moon S. J. (1998), A common-multiplicand method to the Montgomery algorithm for speeding up exponentiation, *Information processing letters*, 66(2),105–107.

Keshavarzi, P. and Harrison, C. (1998), A new modular multiplication algorithm for VLSI implementation of public-key cryptography, *in proceedings of First international symposium on communication systems and digital signal processing*, 516-519.

Koc, C.K., and Hung, C.Y. (1992), Adaptive m-ary segmentation and canonical recoding algorithms for multiplication of large binary numbers, *computer mathematic application*, 24, 3-12.

Montgomery, P. L. (1985), Modular multiplication without trial division, *Mathematics of computation*, 44, 519-521.

Nedjah, N., and Mourelle, L.M. (2009a), A hardware/software co-design versus hardware-only implementation of modular exponentiation using the sliding-window method, *Journal of circuts, systems and computers*, 18, 295-310.

Nedjah, N., and Mouller, L.M. (2009b), High-performance hardware of the sliding-window method for parallel computation of modular exponentitions, *International journal of parallel programming*, Springer Netherlands, 37, 537-555.

Philips, B., and Burgess, N. (2004), Minimal weight digit set conversions, *IEEE Transaction on computers*, 53, 666-677.

Pinckney, N., and Harris, D. (2008), Parallelized radix-4 scalable Montgomery multipliers, *Journal of Integrated Circuits and Systems*, 3(1), 39-45.

Rezai, A., and Keshavarzi, P. (2011), High-performance Modular Exponentiation Algorithm by Using a New Modified Modular Multiplication Algorithm and Common-Multiplicand-Multiplication Method, *in Proceedings of world congress on internet security*, in press.

Shieh, M., and Lin, W. (2010), Word-based Montgomery modular multiplication algorithm for low-latency scalable architectures, *IEEE Transaction on computers*, 59(8), 1145-1151.

Tawalbeh, L. A., Tenca, A. F., and Koc, C. K. (2005), A radix-4 scalable design, *IEEE Potentials*, 24(2),16-18.

Wu, C. (2009a), An efficient common-multiplicand-multiplication method to the Montgomery algorithm for speeding up exponentiation, *Information Sciences*, 179, 410-421.

Wu, C. (2009b), Fast parallel Montgomery binary exponentiation algorithm using canonical signed-digit recoding technique, *Algorithms and Architectures for Parallel Processing, LNCS* 5574, 428-438.

Wu, C., Lou, D. and Chang, T. (2008), An efficient Montgomery exponentiation algorithm for public-key cryptosystem, *In proceedings of IEEE international conference on intelligence and security information*, 284-285.

Xiangyan, F., Jiahang, Z., Tinggang, X., and Youguang, Y. (2009), The researcher and implement of high-speed modular multiplication algorithm basing on parallel pipelining, *In Proceedings of the Asia-Pacific Conference on Information Processing*, 1,398-403.