Computational Sciences and Engineering 2(2) (2022) 227-238

University of Guilan

Computational Sciences and Engineering

journal homepage: https://cse.guilan.ac.ir/

Algorithm Design and Theoretical Analysis of a New Bit Forwarding Large Integer Modular Exponentiation Algorithm

Manizhe Abbasi [a], Abdalhossein Rezai [b,*], Asghar Karimi [a]

[a] ACECR Institute of Higher Education, Isfahan Branch, Isfahan, Iran

[b] Department of Electrical Engineering, University of Science and Culture, Tehran, Iran

ARTICLE INFO

Article history:
Received 1 January 2023
Received in revised form 23 January 2023
Accepted 13 February 2023
Available online 13 February 2023

Keywords:
Modular exponentiation
Bit forwarding technique
Modular multiplication
Complexity analysis
Multibit-scan-multibit-shift technique

ABSTRACT

One of the most principal operations in many PKCs is Modular Exponentiation (ME). This operation is usually performed by successive modular multiplications. So, the efficiency of these PKCs is released on the efficiency of the Modular Multiplication (M2) and modular exponentiation implementation. Therefore, it is essential to minimize the execution time of the M2 and the number of required M2 for performing the ME operation. This paper proposes a novel ME algorithm. In the developed algorithm, the Bit Forwarding (BF) and multibit-scan-multibit-shift techniques are employed for the performance improvement in the ME operation. The complexity analysis is accomplished to show that the developed exponentiation algorithm has benefit in the number of required multiplications. The results indicate that the presented algorithm improves the results compared to other modular exponentiation algorithms by about 11%-85%.

1. Introduction

Cornerstone operation in many computers arithmetic fields such as Public-Key Cryptography (PKC) is Modular Exponentiation (ME), $C = M^E$, wherein E is called exponent and N called modulus.

The ME includes a series of repeated Modular Multiplications (M2s). Therefore, the efficiency of the ME is affected by efficient execution of the M2s [3]. As a result, the performance of many computers arithmetic operations is determined by the performance of the M2 operation and the required M2 count [11,12].

* Corresponding author.
E-mail addresses: rezai@usc.ac.ir; mzhabbasi@gmail.com; karimi@jdeihe.ac.ir

https://doi.org/10.22124/cse.2023.23755.1041

The Montgomery M2 algorithm is the most commonly used algorithm for efficient implementation of M2 [7]. It can perform the trial division by using shift and binary addition operations, which can implement using simple hardware, but it consumes a lot of time.

There are several techniques and architectures to improve the performance of the multiplication method such as high-radix [6, 8, 13, 14], systolic array [19,24], carry-save addition [13,17], and scalable architecture [5,15].

Recently, Rezai and Keshavarzi proposed a novel integer representation based on the Modified Canonical Recoding (MCR) technique for the multiplier in [13]. Using this representation for multiplier in the multiplication, three operations, including sequence multiplication by zero bits, required additions, and the non-zero digit multiplication are calculated in one clock cycle using multibit-shift technique and one binary addition. Hamming weight of this representation for the multiplier is n/3, that leads to significant reduction of required partial multiplications. They improved this M2 algorithm in [14] and presented compact SD M2 algorithm.

On the other hand, binary method is most widely used and well-recognized method to compute a ME, which employs the binary representation of the exponent E and consists of a long chain of squaring and multiplication operations. In recent years, many new and hybrid methods for implementation of ME are created including, sliding window method [10], M-ary method [9], Common-Multiplicand-Multiplication (CMM) method [4, 11, 12, 16, 21, 22, 23], signed-digit recoding technique [2, 14, 20, 22] and Bit Forwarding (BF) technique [18].

Recently, Vollala et. al. [18] have offered bit forwarding techniques for decreasing the required M2s count in the ME. They also provided Montgomery M2 based on BF technique in radix-2 and high-radix versions.

This study develops and evaluates novel exponentiation algorithm. The proposed algorithm utilizes the BF technique for decreasing the required multiplications in ME and utilizes the multibit-scan-multibit-shift technique in the multiplier for reducing the complexity of the M2 in the ME operation. The complexity analysis show that the developed ME algorithm has advantages compared to other ME algorithms.

The content of this paper is structured as follows: Section 2 explains the background of the proposed ME. Section 3 presents the developed ME algorithm. Complexity analysis and comparison results are provided in section 4. Conclusion of this research is afforded in section 5.

2. Background

2.1. Montgomery M2

Montgomery M2 algorithm is a method for multiplying two integers modulo M. This algorithm computes $X.Y.(2^{-n}) \bmod M$, while avoiding division by M. The main idea in this algorithm is to transform the integers to the m-residues and then compute the multiplication. Finally, the achieved results are transformed back. Algorithm 1 displays the simplest Montgomery M2 algorithm [7, 11, 12].

Algorithm 1. The Radix-2 Montgomery M2 algorithm (Mont(X, Y, M)) [7]

Input : X,Y,M;
Output: $S(n) := X.Y.2^{-n} \bmod M$;

1. S(n)=0
2. For i=0 TO n-1
3. $q_i = (S(i) + x_i \cdot Y) \bmod 2$;
4. $S(i+1) = (S(i)+x_i.Y+q_i.M)/2$;
5. End for
6. IF $S(n) \geq M$ Then S(n)-M;
7. Return S(n);

The n-bit integers multiplier, multiplicand, and modulus are indicated by X, Y, and M, respectively. It should be noted that M is an odd integer. The $S(n)=X.Y.2^{-n} \bmod M$, indicates the output. Although this algorithm is simple for hardware implementation, it is a time-consuming algorithm. Therefore, several improvements have been proposed for this algorithm [7, 8, 13, 14, 19, 24].

Montgomery M2 algorithm can improve using new hybrid architectures such as combination Carry Save Adder (CSA) architecture with high-radix technique [6, 8, 11, 12]. High-radix M2 process several bits at each clock cycle instead of several clock cycles. Algorithm 2 displays the high-radix CSA Montgomery M2 algorithm [12].

Algorithm 2. The high-radix CSA Montgomery M2 algorithm [12]

Input : X,Y,M;
Output: $S(n) := X.Y.2^{-n} \bmod M$;

1. $S_C(0)=0, S_S(0)=0$;
2. For i=0 TO $(n/r)-1$
3. $q_i = (S_C(i) + S_s(i) + X^{(i)} \cdot Y)(2^{r+1} - M^{-1}_{r\ldots0}) \bmod 2^r$;
4. $S_C(i+1), S_S(i+1) = (S_C(i) + S_s(i) + X^{(i)} \cdot Y + q^{(i)}.M)/\ 2^r$;
5. End for
6. $S(n) = S_C(n) + S_S(n)$;
7. IF $S(n) \geq M$ Then S(n)-M;
8. Return S(n);

In algorithm 2, the number of required clock cycles for radix-2^r is also reduced to n/r [17].

Recently, Rezai and Keshavarzi [13] have offered Variable Length Montgomery M2 (VLM3) algorithm. This algorithm is able to decrease the critical path in the high-radix CSA Montgomery M2 algorithm by relaxing the high-radix partial multiplication. They used the MCR representation [12] for X in the M2 algorithm. With this innovation, the high-radix partial multiplication $X^{(i)}.Y$ is simplified to the binary partial multiplication. Algorithm 3 shows the VLM3 algorithm.

Algorithm 3. The VLM3 algorithm (VLM3 (X_{MCR},Y,M))[13]

Input: $X_{MCR}(k_i.f^{(i)}), Y, M;$

Output: $S=X.Y.2^{-n} \bmod M;$

1. $Sc(0)=0, Ss(0)=0;$
2. For i=0 TO J
3. $a^{(i)} = f^{(i)}$
4. If $f^{(i)} = 3$ Then $((Pc(i),Ps(i))=Sc(i)+Ss(i), a^{(i)} = a^{(i)} - 1;$
5. Else
6. If $k_i = 0$ Then $(Pc(i), Ps(i)) = Sc(i) + Ss(i) + 2^{a^{(i)}}.Y;$
7. Else $(Pc(i).Ps(i)) := Sc(i) + Ss(i) - 2^{a^{(i)}} \cdot Y;$
8. $q^{(i)} = \left((Pc(i) + Ps(i))_{K.....0}\right)(2^{a^{(i)}+1}\text{-}M_{k....0}{}^{-1}) \bmod 2^{a^{(i)}+1}$
9. $(Sc(i+1),Ss(i+1))=(Pc(i)+Ps(i)+q^{(i)}.M)/2^{a^{(i)}+1};$
10. End for
11. $S(n)=S_C(j+1)+S_S(j+1);$
12. If $S(n) \geq M$ then $S(n) = S(n) - M;$
13. Return S(n)

Inputs of this algorithm are X_MCR, Y, and M, where X_MCR shows the multiplier in the MCR representation [12], but multiplicand and modulus are n-bit integers. There are two terms in X_{MCR} representation, k_i, which is used to denote the sign of non-zero bit in every digit, and $f^{(i)}$, which is used to denote the length of the digit. In addition, P_c and P_s are carry and sum components of P [13].

2.2. ME algorithm

The most commonly used method for implementing ME are Right-to-Left (R2L) and Left-to-Right (L2R) methods. The L2R ME method is presented in algorithm 4.

Algorithm 4. The L2R ME algorithm [13]

Input: N. M. E;

Output: $C = M^E \bmod N;$

{pre − computation phase}

1. $M=Mont(M, R^2.N);$
2. $S=R \bmod N;$

{exponentiation phase}

3. For i= 0 To k_e-1
4. If $(e_i = 1)$ Then S=Mont(M, S, N)
5. M=Mont(M, M , N);
6. End for

{post − computation phase}

7. C=Mont(S, 1, N);
8. Return C;

This algorithm computes M^E mod N. The exponent bits are scanned from left to right, square and multiply operations are serially executed. There are 1.5k+2 multiplication operations to execute the ME [13].

The R2L ME algorithm is another way to evaluate M^E mod N [13]. In this algorithm, exponent E is processed from right to left. Multiply and square operations can accomplish in parallel [13]. Therefore, k+2 multiplication operations are required to execute the ME algorithm. This algorithm is detailed in algorithm 5[13].

Algorithm 5. The R2L ME Algorithm [13]

Input: N. M. E;
Output: $C = M^E \bmod N$;
{pre − computation phase}
1. M=Mont(M,R^2. N);
2. $S = R \bmod N$;
{exponentiation phase}
3. For i=k_e-1 To 0
4. S=Mont (S, S, N);
5. If ($e_i = 1$) Then S=Mont(M, S, N)
6. End for
{post − computation phase}
7. C=Mont(S, 1, N);
8. Return C;

Recently, Vollala et. al. [18] have presented the BF techniques to decrease the required M2 in the ME. The objective of the presented BF techniques [18] is to optimize the efficiency by decreasing the utilized clock cycles. This method can improve the performance of the ME algorithms. The BF ME algorithm is displays in algorithm 6 [18].

Algorithm 6. The BF1-Bit Forwarding 1-bit [18]

Input: M. E($e_{k-1}. e_{k-2}.\cdots. e_1. e_0$). $PC = 2^{2n}$
Output: $R = M^E \bmod N$
1. M_1 = AMM (M, PC, N);
2. M_2 = AMM (M_1, M_1, N);
3. M_3 = AMM (M_2, M_1, N);
4. R [k - 1] = M_1;
5. for i = k - 2 Down to 0 ;
6. R [i] = AMM (R [i + 1], R [i + 1], N);
7. if (($e_i \neq 0$) AND ($e_{i-1} \neq 0$)) then
8. i = i - 1; // Forwarding one bit //
9. R [i] = AMM (R [i + 1], R [i + 1], N);
10. R [i] = AMM (R [i], M_3, N) ;
11. else if ($e_i \neq 0$) then
12. R [i] = AMM (R [i], M_1, N);
13. end if
14. end for
15. Res = AMM (R [0], 1, N);
16. Return Res

In algorithm 6, AMM is recognized that as customized Montgomery multiplication algorithm according to BF technique [18]. In the BF technique, for every pair of ones in the exponent, one bit is forwarded and then the partial production is multiplied with $M_{2^{c-1}} = M^{2^{c-1}} \bmod N$. Moreover, M_1 value and a cubic value $M_3 = M^3 \bmod N$ have been pre-computed and stored in registers.

3. The proposed algorithms for ME

This section presents novel modified ME algorithms to evaluate $M^E \bmod N$. The proposed algorithms use the VLM3 algorithm, MCR representation and BF technique. The proposed algorithms, namely BF-MCR1 and BF-MCR2 mention to bit forwarding 1-bit and bit forwarding 2-bits, respectively. The proposed algorithms have the following characteristics:

a) The MCR representation is applied to multiplier. The encoded multiplier guarantees minimum Hamming weight. However, format conversions from the binary into its MCR in the proposed algorithms are performed.
b) The performance of the binary algorithm can be improved using the BF technique. This technique can decrease the required M2 using forwarding consecutive ones in the exponent.
c) The multiplications are performed using the VLM3 algorithm, in which the ME algorithm take less clock cycles.

3.1. The proposed BF-MCR1 ME algorithm

The proposed BF-MCR1 ME algorithm, which employs the VLM3 algorithm, is explained in algorithm 7.

Algorithm 7. The proposed BF-MCR1 ME algorithm

Input : M, E, N
Output : $T_{re}= M^E \bmod N$
{pre − computation phase}
1. convert 1, R^2 from binary representation to the MCR representation using algorithm 5 in [12], 1_{MCR}. R^2_{MCR};
2. M*=VLM3(M, R^2_{MCR}, N)
3. convert M* from binary representation to the MCR representation, M_{MCR};
4. M_2= VLM3(M*, M_{MCR},N);
5. convert M_2 from binary representation to the MCR representation, M_{2MCR};
6. M_3 = VLM3(M*, M_{2MCR},N);
7. convert M_3 from binary representation to the MCR representation, M_{3MCR}

{exponentiation phase}
8. T[k-1]=M*;
9. For i=k − 2 down to 0 do
10. T[i]=VLM3 (T[i+1], M_{MCR},N);
11. If (($e_i \neq 0$) *AND* ($e_{i-1} \neq 0$))
12. i=i-1; //forwarding one bit
13. Convert T[i+1] from binary representation to the MCR representation, T_{1MCR};
14. T[i]= VLM3 (T[i+1], T_{1MCR},N);
15. T[i]= VLM3 (T[i], M_{3MCR}, N);
16. ELSEIF ($e_i \neq 0$) then
17. T[i]=VLM3 (T[i], M_{MCR}, N);
18. End if
19. End for
20. T_{re}=VLM3(T[0],1_{MCR}, N);
21. Return T_{re}

In this algorithm, M* denotes M in Montgomery domain, M_2 and M_3 denote $M^2 \bmod N$ and $M^3 \bmod N$, respectively. In the exponentiation phase, the T[i+1] in the binary representation is converted to the MCR representation in step 13. Moreover, a pair of successive 1's in the exponent is scanned independently. If it is identified, then one bit will forward, and the partial result is multiplied with M_3 as described in Algorithm 7.

3.2. The proposed BF-MCR2 ME algorithm

The proposed BF-MCR2 ME algorithm, which employs the VLM3 algorithm, is explained in algorithm 8.

Algorithm 8 : The proposed BF-MCR2 ME algorithm

Input : M, E= $(e_{k-1}, e_{k-2},\ldots,e_2,e_1,e_0)_2$, N

Output : $T_{re} = M^E \bmod N$

{pre-computation phase}

1. Convert 1, R^2 from binary representation to the MCR representation, 1_{MCR}, R^2_{MCR};
2. M*=VLM3(M, R^2_{MCR}, N)
3. convert M* from binary representation to the MCR representation, M_{MCR};
4. M_2= VLM3(M*, M_{MCR}, N);
5. convert M_2 from binary representation to the MCR representation, M_{2CSD};
6. M_3 = VLM3(M*,M_{2MCR},N);
7. convert M_3 from binary representation to the MCR representation, M_{3MCR};
8. M_6=VLM3(M_3, M_{3MCR}, N);
9. convert M_6 from binary representation to the MCR representation, M_{6MCR};
10. M_7=VLM3(M*, M_{6MCR}, N);
11. convert M_7 from binary representation to the MCR representation, M_{7MCR};

{exponentiation phase}

12. T[K-1]=M*;
13. For i=k-2 down to 0 do
14. T[i]=VLM3(T[i+1], M_{MCR},N);
15. If (($e_i \neq 0$) AND ($e_{i-1} \neq 0$) AND ($e_{i-2} \neq 0$)) then
16. i=i-1;
17. Convert T[i+1] from binary representation to the MCR representation, T_{1MCR};
18. T[i]=VLM3(T[i+1], T_{1MCR}. N);
19. i=i-1;
20. Convert T[i+1] from binary representation to the MCR representation , T_{2MCR};
21. T[i]=VLM3(T[i+1], T_{2MCR}. N);
22. T[i]=VLM3(T[i], M_{7MCR}. N);
23. Elseif(($e_i \neq 0$) AND ($e_{i-1} \neq 0$)) then
24. i=i-1;
25. Convert T[i+1] from binary representation to the MCR representation , T_{3MCR}
26. T[i]=VLM3(T[i+1], T_{3MCR} . N);
27. T[i]=VLM3(T[i], M_{3MCR}. N);
28. Elseif ($e_i \neq 0$) then
29. T[i]=VLM3(T[i], M_{MCR}. N);
30. End if
31. End for
32. T_{re}=VLM3(T[0],1_{MCR}. N);
33. Return T_{re}

As given in algorithm 8, M*, M_2, M_3, M_6, M_7 denote M in Montgomery domain, M^2mod N, M^3 mod N, M^6 mod N, and M^7 mod N, respectively. M_3 and M_7 values need to pre-compute and store in two registers. To reduce the M2s, the BF-MCR2 checks for three successive 1's in the exponent, if it is identified, then 2 bits is forwarded, and the result is multiplied by M_7. The BF-MCR2 also checks for a pair of successive 1's in the exponent. If it is identified, then one bit will be forwarded, and the partial result will be multiplied by M_3.

4. Complexity analysis of the developed ME algorithms

In this section, we provide the theoretical analysis for efficiency of the proposed BF-MCR ME algorithms.

Based on [18], for k-bit exponent and kτ non-zero bits in the exponent, with c pairs of consecutive ones, the required multiplication steps (clock cycles) in the BF-MCR1 algorithm is n*(n/3+6)*(k+kτ -c+1), where n denotes the modulus bit length.

Similarly, the average number of required multiplication steps for BF-MCR2 algorithm is n*(n/3+6)*(k+kτ -c-2·d+4), where d is the number of three successive ones.

However, the Montgomery ME algorithms such as [18] (for BF1) and [18] (for BF2) require n*(n+3)*(k+kτ -c) and n(n+1)(k+kτ -c-2·d+4) multiplication steps respectively as shown in Table 1.

Table 1. The average number of required multiplication steps in ME algorithms

Reference	Required multiplication steps (clock cycles)
[1]	$1.5k*(2n^2+n)$
[4]	$0.5k*(5n^2+4n)$
[12]	$0.611k*(n^2-5n-3)$
[18] (for BF1)	$n*(n+3)*(k+k\tau-c+1)$
[18] (for BF2)	$n*(n+3)*(k+k\tau-c-2\cdot d+4)$
[22]	$1.833k*(n^2-n-2)$
This paper (for BF-MCR1)	$n*\left(\frac{n}{3}+6\right)*(k+k\tau-c)$
This paper (for BF-MCR2)	$n*\left(\frac{n}{3}+6\right)*(k+k\tau-c-2\cdot d+4)$

Table 2 summarizes the expected number of M2s for the different exponentiation algorithms.

Table 2. The expected number of M2s for exponentiation algorithms

Reference	n=k		
	512	1024	2048
[1]	403046400	3222798336	$2.577609523\times 10^{10}$
[4]	338165760	2686451712	$2.151677952\times 10^{10}$
[22]	245538708	1966242970	$1.573765444\times 10^{10}$
[12]	81204304	652849100	5235628929
[18] (for BF1)	180515328	1437003377	$1.146256054\times 10^{10}$
[18] (for BF2)	172963532	1370623355	$1.082394233\times 10^{10}$
This paper (for BF-MCR1)	61924352	485997247	3848797348
This paper (for BF-MCR2)	59333768	463547398	3634065572

Table 3 and table 4 summarize the multiplication steps improvements for the BF-MCR1 and BF-MCR2 ME algorithms in comparison with the ME algorithms in [1, 4, 12, 18, 22] for 2048-bit exponent. It should be noted that values in table 2 are calculated using Table II, and Table III in [18].

Table 3. The multiplication steps improvement (for BF-MCR1)

Reference	Improvement (%)
[1]	85
[4]	82.1
[22]	75.5
[12]	26.5
[18] (for BF1)	66.4

Table 4. The multiplication steps improvement (for BF-MCR2)

Reference	Improvement (%)
[1]	85.9
[4]	83.1
[22]	76.9
[12]	30.6
[18] (for BF2)	66.4

Based on our analysis, which are shown in table 3, the BF-MCR1 algorithm are able to decrease the required M2 for 2048-bit exponent in comparison with [1, 4, 12, 18, 22] by about 85%, 82.1%, 26.5%, 66.4%, and 75.5%, respectively.

Similarly, based on our analysis, which are displayed in table 4, the BF-MCR2 algorithm are able to decrease the required M2 for 2048-bit exponent in comparison with [1, 4, 12, 18, 22] by about 85.9%, 83.1%, 30.6%, 66.4%, and 76.9%, respectively.

In addition, in the developed BF-MCR1 ME algorithm, the average required multiplication steps is decreased in comparison with [1, 4, 12, 18 (for BF1), 22] by about

$$1-\frac{\mathrm{n}*\left(\frac{\mathrm{n}}{3}+6\right)*(\mathrm{k}+k\tau\ -\mathrm{c}+1)}{1.5\mathrm{k}*(2n^2+\mathrm{n})}*100\approx 85\ \%$$

$$1-\frac{\mathrm{n}*\left(\frac{\mathrm{n}}{3}+6\right)*(\mathrm{k}+k\tau\ -\mathrm{c}+1)}{0.5\mathrm{k}*(5n^2+4\mathrm{n})}*100\approx 82\ \%$$

$$1-\frac{\mathrm{n}*\left(\frac{\mathrm{n}}{3}+6\right)*(\mathrm{k}+k\tau\ -\mathrm{c}+1)}{0.611\mathrm{k}*(n^2-5n-6)}*100\approx 29\ \%$$

$$1-\frac{\mathrm{n}*\left(\frac{\mathrm{n}}{3}+6\right)*(\mathrm{k}+k\tau\ -\mathrm{c}+1)}{\mathrm{n}*(\mathrm{n}+3)*(\mathrm{k}+k\tau\ -\mathrm{c}+1)}*100\approx 66.7\ \%$$

$$1-\frac{\mathrm{n}*\left(\frac{\mathrm{n}}{3}+6\right)*(\mathrm{k}+k\tau\ -\mathrm{c}+1)}{1.833\mathrm{k}*(n^2-n-2)}*100\approx 76\ \%$$

Similarly, in the BF-MCR2 ME algorithm, the average required multiplication steps is decreased in comparison with [1, 4, 12, 18 (for BF2), 22] by about

$$1-\frac{\mathrm{n}*\left(\frac{\mathrm{n}}{3}+6\right)*(\mathrm{k}+k\tau\ -\mathrm{c}-2\cdot\mathrm{d}+4)}{1.5\mathrm{k}*(2n^2+\mathrm{n})}*100\approx 86\ \%$$

$$1-\frac{\mathrm{n}*\left(\frac{\mathrm{n}}{3}+6\right)*(\mathrm{k}+k\tau\ -\mathrm{c}-2\cdot\mathrm{d}+4)}{0.5\mathrm{k}*(5n^2+4\mathrm{n})}*100\approx 83\ \%$$

$$1-\frac{\mathrm{n}*\left(\frac{\mathrm{n}}{3}+6\right)*(\mathrm{k}+k\tau\ -\mathrm{c}-2\cdot\mathrm{d}+4)}{0.611\mathrm{k}*(n^2-5n-6)}*100\approx 31\ \%$$

$$1-\frac{\mathrm{n}*\left(\frac{\mathrm{n}}{3}+6\right)*(\mathrm{k}+k\tau\ -\mathrm{c}-2\cdot\mathrm{d}+4)}{\mathrm{n}*(\mathrm{n}+3)*(\mathrm{k}+k\tau\ -\mathrm{c}-2\cdot\mathrm{d}+4)}*100\approx 66.7\ \%$$

$$1-\frac{\mathrm{n}*\left(\frac{\mathrm{n}}{3}+6\right)*(\mathrm{k}+k\tau\ -\mathrm{c}-2\cdot\mathrm{d}+4)}{1.833\mathrm{k}*(n^2-n-2)}*100\approx 77\ \%$$

Based on our analysis, the developed ME algorithms reduce the multiplication steps considerably regarding to other ME algorithms in [1, 4, 12, 18 22].

5. Conclusion

In this study, two algorithms are presented to evaluate M^E mod N, which are named BF-MCR1 and BF-MCR2. These novel algorithms provide improvements in terms of the number of required multiplication steps. These improvements in the number of required multiplications are obtained by performing the MCR representation, the VLM3 algorithm and BF technique. So, the number of required partial multiplications are considerably decreased, which have a good impact on the efficiency of the proposed ME algorithms. Moreover, by performing the BF technique in the BF-MCR ME algorithms using forwarding consecutive ones in the exponent, the efficiency of the ME methods is improved. Our complexity analysis indicate that the average required multiplication steps in the developed BF-MCR1 ME algorithm are improved for 2048-bit exponent in comparison with [1, 4, 12, 18, 22] by about 85%, 82.1%, 26.5%, 66.4%, and 75.5%, respectively. Moreover, the average required multiplication steps in the developed BF-MCR2 ME algorithm are improved for 2048-bit exponent in comparison with [1, 4, 12, 18, 22] by about 85.9%, 83.1%, 30.6%, 66.4%, 76.9%, respectively.

References

[1] S. R. Dusse and B. S Kaliski, A cryptographic library for the Motorola DSP 56000. Proc. Of Adv. Cryptol. EUROCRYPT'90. 73 (1990) 230–244.
[2] O. Egecioglu and C. K. Koc, Exponentiation using Canonical Recoding. Theoret. Comput.Sci. 129 (1994) 407–417.
[3] Daniel M.Gordon, A survey of fast exponentiation methods. Journal of algorithms. 27(1998) 129-146.
[4] J. C. Ha and S. J. Moon, A common-multiplicand method to the Montgomery algorithm for speeding up exponentiation. Inf. Process. Lett. 66 (1998) 105–107.
[5] M. Huang, K. Gaj and T. El-Ghazawi, New hardware architectures for Montgomery modular multiplication algorithm. IEEE Trans. Comput. 60 (2011) 923-936.
[6] S.-R. Kuang, J. P. Wang, K. C. Chang, and H. W. Hsu, Energy-efficient high-throughput Montgomery modular multipliers for RSA cryptosystems. IEEE Trans. Very Large Scale Integr. (VLSI) Syst. 21(2013) 1999–2009.
[7] P.L. Montgomery, Modular multiplication without trial division. Math. Comput. 44 (1985) 519–521.
[8] A. Miyamoto, N. Homma, T. Aoki and A. Satoh, Systematic design of RSA processors based on high-radix Montgomery multipliers. IEEE Trans. Very Large Scale Integr. (VLSI) Syst.19 (2011) 1136–1146.

[9] L.M. Mourelle and N. Nedjah, Fast reconfigurable hardware for the m-ary modular exponentiation. Proc. Euromicro Symp. on Digital System Design: Architectures, Methods and Tools . (2004) 516–523.
[10] N. Nedjah and L. M. Mourelle, Efficient hardware for modular exponentiation using the sliding-window method with variable-length partitioning. in Proc. 9th Int. Conf. Young Comput. Sci. (2008) 1980–1985.
[11] A.Rezai and P.Keshavarzi, A new CMM-NAF modular exponentiation algorithm by using a new modular multiplication algorithm. Trends in applied sciences research. 7(2012) 240-247.
[12] A.Rezai and P.Keshavarzi, Algorithm design and theorical analysis of a novel CMM modular exponentiation algorithm for large integers.RAIRO, Theor, Inf. appl. 49(2015) 255-268.
[13] A. Rezai and P. Keshavarzi, High-Throughput Modular Multiplication and Exponentiation Algorithm Using Multibit-Scan-Multibit-Shift technique. IEEE Trans. VLSI syst. 23(2015) 1710-1719.
[14] A. Rezai and P. Keshavarzi, Compact SD: A New Encoding Algorithm and Its Application in Multiplication. Int. j. comput. Math.94 (2017) 554-569.
[15] A.Rezai and P.Keshavarzi, high-performance scalable architecture for modular multiplication using a new digit-serial computation, Microelectronics journal. 55 (2016) 169-178.
[16] A.Rezai and P.Keshavarzi, High-perormance modular exponentiation algorithm by using a new modified modular multiplication algorithm and common-multiplicand multiplication method, 2011 World Congress on Internet Security (WorldCIS-2011), London. (2011) 192-197.
[17] G. D. Sutter, J. P. Deschamps and J. L. Imana, Modular multiplication and exponentiation architecture for fast RSA cryptosystem based on digit serial computation, IEEE Trans. Ind. Electron. 58 (2011) 3101-3109.
[18] [18] S.Vollala, K.Geetha and N.Ramasubramanian, Efficient modular exponential algorithms compatible with hardware implementation of public-key cryptography. Security Comm. Networks. 9 (2016) 3105–3115
[19] C. D. Walter, Systolic modular multiplication, IEEE Trans. Comput.42 (1993) 376-378.
[20] C. L. Wu, D. C. Lou and T. J. Chang, An Efficient Montgomery Exponentiation Algorithm for Public-key Cryptosystem. In Proc. of IEEE. Int. Conf. Intell. Security Inform., Taipei,Taiwan (2008) 284–285
[21] C. L. Wu, D. C. Lou and T. J. Chang, Fast modular multiplication based on complement representation and canonical recoding. Int. J. comput. Math. 87 (2010) 2871–2879
[22] C.L. Wu, An efficient common-multiplicand-multiplication method to the Montgomery algorithm for speeding up exponentiation. Inform. Sci. 179 (2009) 410-421.
[23] T. Wu, S. Li, and L. Liu, Fast, compact and symmetric modular exponentiation architecture by common-multiplicand Montgomery modular multiplications. Integr., VLSI J. 36(2013) 323–332.
[24] J. Xie, J. J. He and P. K. Meher, Low latency systolic Montgomery multiplier for finite field GF(2m) based on pentanomials, IEEE Trans. Very Large Scale Integr. (VLSI) Syst. 21 (2013) 385-389.