e-ISSN: 2348-6848
p-ISSN: 2348-795X
Volume 05 Issue 01
January 2018

Implementation of Low-Cost High-Performance Montgomery Modular Multiplication

[1]Medepalli Narasimha Rao, [2]Keerti kumar korlapati

[1]Asistant professor, [2]Associate professor,
[1,2]Dept. of ECE,
[1,]Kodada institute of Technology and Science for Wombn, Kodada, Telangana
[2.] Vaageswari College of Engineering, karimnagar, Telangana.

Abstract: In this we are introducing a simple and efficient Montgomery multiplication algorithm such that the low-cost and high-performance Montgomery modular multiplier can be implemented accordingly. The proposed multiplier receives and outputs the data with binary representation and uses only one-level carry-save adder (CSA) to avoid the carry propagation at each addition operation. To overcome the weakness of the one level CSA(more clock cycles), a configurable CSA (CCSA), which could be one full-adder or two serial half-adders, is proposed to reduce the extra clock cycles for operand pre computation and format conversion by half. In addition, a mechanism that can detect and skip the unnecessary carry-save addition operations in the one-level CCSA architecture while maintaining the short critical path delay is developed. As a result, the extra clock cycles for operand pre computation and format conversion can be hidden and high throughput can be obtained. Experimental results show that the proposed Montgomery modular multiplier can achieve higher performance and significant area–time product improvement when compared with previous designs.

Key words: Carry-save addition, Montgomery modular multiplier, public-key cryptosystem.

I.INTRODUCTION

In many public-key cryptosystems [1]–[3], modular multiplication (MM) with large integers is the most critical and time-consuming operation. Therefore, numerous algorithms and hardware implementation have been presented to carry out the MM more quickly, and Montgomery's algorithm is one of the most well-known MM algorithms. Montgomery's algorithm [4] determines the quotient only depending on the least significant digit of operands and replaces the complicated division in conventional MM with a series of shifting modular additions As a result; it can be easily implemented into VLSI circuits to speed up the encryption/decryption process. However, the three-operand addition in the iteration loop of Montgomery's algorithm as shown in step 4 of Fig. 1 requires long carry propagation for large operands in binary representation. To solve this problem, several approaches based on carry-save addition were proposed to achieve a significant speedup of Montgomery MM. Based on the representation of input and output operands, these approaches can be roughly divided into semi-carry-save (SCS) strategy and full carry-save (FCS) strategy.

```
Algorithm MM:
Radix-2 Montgomery modular multiplication
Inputs : $A, B, N$ (modulus)
Output : $S[k]$
1.  $S[0] = 0$;
2.  for $i = 0$ to $k - 1$ {
3.      $q_i = ( S[i]_0 + A_i \times B_0 ) \bmod 2$;
4.      $S[i+1] = ( S[i] + A_i \times B + q_i \times N ) / 2$;
5.  }
6.  if ( $S[k] \geq N$ ) $S[k] = S[k] - N$;
7.  return $S[k]$;
```

Fig. 1. MM algorithm.

In the SCS strategy [5]–[8], the input and output operands (i.e., A, B, N, and S) of the Montgomery MM are represented in binary, but intermediate results of shifting modular additions are kept in the carry-save format to avoid the carry propagation. However, the format conversion from the carry-save format of the final modular product into its binary representation is needed at the end of each MM. This conversion can be accomplished by an extra carry propagation adder (CPA) [5] or reusing the carry-save adder (CSA) architecture [8] iteratively. Contrary to the SCS strategy, the FCS strategy [9], [10] maintains the input and output operands A, B, and S in the carry-save format, denoted as (AS, AC), (BS, BC), and (SS, SC), respectively, to avoid the format conversion, leading to fewer clock cycles for completing a MM. Nevertheless, this strategy implies that the number of operands will increase and that more CSAs and registers for dealing with these operands are required. Therefore, the FCS-based Montgomery modular multipliers possibly have higher hardware complexity and longer critical path than the SCS-based multipliers.

Accordingly, this paper aims at enhancing the performance of CSA-based Montgomery multiplier while maintaining low hardware complexity. Instead of the FCS-based multiplier with two-level CSA architecture in [10], a new SCS-based Montgomery MM algorithm and its corresponding hardware architecture with only one-level CSA are proposed in this paper. The proposed algorithm and hardware architecture have the following several advantages and novel contributions over previous designs. First, the one-level CSA is utilized to perform not only the addition operations in the iteration loop of Montgomery's algorithm but also B + N and the format conversion, leading to a very short critical path and lower hardware cost. However, a lot of extra clock cycles are required to carry out B + N and the format conversion via the one-level CSA architecture. Therefore, the benefit of short critical path will be lessened. To overcome the weakness, we then modify the one-level CSA architecture to be able to perform one three-input carry-save addition or two serial two-input carry-save additions, so that the extra clock cycles for B + N and the format conversion can be reduced by half. Finally, the condition and detection circuit, which are different with that of FCS-MMM42 multiplier in [10], are developed to precompute quotients and skip the unnecessary carry-save addition operations in the one-level configurable CSA (CCSA) architecture while keeping a short critical path delay. Therefore, the required clock cycles for completing one MM operation can be significantly reduced. As a result, the proposed Montgomery multiplier can obtain higher throughput and much smaller area-time product (ATP) than previous Montgomery multipliers.

II.LITERATURE SURVEY

A. Montgomery Multiplication

Fig. 1 shows the radix-2 version of the Montgomery MM algorithm (denoted as MM algorithm). As mentioned earlier, the Montgomery modular product S of A and B can be obtained as $S = A \times B \times R^{-1}$ (mod N), where R^{-1} is the inverse of R modulo N. That is, $R \times R^{-1} = 1$ (mod N). Note that, the notation X_i in Fig. 1 shows the i th bit of X in binary representation. In addition, the notation $X_{i:\ j}$ indicates a segment of X from the i th bit to j th bit. Since the convergence range of S in MM algorithm is $0 \leq S < 2N$, an additional operation $S = S - N$ is required to remove the oversize residue if $S \geq N$. To eliminate the final comparison and subtraction in step 6 of Fig. 1, Walter [22] changed the number of iterations and the value of R to k + 2 and 2^{k+2} mod N, respectively. Nevertheless, the long carry propagation for the very large operand addition still restricts the performance of MM algorithm.

In this section, we propose a new SCS-based Montgomery MM algorithm to reduce the critical path delay of Montgomery multiplier. In addition, the drawback of more clock cycles for completing one multiplication is also improved while maintaining the advantages of short critical path delay and low hardware complexity.

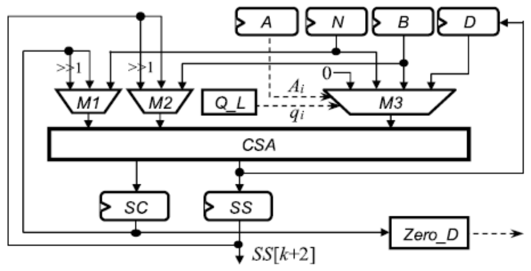
B. Critical Path Delay Reduction

The critical path delay of SCS-based multiplier can be reduced by combining the advantages of FCS-MM-2 and SCS-MM-2. That is, we can pre compute D = B + N and reuse the one-level CSA architecture to perform B+N and the format conversion. Fig. 2(a) and (b) shows the modified SCS-based Montgomery multiplication (MSCS-MM) algorithm and one possible hardware architecture, respectively. The Zero_D circuit in Fig. 2(b) is used to detect whether SC is equal to zero, which can be accomplished using one NOR operation. The Q_L circuit decides the qi value according to step 7 of Fig. 2(a). The carry propagation addition operations of B + N and the format conversion are performed by the one-level CSA architecture of the MSCS-MM multiplier through repeatedly executing the carry-save addition (SS, SC) = SS + SC + 0 until SC = 0. In addition, we also pre compute Ai and qi in iteration i−1 so that they can be used to immediately select the desired input operand from 0, N, B, and D through the multiplexer M3 in iteration i . Therefore, the critical path delay of the MSCS-MM multiplier can be reduced into $T_{MUX4} + T_{FA}$. However, in addition to performing the three-input carry-save additions [i.e., step 12 of Fig. 2(a)] k + 2 times, many extra clock cycles are required to perform B + N and the format conversion via the one-level CSA architecture because they must be performed once in every MM.

Algorithm Modified SCS-MM:
Modified SCS-based Montgomery multiplication

```
Inputs  : $A, B, N$ (modulus)
Output : $SS[k+2]$
1.  $(SS, SC) = (B + N + 0)$;
2.  while ($SC$ != 0)
3.      $(SS, SC) = (SS + SC + 0)$;
4.  $D = SS$;
5.  $SS[0] = 0$;  $SC[0] = 0$;
6.  for $i = 0$ to $k + 1$ {
7.      $q_i = (SS[i]_0 + SC[i]_0 + A_i \times B_0) \bmod 2$;
8.      if ($A_i = 0$ and $q_i = 0$)   $x = 0$;
9.      if ($A_i = 0$ and $q_i = 1$)   $x = N$;
10.     if ($A_i = 1$ and $q_i = 0$)   $x = B$;
11.     if ($A_i = 1$ and $q_i = 1$)   $x = D$;
12.     $(SS[i+1], SC[i+1]) = (SS[i] + SC[i] + x) / 2$;
13. }
14. while ($SC[k+2]$ != 0)
15.     $(SS[k+2], SC[k+2]) = (SS[k+2] + SC[k+2] + 0)$;
16. return $SS[k+2]$;
```

(a)



(b)

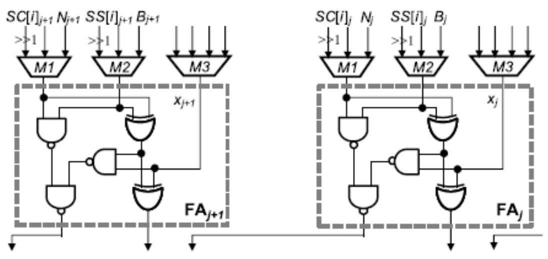
Fig. 2. (a) Modified SCS-based Montgomery multiplication algorithm. (b) MSCS-MM multiplier.

Furthermore, the extra clock cycles for performing B+N and the format conversion through repeatedly executing the carry-save addition (SS, SC) = SS+SC+0 are dependent on the longest carry propagation chain in SS + SC. If SS = 111…1112 and SC = 000…0012, the one-level CSA architecture needs k clock cycles to complete SS + SC.

That is, ~3k clock cycles in the worst case are required for completing one MM. Thus, it is critical to reduce the required clock cycles of the MSCS-MM multiplier.

B. Clock Cycle Number Reduction

To decrease the clock cycle number, a CCSA architecture which can perform one three-input carry-save addition or two serial two-input carry-save additions is proposed to substitute for the one-level CSA architecture in Fig. 2(b). Fig. 3(a) shows two cells of the one-level CSA architecture in Fig. 2(b), each cell is one conventional FA which can perform the three-input carry-save addition. Fig. 3(b) shows two cells of the proposed configurable FA (CFA) circuit. If α = 1, CFA is one FA and can perform one three-input carry-save addition (denoted as 1F_CSA). Otherwise, it is two half-adders (HAs) and can perform two serial two-input carry-save additions (denoted as 2H_CSA), as shown in Fig. 3(c). In this case, G1 of CFA$_j$ and G2 of CFA$_{j+1}$ in Fig. 3(b) will act as HA1 $_j$ in Fig. 3(c), and G3, G4, and G5 of CFA$_j$ in Fig. 3(b) will behave as HA2$_j$ in Fig. 3(c).Moreover, we modify the 4-to-1 multiplexer M3 in Fig. 2(b) into a simplified multiplier SM3 as shown in Fig. 3(d) because one of its inputs is zero, where ~ denotes the INVERT operation. Note that M3 has been replaced by SM3 in the proposed one-level CCSA architecture shown in Fig. 3(b). According to the delay ratio, T_{SM3} (i.e., 0.68 × T_{FA}) is approximate to T_{MUX3} (i.e., 0.63 × T_{FA}) and T_{MUXI2} (i.e., 0.23 × T_{FA}) is smaller than T_{XOR2} (i.e., 0.34×T_{FA}).


$SC[i]_{j+1}$ N_{j+1} $SS[i]_{j+1}$ B_{j+1}
$SC[i]_j$ N_j $SS[i]_j$ B_j
>>1
M1
M2
M3
x_{j+1}
x_j
FA$_{j+1}$
FA$_j$

(a)

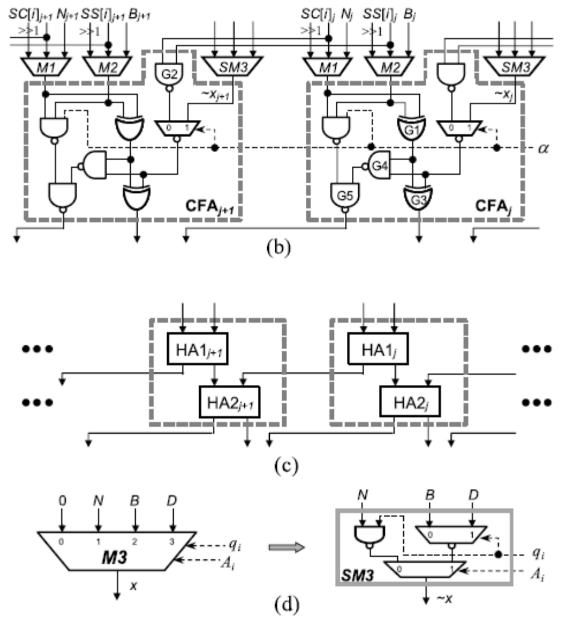


Fig. 3. (a) Conventional FA circuit. (b) Proposed CFA circuit. (c) Two serial HAs. (d) Simplified multiplexer SM3.

Therefore, the critical path delay of the proposed one-level CCSA architecture in Fig. 3(b) is approximate to that of the one-level CSA architecture in Fig. 3(a). As a result, steps 3 and 15 of Fig. 2(a) can be replaced with (SS, SC) = 2H_CSA (SS, SC) and (SS [k + 2], SC [k + 2]) = 2H_CSA (SS[k +2], SC[k +2]) to reduce the required clock cycles by approximately a factor of two while maintaining a short critical path delay. In addition, we also skip the unnecessary operations in the for loop (steps 6 to 13) of Fig. 2(a) to further decrease the clock cycles for completing one Montgomery MM. The crucial computation in the for loop of Fig. 2(a) is performing the following three-to-two carry-save addition:

III.PROPOSED SYSTEM

On the bases of critical path delay reduction, clock cycle number reduction, and quotient pre computation mentioned above, a new SCS-based Montgomery MM algorithm (i.e., SCS-MM-New algorithm shown in Fig. 4) using one-level CCSA architecture is proposed to significantly reduce the required clock cycles for completing one MM. As shown in SCS-MM-New algorithm, steps 1–5 for producing ˆB and ˆD are first performed. Note that because qi+1 and qi+2 must be generated in the i th iteration, the iterative index i of Montgomery MM will start from −1 instead of 0 and the corresponding initial values of ˆ q and ˆA must be set to 0. Furthermore, the original for loop is replaced with the while loop in SCS-MM-New algorithm to skip some unnecessary iterations when skipi+1 = 1. In addition, the ending number of iterations in SCS-MM-New algorithm is changed to k + 4 instead of k + 1 in Fig. 2(a). This is because B is replaced with ˆB and thus three extra iterations for computing division by two are necessary to ensure the correctness of Montgomery MM. In the while loop, steps 8–12 will be performed in the proposed one-level CCSA architecture with one 4-to-1 multiplexer. The computations of qi+1, qi+2, and skipi+1 in step 13 and the selections of ˆA , ˆ q, and i in steps 14–20 can be carried out in parallel with steps 8–12. Note that the right-shift operations of steps 12 and 15 will be delayed to next clock cycle to reduce the critical path delay of corresponding hardware architecture. The hardware architecture of SCS-MM-New algorithm, denoted as SCS-MM-New multiplier, are shown in Fig. 5, which consists of one one-level CCSA architecture, two 4-to-1 multiplexers (i.e., M1 and M2), one simplified multiplier

SM3, one skip detector Skip_D, one zero detector Zero_D, and six registers. Skip_D is developed to generate skipi+1, ˆ q, and ˆA in the i th iteration. Both M4 and M5 in Fig. 5 are 3-bit 2-to-1 multiplexers and they are much smaller than k-bit multiplexers M1, M2, and SM3. In addition, the area of Skip_D is negligible when compared with that of the k-bit one-level CCSA architecture. The select signals of multiplexers M1 and M2 in Fig. 11 are generated by the control part, which are not depicted for the sake of simplicity.

Algorithm SCS-MM-New:
Proposed SCS-based Montgomery multiplication

```
Inputs  : $A$, $B$, $\hat{N}$ (new modulus)
Output : $SS[k+5]$
1.  $\hat{B} = B << 3$;  $\hat{q} = 0$;  $\hat{A} = 0$;  $skip_{i+1} = 0$;
2.  $(SS, SC) = $ 1F_CSA( $\hat{B}$, $\hat{N}$, 0);
3.  while ($SC$ != 0)
4.      $(SS, SC) = $ 2H_CSA($SS$, $SC$);
5.  $\hat{D} = SS$;
6.  $i = -1$;  $SS[-1] = 0$;  $SC[-1] = 0$;
7.  while ( $i \le k + 4$ ) {
8.      if ( $\hat{A} = 0$ and $\hat{q} = 0$)   $x = 0$;
9.      if ( $\hat{A} = 0$ and $\hat{q} = 1$)   $x = \hat{N}$ ;
10.     if ( $\hat{A} = 1$ and $\hat{q} = 0$)   $x = \hat{B}$ ;
11.     if ( $\hat{A} = 1$ and $\hat{q} = 1$)   $x = \hat{D}$ ;
12.     $(SS[i+1], SC[i+1]) = $ 1F_CSA($SS[i]$, $SC[i]$, $x$)>>1;
13.       compute $q_{i+1}$, $q_{i+2}$, and $skip_{i+1}$ by (5), (7) and (8);
14.     if ($skip_{i+1} = 1$){
15.         $SS[i+2] = SS[i+1]$>>1;  $SC[i+2] = SC[i+1]$>>1;
16.         $\hat{q} = q_{i+2}$;  $\hat{A} = A_{i+2}$;  $i = i + 2$;
17.     }
18.     else{
19.         $\hat{q} = q_{i+1}$;  $\hat{A} = A_{i+1}$;  $i = i + 1$;
20.     }
21. }
22. $\hat{q} = 0$;  $\hat{A} = 0$;
23. while ($SC[k+5]$ != 0)
24.     $(SS[k+5], SC[k+5]) = $ 2H_CSA($SS[k+5]$, $SC[k+5]$);
25. return $SS[k+5]$;
```

Fig. 4. SCS-MM-New algorithm.

At the beginning of Montgomery multiplication, the FFs stored skipi+1, ˆ q, ˆA are first reset to 0 as shown in step 1 of SCS-MM-New algorithm so that ˆD = ˆB + ˆN can be computed via the one-level CCSA architecture.

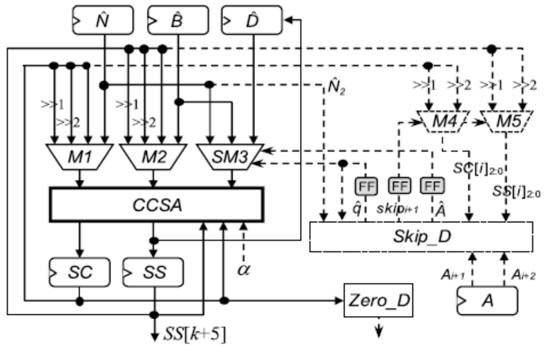


Fig.5. SCS-MM-New multiplier.

When performing the while loop, the skip detector Skip_D shown in Fig. 6 is used to produce skipi+1, ˆ q, and ˆA. The Skip_D is composed of four XOR gates, three AND neither gates, one NOR gate, and two 2-to-1 multiplexers. It first generates the q_{i+1}, q_{i+2}, and $skip_{i+1}$ signal in the i th iteration according to (5), (7), and (8), respectively, and then selects the correct ˆ q and ˆA according to $skip_{i+1}$.

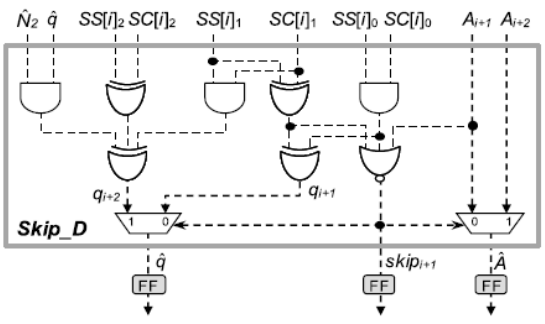


Fig. 6. Skip detector Skip_D.

At the end of the i th iteration, ˆ q, ˆA, and skipi+1 must be stored to FFs. In the next clock cycle of the i th iteration, SM3 outputs a proper x according to ˆ q and ˆA generated in the i th iteration as shown in steps 8–11, and M1 and M2 output the correct SC and SS according to skipi+1 generated in the i th iteration. If $skip_{i+1}$ = 0, SC >>1 and SS >>1 are selected. Otherwise, SC >>2 and SS >>2 are selected. That is, the right-shift 1-bit operations in steps 12 and 15 of SCS-MM-

New algorithm are performed together in the next clock cycle of iteration i. In addition, M4 and M5 also select and output the correct SC[i $]_{2:0}$ and SS[i $]_{2:0}$ according to skipi+1 generated in the i th iteration. Note that SC[i $]_{2:0}$ and SS[i $]_{2:0}$ can also be obtained from M1 and M2 but a longer delay is required because they are 4-to-1 multiplexers. After the while loop in steps 7–21 is completed, ˆ q and ˆA stored in FFs are reset to 0. Then, the format conversion in steps 23 and 24 can be performed by the SCS-MM-New multiplier similar to the computation of ˆD = ˆB + ˆN in steps 3 and 4. Finally, SS [k +5] in binary format is outputted when SC[k + 5] is equal to 0.

IV.CONCLUSION

FCS-based multipliers maintain the input and output operands of the Montgomery MM in the carry-save format to escape from the format conversion, leading to fewer clock cycles but larger area than SCS-based multiplier. To enhance the performance of Montgomery MM while maintaining the low hardware complexity, this paper has modified the SCS-based Montgomery multiplication algorithm and proposed a low-cost and high-performance Montgomery modular multiplier. The proposed multiplier used one-level CCSA architecture and skipped the unnecessary carry-save addition operations to largely reduce the critical path delay and required clock cycles for completing one MM operation. Experimental results showed that the proposed approaches are indeed capable of enhancing the performance of radix-2 CSA-based Montgomery multiplier while maintaining low hardware complexity.

V.REFERENCES

[1] R. L. Rivest, A. Shamir, and L. Adleman, “A method for obtaining digital signatures and public-key cryptosystems,” Commun. ACM, vol. 21, no. 2, pp. 120–126, Feb. 1978.

[2] V. S. Miller, “Use of elliptic curves in cryptography,” in Advances in Cryptology. Berlin, Germany: Springer-Verlag, 1986, pp. 417–426.

[3] N. Koblitz, “Elliptic curve cryptosystems,” Math. Comput., vol. 48, no. 177, pp. 203–209, 1987.

[4] P. L. Montgomery, “Modular multiplication without trial division,” Math. Comput., vol. 44, no. 170, pp. 519–521, Apr. 1985.

[5] Y. S. Kim, W. S. Kang, and J. R. Choi, “Asynchronous implementation of 1024-bit modular processor for RSA cryptosystem,” in Proc. 2nd IEEE Asia-Pacific Conf. ASIC, Aug. 2000, pp. 187–190.

[6] V. Bunimov, M. Schimmler, and B. Tolg, “A complexity-effective version of Montgomery’s algorihm,” in Proc. Workshop Complex. Effective Designs, May 2002.

[7] H. Zhengbing, R. M. Al Shboul, and V. P. Shirochin, “An efficient architecture of 1024-bits cryptoprocessor for RSA cryptosystem based on modified Montgomery’s algorithm,” in Proc. 4th IEEE Int. Workshop Intell. Data Acquisition Adv. Comput. Syst., Sep. 2007, pp. 643–646.

[8] Y.-Y. Zhang, Z. Li, L. Yang, and S.-W. Zhang, “An efficient CSA architecture for Montgomery modular multiplication,”

Microprocessors Microsyst., vol. 31, no. 7, pp. 456–459, Nov. 2007.
[9] C. McIvor, M. McLoone, and J. V. McCanny, "Modified Montgomery modular multiplication and RSA exponentiation techniques," IEE Proc.-Comput. Digit. Techn., vol. 151, no. 6, pp. 402–408, Nov. 2004.
[10] 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., vol. 21, no. 11, pp. 1999–2009, Nov. 2013.