American Journal of Electronics & Communication

Sudarsan Biswas et. al., American Journal of Electronics & Communication, Vol. III (3), 16-19

Comparative Analysis of Multi-digit Modular Multiplication Algorithms for public key Crypto system in Big Data Security.

Sudarsan Biswas[1], Neepa Biswas[2]
[1]RCC Institute of Information Technology, Kolkata
[2]University Engineering & Management, Kolkata
[1]E-mail: biswas.sudarsan@gmail.com
[2]E-mail: biswas.neepa@gmail.com

ABSTRACT

Big data security is a significant concern today due to rapidly growing data volume, velocity, and varieties of data in nature. Data security is a major concern because during transmission through insecure channels. From its recognition, the Montgomery multi-digit multiplication algorithms are still well-accepted approaches in the numerical calculation, chaos arithmetic, and high-performance implementation of public key cryptosystems such as RSA, Diffie-Hellman, and the Elliptic curve cryptosystems. We have presented the comparative study of multi-digit integer multiplication algorithms, and their performance is evaluated in terms of CPU execution time. Montgomery modular multiplication is a good practice, among others.

Keywords Montgomery, Karatsuba, Classical Multiplication, Performance Analysis.

1. INTRODUCTION

Modular arithmetic is one of the cornerstones of public-key cryptography. Therefore, it is desirable to have an efficient implementation of modular arithmetic approach. While modular additions and subtractions are rather trivial cases, efficient modular multiplication remains an elusive target for optimization. The two most widely used algorithms for modular multiplication are Montgomery's method [3]. Modular reduction algorithms can be categories in two classes, the left-to-right and right-to-left algorithms. Variations of these two schemes were reported in [4, 5], but the essential idea is the same.

R. Garg and R. Vig [6] made an interesting observation on efficient Montgomery Multiplication Algorithm and RSA Cryptographic Processor that employs multi-bit shifting and carry save addition to perform long integer arithmetic. C. K. Koc and T. Acar [7] present a new method for fast implementation of exponentiation operation in GF(2^k). The proposed method is based on the Montgomery multiplication algorithm [3] which eliminates the mod *n* reduction steps. The result tends to reduce the size of the timing characteristics. Also Montgomery multiplication algorithms speed up the computation operation for modular exponentiations and modular multiplications [8]. The complexity of classical method for multiplying integers is O(N^2) and the same is applicable for the modulo operation. Besides for security reasons these algorithms are applied to very large integers (1024 bits or higher). Other methods exist to multiply (large) integers. Karatsuba and Ofman [9] proposed a recursive algorithm, which is about O($N^{1.58}$) C. K. Koc and T. Acar [10] proposed the Montgomery multiplication in GF(2^k), they have used the bit level and word level algorithms for computing the product. A through performance analysis and the algorithm comparison is done with the other standard multiplication algorithm in GF(2^k). J. P. David, K. Kalach, N. Tittley [11] made the comparison of different Modular Multiplication and Exponentiation Algorithms.

S. Yazaki and K. Abe [12] present two design choices of the Karatsuba hardware: IKM (Iterative Karatsuba Multiplier) and RKM (Recursive Karatsuba Multiplier). Z. Dyka Z, P. Langendoerfer [13] proposed a hardware-based Karatsuba algorithm implementations over a Galois field (GF), that can used in elliptic curve cryptography and many other applications. C. McIvor, M. McLoone and J. V. McCanny [14] proposed a improved Montgomery multiplication along with associated RSA modular exponentiation algorithms and designed a circuit architectures. They modified multipliers to use carry save adders (CSAs) to perform large word length additions. Proposed modified Montgomery multiplication algorithms are suitable for RSA exponentiation, that can overcome carry propagation problem. The proposal uses a five-to-two CSA and a four-to-two CSA respectively. Each one can calculate a Montgomery multiplication in only k + 1 and k + 2 clock cycles, respectively. Here k denotes the operand bit length. For the empirical analysis of the Montgomery algorithm in an RSA cryptosystem, Keon-Jik Lee et. al. [15] made an observation on digit-serial-in-serial-out systolic multiplier. Due processing speed, their proposed multiplier can also accommodate bit-level pipelining. It can achieve sample speeds compared to bit-parallel multipliers with a lower area. Chin-Bou Liu et. al. [16] made an interesting observation based on karatsuba multiplication (KM), using tensor production formulation to express KM algorithm in both iterative and recursive form. Shand and Vuillemin applied KM algorithm to the optimal radix choice of modular product applicable for RSA cryptography [17]. Hollerbach presented a generalization of the Karatsuba's algorithm for multiplying very large numbers [18]. D. Yuliang, M. Zhigang, Y. Yizheng, W. Tao [19] presents a hardware, implementation of 1024-bit RSA crypto processor based on Montgomery multiplication algorithm. The processor can encrypt 1024 bit message in less than 0.65

seconds, with which a $3mm^2$ die area. That is suitable for smart IC cards. X. Fang and L. Li [20] introduces classical Knuth multiplication, Karatsuba multiplication and their time complexity. Their experimental result shows more efficiency for implementation of large integer multiplication. K. Shiann-Rong et. al. designed a architecture that results Low-cost high-performance for Montgomery modular multiplication at VLSI domain [21]. D. Jinnan and L. Shuguo presented a low-cost and low-latency Montgomery modular multiplier based on NLP multiplication [22]. Z. Bo, C. Zeming and P. Massoud proposed an Iterative methodology for Montgomery Modular Multiplication Algorithm with low Area-Time product [23].

The paper is organized as the followings. In Section 2, we presented an overview of the different multiplication algorithms. Experimental results with input data sets are presented in Section 3. Performance analysis is reported in Section 4. Conclusions are in Section 5.

2. MULTIDIGITMULTI DIGIT ALGORITHM ANALYSIS

2.1 Montgomery Multiplication

Montgomery multiplication is the key methods of the modular exponentiation operation. It follows a special representation of numbers named as

```
$t_2 = a_1 \times b_1$
10  $u = (a_0 + a_1) \times (b_1 + b_0)$
11  $t_1 = u - t_0 - t_2$
12  return $t = t_2 \times \omega^2 + t1 \times \omega + t_0$
```

This method can be extended to multi-digit long integers by recursion.

2.3.2 Karatsuba - Ofman recursive Algorithm (KORA)

```
Function KOMA (a,b)
Input: a, b
Output: $t = a \times b$
0  if (sizeof (a)==32 and sizeof (b)==32)
       return $a \times b$
1  $t_0$=KORMA($a_0$,$b_0$)
2  $t_2$=KORMA($a_1$,$b_1$)
3  u=KORMA($a_1 + a_0$,$b_1 + b_0$)
4  $t_1 = u - t_0 - t_2$
5  return $t = t_2 \times \omega^2 + t_1 \times \omega + t_0$
```

The following simple example will help illustrate the algorithm

We intend to multiply two integers 5683 by 1148. For clarity, we will use decimal formalism, so ω=10, m=2. Initially x and y are split into two equal sized parts as follow

$X=a_1 \omega + a_0$ and $y=b_1 \omega + b_0$

$X=56\times10^2+83$ $y=11 \times 10^2 + 48$

$P_0 = a_0 \times b_0 = (83 \times 48) = 3984$

$P_2=a_1\times b_1=56\times 11= 616$

$P_1=(a_0 + a_1) (b_0+ b_1) - P_2 - P_0$

$= (83 + 56) (48 + 11) - 616 - 3984$

$=8201- 616 - 3948 =3601$

$P =p_2 \omega^2+p_1 \omega +p_0$

$=616 \times 10^4+ 3601\times 10^2+ 3984$

$=6524084$

3. EXPERIMENTAL RESULTS

We have implemented the Montgomery Karatsuba and Classical multiplication algorithms. We have run the implementation with set of integers of different problem size to obtained execution time. We have executed the multiplication programs 3000 thousand times and obtained the average timings for each data set for multiplier and multiplicand. The different problem sizes are reported in Table 1. The execution times are also plotted in Table 2.

To calculate the processor time we used clock () function which is defined into the header file from the invocation time of the called process and time measured in const CLOCKS_PER_SECs. The CPU execution time depends on many factors. The first one of them is the produced machine code, which depends on both the high level source code, and the compiler used. It also depends on the instruction set of the target processor as well as the number of available registers in the processor. The compiler has a key role in the speed of the executed code. An efficient use of the resources (including the available registers) can speed up the execution. In this paper, we have simulated three different multi-digit multiplication algorithms with respect to the CPU execution time.

4. COMPARATIVE ANALYSIS

In this section, we run the application of above multiplication algorithms whose input is the large integers of different problem size and the corresponding CPU execution time are given into the table 2.

From Figure-1 it is observed that, the execution time interval is increasing linearly for both Classical Algorithm and Karatsuba upon increasing the number of digits. The performance of the Karatsuba algorithm is going to be worse compared to Classical algorithm. Moreover, compared to other two algorithms the execution time taken for Montgomery algorithm is almost steady instead of increasing the number of digits. It is observed that for multiplication of 6 digits, 12 digits and 18 digits numbers require same execution time. So it may be concluded that, the execution time will be same for the multiplication of those numbers, which is a multiple of 6 digits. Furthermore, up to 4 digits multiplication the execution time is almost same and shown in table 2. However, for 5 digits and 6 digits multiplication, the execution time is decreasing a little. The same behavior is observed for the higher order digits multiplication. So Montgomery algorithm shows a steady behavior other than the Karatsuba and Classical algorithm.

Figure 1: Comparison result of different multiplication algorithm

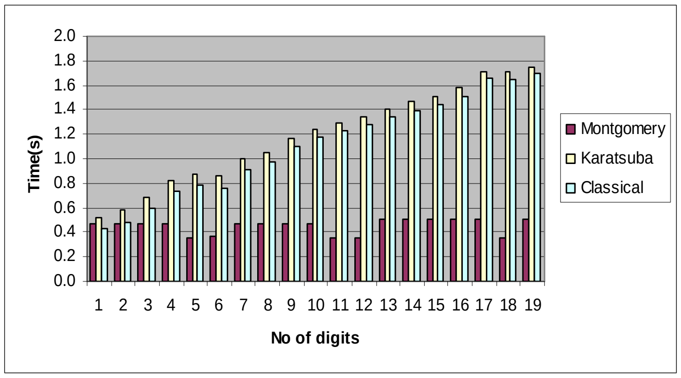

Figure 2 shows that, Karatsuba multiplication is the slowest of doing multiplication when the problem size is less than twenty digits, even slower than Classical multiplication, followed by Montgomery multiplication

Figure 2: Comparison results of different Multiplication Algorithms.

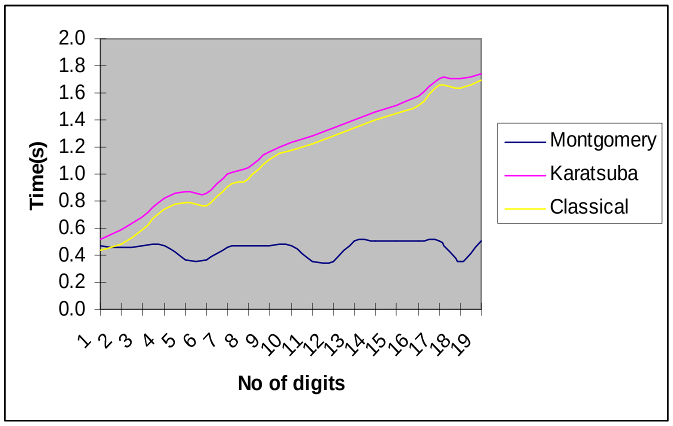

Table 1: Different input Data sets of Multiplication Algorithms

MULTIPLICAND	MULTIPLIER
8	6
97	58
759	643
8937	7839
97835	82317
983476	783493
9683773	8348734
89786547	67343576
963793649	813745687
9878939849	9748769817
73849657921	63456175327
984126781019	483456187104
4356739143142	3212496278273
83493192142136	45314508132932
736142139212367	248132181231079
6932101437124869	5231428739023671
74312901325583109	32143268730947712
983423821623983407	423456789012345678
6334567842123416718	4334267219123156717

Table 2: Computation time of different Multiplication algorithms (ms)

No of digits	Montgomery	Karatsuba	Classical
1	0.4686667	0.5206667	0.4323333
2	0.4634896	0.5833333	0.4793333
3	0.4688212	0.6820000	0.5886667
4	0.4688195	0.8176667	0.7393333
5	0.3594549	0.8750000	0.7863333
6	0.3644532	0.8596667	0.7656667
7	0.4637882	0.9946667	0.9066667
8	0.4687778	1.0520000	0.9686667
9	0.4688229	1.1666667	1.1043333
10	0.4688229	1.2396667	1.1770000
11	0.3543330	1.2863333	1.2293333
12	0.3544514	1.3440000	1.2813333
13	0.5001182	1.4010000	1.3386667
14	0.5055000	1.4636667	1.3960000
15	0.5105018	1.5103333	1.4476667
16	0.5051702	1.5783333	1.5103333
17	0.5051684	1.7033333	1.6563333
18	0.3488351	1.7033333	1.6406667
19	0.5107830	1.7450000	1.6926667

Table 3: Montgomery multiplication example

i	x_i	x_iy_0	u_i	x_iy	u_im	A
0	3	3×8=24	4	3444	290556	29400
1	8	8×8=64	4	9184	290556	32914
2	6	6×8=48	2	6888	145278	18508
3	5	5×8=40	8	5740	58112	60536
4	0	0×8=0	6	0	435834	49637

5. CONCLUSIONS

We present the comparison of multi-digit multiplication algorithms like Montgomery, Karatsuba and Classical algorithms. Among these three, Montgomery multiplication algorithm performs large integer modular multiplication with greater speed and stable timing characteristics up to twenty digits. This may be applied for the higher digits multiplication also. Karatsuba and Classical algorithm takes almost same time for multiplication up to twenty digits but classical algorithm faster than Karatsuba. With these techniques, n-bit multiplication which otherwise has a complexity of O (n^2) can be performed with a complexity of O $(n^{\log 3})$. In future work the enhancement of Montgomery modular multiplication for Big data security.
Cryptography is an essential task of IoT design. The Montgomery modular multiplication approach can be applied for cryptographic approach in IOT devices.

6. REFERENCES

[1] W. Stallings, "Cryptography and Network Security, Principles and practices" Edition 3d, Pearson Education.
[2] A.Menezes, P.Van Oorschot, and S.Vanstone, "Handbook of applied Cryptography", CRC Press, 1996.
[3] P. L. Montgomery, "Modular Multiplication without Trial Division" Mathematics of Computation, vol. 44, no. 170, pp. 519—521, April 1985.
[4] C.K.Koç, T.Açar and B.S.Kaliski Jr. "Analyzing and Comparing Montgomery Multiplication Algorithms" IEEE Micro, June 1996.
[5] J. J. Quisquater, "Encoding system according to the socalled RSA-method, by means of a microcontroller andarrangement implementing this system" U.S. Patent #5,166,978, November 1992.
[6] R. Garg, "An Efficient Montgomery Multiplication Algorithm and RSA Cryptographic Processor" International Conference on Computational Intelligence and Multimedia Applications 2007.
[7] C. K. Koc and T. Acar ,"Fast software exponentiation in GF(2^k)" In Proceeding of Third Annual Workshop on Selected Area in Cryptography, pages 95-106,Queen's University ,Kingston, Ontario, Canada 15-16 1996.
[8] Thierry Moreau, "Software Acceleration for Public Key Cryptography" Connotech Experts-conceals Inc, May 1977.
[9] A. A. Karatsuba and Y.Ofman, "Multiplication of Multi-digit Numbers on Automata" Soviet Physics Doklady, vol 7, pp.595-596 1963
[10] C. K. Koc and T. Acar, "Montgomery multiplication in GF(2^k)" In Proceeding of Third Annual Workshop on Selected Area in Cryptography, pages 95-106, Queen's University, Kingston, Ontario, Canada 15-16 1996.
[11] J. P. David, K. Kalach, N. Tittley "Hardware complexity of Modular Multiplication and Exponentiation" IEEE transaction on Computers vol. 56, No. 10, October 2007.
[12] S. Yazaki and K. Abe, "VLSI Design of Karatsuba Integer Multipliers and Its Evaluation" Electronics and Communications in Japan, Vol. 92, No. 4, 2009.
[13] Z. Dyka , P. Langendoerfer, "Area efficient hardware implementation of elliptic curve cryptography by iteratively applying Karatsuba's method" Proc of the Design, Automation and Test in Europe Conference and Exhibition, Vol. 3, p 70–75, 2005.
[14]. C. McIvor, M. McLoone and J. V. McCanny ,"Modified Montgomery modular multiplication and RSA exponentiation techniques" IEE Proc.-Comput. Digit. Tech., Vol. 151, No. 6, November 2004
[15] K. Lee, K. W. Kim, K. Y.Yoo , "Digit-serial-in-serial-out systolic multiplier for Montgomery algorithm" Information Processing Letters 82 (2002) 65–71.
[16] C. Liu, C. Huang, C. L. Lei, "Design and Implementation of Long-Digit Karatsuba's Multiplication Algorithm Using Tensor Product Formulation" The Ninth Workshop on Compiler Techniques for HighPerformance Computing.
[17] M. Shand and J. Vuillemin, "Fast implementations of RSA cryptography" In Proc. of the 11th IEEE Symposiumon Computer Arithmetic, pages 252–259,1993
[18] U. Hollerbach, "Fast multiplication & division of very large numbers" InSci. Math .Research Posting , 1996.
[19] D.Yuliang, M.Zhigang, Y. Yizheng, W. Tao "Implementation of RSA Crypto-processor Based on Montgomery algorithm"
[20] X. Fang, L. Li, "On Karatsuba Multiplication Algorithm" IEEE computer socity ©2007 IEEE DOI 10.1109/ISDPE.2007.11
[21] K. Shiann-Rong ,W. Kun-Yi and L. Ren-Yao, "Low-cost high-performance VLSI architecture for Montgomery modular multiplication " , IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol 24, no. 2, pages 434–443, 2015.
[22] D. Jinnan and L. Shuguo, " A low-latency and low-cost Montgomery modular multiplier based on NLP multiplication" IEEE Transactions on Circuits and Systems II: Express Briefs , vol 67, pages 1319–1323, 2019.
[23] Z., Bo, C. Zeming and P. Massoud, " An Iterative Montgomery Modular Multiplication Algorithm With Low Area-Time Product " IEEE Transactions on Computers, 2022.