An Improved Montgomery's Method Over Public-Key Cryptosystem

GVS Raj Kumar, Lagadapati Maanasa , K.Naveen Kumar, P.Chandra Sekhar , Mahesh Kasula

Dept of Information Technology, GITAM University, Visakhapatnam, India

***Abstract*- This paper deals with improving Montgomery's algorithm. We improve mongomery's algorithm such that modular multiplications can be executed two times faster. Each iteration in our algorithm requires only one addition, while that in Montgomery's requires two additions. We then propose a cellular array to implement modular exponentiation for the Rivest–Shamir–Adleman cryptosystem. It has approximately** $2n$ **cells, where** n **is the word length. The cell contains one full-adder and some controlling logic. The time to calculate a modular exponentiation is about** $2n^2$ **clock cycles.**

***Keywords*-Cellular array,data security,modular multiplication, Montgomery's algorithm, public-key cryptography, RSA.**

I. INTRODUCTION

One of the most widely used and popular public key cryptographic system is Rivest, Shamir and Adleman(RSA) encryption algorithm. RSA PublicKeyCryptosystem (PKC) is widely used in today's secure electronic communication. It was invented in 1978 by Rivest *et al.* [1].

Classical symmetric cryptographic algorithms provide a secure communication channel to each pair of users. In order to establish such a channel, the symmetric key algorithms employ a classical encryption scheme in which both the algorithm depends on the same secret key, *k.* This key is used for both encryption and decryption. After establishing a secure communication channel, the secrecy of a message, *m* can be guaranteed. Symmetric cryptography also includes methods to detect modification of messages and methods to verify the origin of a message. Thus, confidentiality and integrity can be accomplished using secret key techniques. A problem of concern is how to send the secret key between the transmitter and receiver. For a PKC, the decryption key is different from the encryption key, and the public key is either of the two keys. Deriving the decryption/encryption key from the public encryption/decryption key is difficult by definition. If the transmitter encrypts a plaintext by the receiver's public encryption key, only the receiver has the key to decrypt the ciphertext to its original plaintext. Moreover, the PKC's also provide a powerful solution to the implementation of digital signature [1], [2]. The security of a PKC is provided by the characteristics of its one-way function. Let $f: x \rightarrow y$ denote a one-way mapping, then the calculation of $f^{-1}(y)$ has to be hard, given the calculation of f(x). The famous RSA scheme [1] is based on the Euler and Fermat theorem [3]. Its security is related to the decomposition of N, which is the product of two distinct large prime numbers. It is known that large number decomposition is hard.

The core arithmetic of RSA is modular exponentiation which can be accomplished by a sequence of modular multiplication. Therefore, fast modular multiplication becomes the key to real-time encryption and decryption. However, since the numbers in a usable RSA PKC are very large (512 bytes or more), its implementation is challenging. Also, due to range comparison and adjustment, the implementation of modular multipliers is much harder than that of normal multipliers [4], [5]. To reduce the time complexity for comparison, a modular multiplication algorithm based on Montgomery's modular arithmetic [6] was proposed by Eldridge [7]. Montgomery's algorithm needs n iterations in each modular multiplication and two additions per iteration.In modified montgomery's algorithm we separate the multiplication and modular reduction steps in montgomery's algorithm such that only one addition is required in each iteration. The number of iterations in the modified algorithm is two times that of Montgomery's, hence the overall computation time is not reduced. However, the modified algorithm guarantees that the partial products in all modular multiplications fall in the correct range , hence, the post adjustment in the original algorithm is removed. The modified algorithm leads to both simpler architecture and better performance.So we further modified the algorithm by considering only the least significant half of the product in the modular reduction step. The number of iterations is the same as in Montgomery's algorithm, hence, the computation time is reduced in half. In this paper, we revise Montgomery's algorithm such that only one addition is required in each iteration, and the number of iterations is not increased. Therefore, the

speed of modular multiplication can be doubled. Even in software implementation, the proposed modular arithmetic is at least twice faster than other existing algorithms without using the Chinese Remainder Theorem (CRT). The proposed algorithm is implemented by a 2's-complement multiplier and a modular shifter-adder, both of which are designed as linear cellular arrays. For a word length of n, both arrays have about n cells, and each cell contains one Full-Adder (FA) and some controlling logic. The time to calculate a modular exponentiation is $2n^2$ clock cycles.

II. MONTGOMERY'S METHODS

The motivation for studying high speed and space-efficient algorithms for modular multiplication comes from their applications in public key cryptography. The RSA algorithm requires a large number of modular exponentiation, which binary methods can break into a series of modular multiplications. These computations are time taking, bringing down the speeds for key generation, encryption and decryption. One of the most interesting and useful advances in this realm has been what we call the Montgomery multiplication algorithm.

The Montgomery multiplication algorithm speeds up the modular multiplications and squaring required for exponentiation. It computes the Montgomery product.

Montgomery Multiplication Algorithm

A.Montgomery's Algorithm

Let $A= \{a_{n-1}, a_{n-2}, \ldots\ldots a_1, a_0\}$ and $B= \{b_{n-1}, b_{n-2}, \ldots\ldots b_1, b_0\}$ be two n-bytes integers, and N be an n-bytes odd integer, where $0 \leq B < N$.

FUNCTION MM (A, B, N):

Step 1: $S_0=0$

Step 2: for i=0 to n-1

Step 3: if ($S_i + a_i B$ is even)

$S_{i+1} = (S_i + a_i B)/2$

Step 4: else

$S_{i+1} = (S_i + a_i B + N)/2$

Step 5: return $\mathbf{S_n}$

A sequence $\mathbf{S_1, S_2 \ldots S_n}$ is generated by **MM ( ),** where $\mathbf{2^j S_j \equiv \left(\sum_{i=0}^{j-1} a_i 2^i\right) B.(mod\ N)}$ and $\mathbf{0 \leq S_j < N+B<2N}$ for j=1 to n. Let $\mathbf{R= 2^n}$, then

$$\mathbf{RS_n \equiv AB\ (mod\ N)} \qquad \text{--(1)}$$

Montgomery's algorithm needs n iterations in each modular multiplication and requires two additions per iteration. Also, the final result S_n is not in correct range, hence, post adjustment is required.

B. Improved Montgomery's Algorithm

If $\mathbf{a_i\ B}$ in procedure **MM( )** can be precomputed,its implementation will be much easier. It first calculates the product **X=A*B** which is represented as $\sum_{i=0}^{2n-1} \mathbf{x_i 2^i}$, and then employs Montgomery's algorithm to reduce the range.The modified algorithm is

FUNCTION CM(A,B,N)

Step 1: $\sum_{i=0}^{2n-1} \mathbf{x_i\, 2^i = A*B}$

Step 2: $S_0=0$

Step 3: for i=0 to 2n-1

Step 4: if ($S_i + x_i$ is even)

$S_{i+1} = (S_i + x_i)/2$

Step 5: else

$S_{i+1} = (S_i + x_i + N)/2$

Step 6: return $\mathbf{S_{2n}}$

A sequence $\mathbf{S_1, S_2, \ldots\ldots, S_n}$ is generated by **CM( )** , where $\mathbf{2^j\, S_j \equiv \left(\sum_{i=0}^{j-1} x_i\, 2^i\right) \pmod N}$ and $\mathbf{0 \le S_j < N}$, for j=1 to 2n. Note that the final result is $\mathbf{S_{2n}}$ instead of $\mathbf{S_n}$, **and**

$$\mathbf{R^2\, S_{2n} \equiv AB \pmod N} \qquad \text{--(2)}$$

In this algorithm, only one addition is required in each iteration. The number of iterations, however, is two times that of Montgomery's algorithm , hence the overall computation time of the iterations is about the same.The advantage of this algorithm is that the partial products in all modular multiplications fall in the correct range , hence the post adjustment in the original algorithm is removed.

It has been shown that the most significant half of the product as derived by the above algorithm can be considered separately to reduce the number of iterations.Based on that, we further improve the algorithm to enhance the computation speed.We let

$$\mathbf{X = RX_M + X_L} \qquad \text{-- (3)}$$

Where $\mathbf{X_M}$ and $\mathbf{X_L}$ are the most and least significant halves of X , respectively.Note that $\mathbf{0 \le X_L < R}$, and we can use procedure **Y( )** described below to find $\mathbf{S_n}$ such that

$$\mathbf{RS_n \equiv X_L \pmod N} \qquad \text{-- (4)}$$

FUNCTION Y (X_L,N):

Step 1: $S_0=0$

Step 2: for i=0 to n-1

Step 3: if($S_i + x_i$ is even)

$S_{i+1} = (S_i + x_i)/2$

Step 4: else

$S_{i+1} = (S_i + x_i + N)/2$

Step 5: return $\mathbf{S_n}$

Therefore, $\mathbf{X = RX_M + RS_n \pmod N}$

i.e., $\mathbf{X_M + S_n \equiv R^{-1} X \pmod N}$ -- (5)

where $\mathbf{R^{-1}R \equiv 1 (mod\ N).}$ This also holds for negative X.

Now, let A and B be (n+1)-b 2's complement numbers and

$-\mathbf{N} \le \mathbf{A,B} < \mathbf{N}$.Since $\mathbf{X = A * B}$, we have $-\mathbf{N^2} \le \mathbf{X} \le \mathbf{N^2}$.

We can represent X as a (2n+1)-b 2's complement number, i.e.,

$$\mathbf{X} = -\mathbf{x_{2n}}\ \mathbf{2^{2n}} + \sum_{i=0}^{2n-1} \mathbf{x_i}\ \mathbf{2^i} = \mathbf{R}\ \mathbf{X_M} + \mathbf{X_L} \qquad \text{-- (6)}$$

Where $\mathbf{X_M} = -\mathbf{x_{2n}}\ \mathbf{2^n} + \sum_{i=0}^{2n-1} \mathbf{x_{i+n}}\ \mathbf{2^i}$ and

$$\mathbf{X_L} = \sum_{i=0}^{n-1} \mathbf{x_i}\ \mathbf{2^i}$$

From the following inequality:

$$-\mathbf{R}\ (\mathbf{N+1}) < -\mathbf{N^2} - \mathbf{X_L} \le \mathbf{R}\ \mathbf{X_M} \le \mathbf{N^2} - \mathbf{X_L} < \mathbf{R}\ \mathbf{N}$$

We have $-(\mathbf{N+1}) < \mathbf{X_M} < \mathbf{N}$ i.e., $-\mathbf{N} \le \mathbf{X_M} < \mathbf{N}$,We define

$$\mathbf{F_N} = \mathbf{X_M} + \mathbf{S_n}, \qquad \text{if } \mathbf{X < 0}$$

$$\mathbf{X_M} + \mathbf{S_n} - \mathbf{N}, \quad \text{if } \mathbf{X \ge 0}$$

$$\mathbf{F_N} \equiv \mathbf{R^{-1}}\ \mathbf{X}\ (\mathbf{mod\ N}) \qquad \text{-- (7)}$$

Since $-\mathbf{N} \le \mathbf{X_M} < \mathbf{N}$ and $\mathbf{0} \le \mathbf{S_n} < \mathbf{N}$ we obtain

$$-\mathbf{N} \le \mathbf{F_N} < \mathbf{N}$$

The proposed modular multiplication procedure is as follows:

FUNCTION SM(A,B,N)

Step 1: $\mathbf{A \times B \rightarrow (X_M , X_L)}$

Step 2: $S_n = Y(X_L,N)$

Step 4: if ($A \times B \ge 0$)

$F_N = X_M + S_n - N$

Step 5: else

$F_N = X_M + S_n$

Step 6: return $\mathbf{F_N}$

We can start procedure Y( ) before finishing the calculation of X_L because each iteration needs only one of the product bits.Therefore, after generating the least significant bit of X_L, we can start procedure Y( ).The next iteration of Y( ) begins after the previous iteration is finished and the next product bit is generated.

Procedure SM( ) takes about n iterations to calculate F_N and each iteration requires only one addition.It is two times faster than procedure MM( ) and procedure CM( ).

The computation of X_M and F_N and the next modular multiplication can overlap.

So this algorithm is implemented by a 2's complement multiplier and a modular shifter-adder, both of which are designed as linear cellular arrays.For a word length of n, both arrays have about n cells, and each cell contains one full-adder and some controlling logic.

Montgomery Exponentiation Algorithm

Let the binary representation of E be $e_{n-1}, e_{n-2}, \ldots\ldots e_1, e_0$, then

$M^E = M^{e0}$, M^{2*e1}, $M^{pow(2,2)*e2}$ ……., $M^{pow(2,n-1)*en-1}$

-- (8)

The proposed modular exponentiation procedure is shown below, where the final value is equal to M^E (mod N).

FUNCTION SE(M,R,E,N)

Step 1: $M_0 = SM(M,R^2,N)$

Step 2: $P_0 = 1$

Step 3: for i=0 to n-1

Step 4: $M_{j+1} = SM(M_j,M_j,N)$

Step 5: if($e_j = 1$) $P_{j+1} = SM(M_j,P_j,N)$

Step 6: else $P_{j+1} = P$

Step 7: return P_n

The final result will be in the range (- N , N) , thus , post adjustment can be easily done by adding P_n with N if $P_n < 0$

III.RSA CRYPTOSYSTEM

A. Software Implementation

The speed of an algorithm implemented as a computer program is mainly determined by the total number of instruction steps to be executed. Let the central processing unit (CPU) word length be w and, without loss of generality, w divides evenly into n. Normally, an n-b arithmetic operation is partitioned into at least n/w w-b operations. For example, an n-b addition can be implemented by n/w w-b additions. An nxn multiplication may be implemented by about 2n/w w-b multiplications and some w-b additions [8]. The latter will be neglected.

Procedure SM( ) has about n n-b additions, while previousalgorithms require 2n n-b additions. Note that when using the approach of [5] complex branch instructions are then required, while in others, only simple conditional-jump (JC) instructions are needed. Also note that each n-b arithmetic operation is assumed to be realized by n/w w-b instructions. Our approach has better performance than others. Using our algorithm, the total number of instructions is $(2n^2/w) +2n+(2n/w)$.

B. Linear Bit Cellular-Array

We use an (n+1)-by-(n+1) 2's complement multiplier to implement the integer product A * B. Our linear array multiplier is based on the Baugh-Wooley 2's complement array multiplier, in which carry signals propagate from the right to left hand side.If we consider the array as a dependence graph and the projection along the (1,1) direction,we obtain the linear cellular array.The calculation of F_N and the multiplication A × B can be done by a singular cellular array, which performs multiplication and accumulation.It first generates X_L and sends to the Y-stage.After 2n clock cycles, it calculates F_N by switching the input data to X_M and S_n sequentially.

IV. EXPERIMENTAL RESULTS

In this work, 1kbyte message was encrypted using RSA algorithm with various lengths of keys ranging from 16 to 128 bits and time taken for each encryption with the respective key lengths were recorded and these values were plotted on a time taken vs key size graph. And it was observed that the time taken for encryption increases as key size increases.

Table I - Encryption of 1 kbyte using RSA

Key Size in bits	RSA - Encryption	RSA-M Encryption	RSA IM Encryption
16	155	143	71
32	186	172	84
48	239	193	89
64	278	202	98

Table II - Decryption of 1 kbyte using RSA

Key Size in bits	RSA - Decryption	RSA-M Decryption	RSA IM Decryption
16	192	157	76
32	249	197	94
48	311	223	109
64	392	251	124

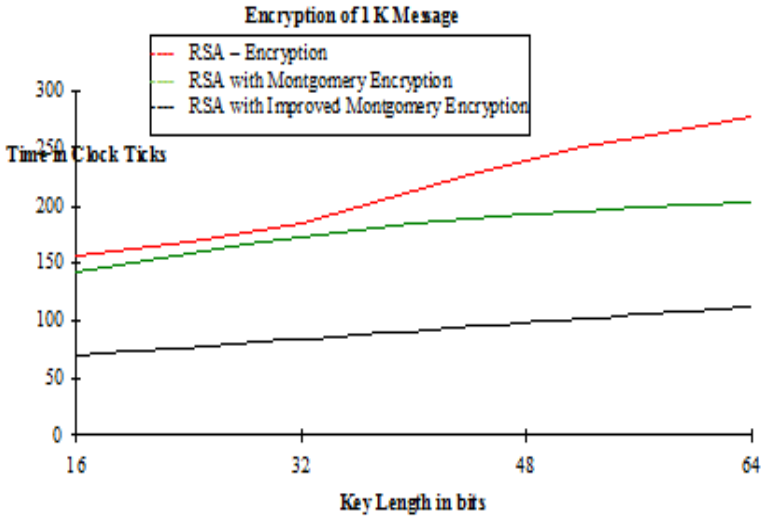

Fig.1: Figure showing results for 1k Message Encryption

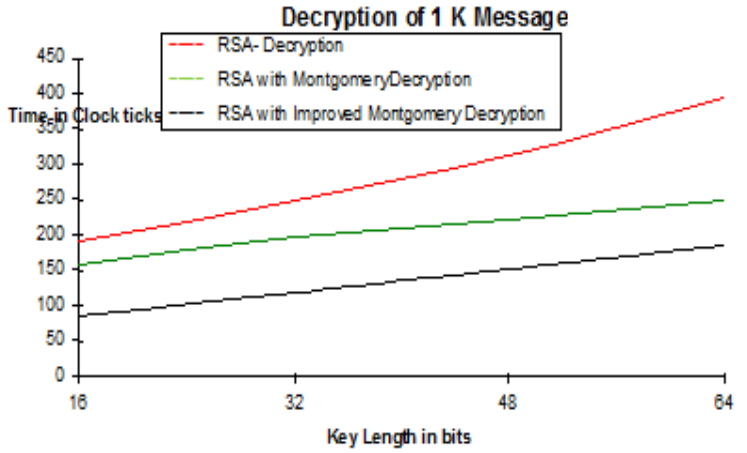

Fig.2: Figure showing results for 1k Message Decryption

The same 1kbyte was encrypted using RSA with Montgomery's algorithm and improved Montgomery's algorithm with various lengths of keys ranging from 16 to 64 bits and time taken for each encryption with the respective key lengths were recorded and plotted on the above time taken vs key size graph. And it was observed that the time taken for encryption increases as key size increases.

From the plotted graphs, it was observed that the time taken for encryption using RSA with Montgomery's method was lesser than the time taken for encryption using RSA and time taken for encryption using RSA with Improved Montgomery's method was lesser than the time taken for encryption using RSA with Montgomery's algorithm and is two times faster.

The obtained cipher texts were decrypted with corresponding private keys and time taken for each decryption with the respective key lengths were recorded and these values were plotted on a time taken vs key size graph. And it was observed that the time taken for decryption increases as key size increases.

The same cipher texts were decrypted using RSA with Montgomery's algorithm and Improved Montgomery's algorithm with various lengths of keys ranging from 16 to 64 bits and time taken for each decryption with the respective key lengths were recorded and plotted on the above time taken vs key size graph And it was observed that the time taken for decryption increases as key size increases.

From the plotted graphs, it was observed that the time taken for decryption using RSA with Montgomery's method was lesser than the time taken for encryption using RSA and time taken for decryption using RSA with Improved Montgomery's method was lesser than the time taken for decryption using RSA with Montgomery's algorithm and is two times faster.

V. CONCLUSION

The proposed new Montgomery's modular arithmetic, and a low-cost high-speed cellular array for modular exponentiation which implements the RSA cryptosystem outperforms the previous algorithms and also the proposed array is two times faster than those based on the original Montgomery's algorithm.

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] T. El Gamal, "A public key cryptosystem and a signature scheme based on discrete logarithms," *IEEE Trans. Inform. Theory*, vol. IT-31, pp.469–472, July 1985.
[3] I. Niven, H. S. Zuckerman, and H. L. Montgomery, *An Introduction to the Theory of Numbers*. New York: Wiley, 1991.
[4] E.-H. Lu, L. Harn, J.-Y. Lee, and W.-Y. Hwang. "A programmable VLSI architecture for computing multiplication and polynomial evaluation modulo a positive integer," *IEEE J. Solid-State Circuits*, vol. 23, pp. 204–207, Feb. 1988.
[5] P. W. Baker. "Fast computation of A*B modulo N," *Electron. Lett.*,vol. 23, no. 15, pp. 794–795, July 1987. 284 IEEE TRANSACTIONS ON VERY LARGE SCALE INTEGRATION (VLSI) SYSTEMS, VOL. 7, NO. 2, JUNE 1999
[6] P. L. Montgomery, "Modular multiplication without trial division,"*Math. Comput.*, vol. 44, pp. 519–521, 1985.
[7] S. E. Eldridge, "A faster modular multiplication algorithm," *Int. J.Comput. Math.*, vol. 40, pp. 63–68, 1991.
[8] D. E. Knuth, *Seminumerical Algorithms, The Art of Computer Programming, vol. 2*, 2nd ed. Reading, MA: Addison-Wesley, 1981.