Tabulated Modular Exponentiation (TME) Algorithm for Enhancing RSA Public –Key Encryption Speed

Hamza Ali and Hamed Fawareh
Department of Computer Science, Zarka Private University, Jordan

Abstract: *Improving software algorithms are not easy task, especially for increasing operating speed, and reducing complexity. Different algorithms implemented in cryptosystems used the exponentiation modular arithmetic, however, they suffer very long time complexity. Therefore, faster algorithms are strongly sought. This paper provides fast algorithms for modular multiplication and exponentiation that are suitable for implementation in RSA and DSS public key cryptographic schemes. A comparison of the time complexity measurements for various widely used algorithms is performed with the aim of looking for an efficient combination for the implementation of RSA cryptosystem. Two such algorithms were proposed in this work. The first is a modified convolution algorithm for modular multiplication while the second is a Tabulated Modular Exponentiation (TME) algorithm based on the modified sign-digit algorithm. They are found to give significant overall improvement to modular exponentiation over that of the fastest algorithms studied.*

Keywords: *Data security, cryptography, authentication, algorithms, RSA, DSS.*

Received February 2, 2003; accepted May 11, 2003

1. Introduction

The last decade of the twentieth century has seen a rapid increase in the use of cryptographic algorithms implementation. This trend is expected to continue in the next millennium in the light of continuing progress and utilization of the public-key cipher technology particularly to obtain separate keys for encryption and decryption. This will lead to provide tremendous improvement for both data secrecy and digital signature applications.

Cryptographic algorithms make use of secret keys known to send and receive information. When the keys are known the encryption/decryption process is an easy task, however decryption will be impossible without knowing the correct key. The essential of such method is security of sending the secret key to the receiver, which should be protected from intrusion of cryptanalyst (i.e. cryptanalysis is the process of attempting to break the secret communication systems when the key is unknown). Public key cryptosystem proposed by Diffie and Hillman [1] provided a mean to avoid sending secret keys over communication channels, i.e. reducing the risk of key compromise. Many algorithms are introduced based on the public key cipher [2, 3, 4, 5, 6, 7, 8]; the most widely known and used of them is the RSA crypto-algorithm. These algorithms are suitable for privacy and authentication [9].

RSA crypto-algorithm, proposed by Rivest, Shamir and Adleman [2] is based on performing exponentiations in modular arithmetic both for encryption and decryption. It makes use of the fact that finding large prime number is comparatively easy, but factoring the product of two such primes is computationally infeasible with the current known algorithms and computation speeds. Such property has made RSA technique very attractive for data security and authentication. However, the main drawback of the RSA cryptosystem lies in the slow computation speed or the time complexity of encryption/decryption operations.

Many authentication schemes involve computation of modular exponentiation, achieved by number of methods that include modular multiplication, such as RSA, El Gamal and Fiat-Shamir signature schemes. RSA scheme easily produces signature because it is a bijection. ElGamal signature scheme [10, 11] is also based on the difficulty of solving the discrete logarithm problem, but the size of the signature in this scheme is double the size of the document. The signing procedure uses three exponentiations to verify the signature for the same level of security, while the public key size and cipher-text will be double that for the RSA system. This paper deals with RSA scheme for the same level of security. The size of public key and digital signature for RSA scheme is shorter than ElGamal's scheme [10]. The scheme developed by Fiat and Shamir [12, 13] for digital signature also uses modular mathematic. It has shown remarkable improvement over RSA in terms of computation time complexity but at the cost of size. The algorithms

considered in this paper aim to arrive at reasonable speed for security and authenticity application. This is achieved by having good combination of algorithms used for RSA scheme, which is briefly outlined in section 2. In addition, fast algorithms commonly implemented for the RSA cryptosystems are compared in section 3. Then two modified algorithms are suggested in section 4, with the aim of reducing the execution speed, i.e. reducing the required time for ciphering and deciphering. The time complexity measurements are included in section 5, and finally conclusions are drawn in section 6.

2. RSA Scheme

The system designers of RSA cryptosystem begin with two large, distinct prime integers, say p and q for each user. Then an integer N is calculated as $N=pq$. Furthermore, another integer e is subsequently chosen relatively prime to $(p-1)(q-1)$. Then integer d is computed as the multiplicative inverse of e modulo $(p-1)(q-1)$. Parameters, N and e are taken as the public key while d is taken as the private key for the user. Therefore any user will have his/her own private key and a published public key.

The major problem in implementing the RSA system is the long time required for enciphering and deciphering algorithms. For this reason, RSA system is found more suitable for digital signature rather than data security. However, as we are interested in improving the computation algorithms, we will only consider encryption and decryption techniques, as outlined in Figure 1.

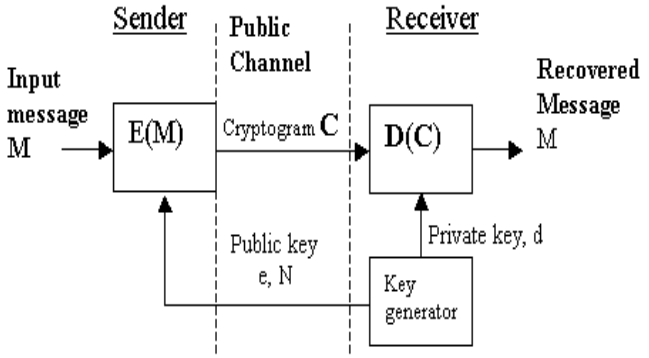

Figure 1. RSA cryptosystem.

To encrypt any message, M by RSA scheme, the public keys e and N are used at the sender end; the message M is chopped into a sequence of blocks as $M_1, M_2, \ldots, M_k$, where each block is represented by an integer M_i having values between 0 and N-1. These blocks are enciphered by the encryption algorithm E giving the ciphered text as the sequence of cryptograms $C = C_1, C_2, \ldots, C_k$, where

$$C_i = E(M_i) = (M_i)^e \bmod N \quad (1)$$

At the receiver end, using his private decryption key, d and N, the intended recipient of the message decrypts the cryptograms by computing

$$D(C_i) = (C_i)^d \bmod N \quad (2)$$

When the whole sequence $C_1, C_2, \ldots, C_k$ is deciphered, the results are concatenated together, the message M is recovered [3].

Obviously after generating the public and private keys e, N and d, basically the RSA scheme involves computations of the modular multiplication and exponentiation, which will be outlined in the next section.

3. Modular Exponentiation

Two algorithms that facilitate the software implementation of RSA scheme are described here. Both algorithms are concerned essentially with modular exponentiation of very large integers. They are the *Repeated Square and Multiply* (RSM) algorithm and the *Modified Signed-Digit* (MSD) algorithm [9]. They are both based on modular multiplication algorithms described in the following subsections.

3.1. Modular Multiplication Algorithms

Various ways are available for performing the modular multiplication. However, two widely used methods are considered here.

1. The Combination Method, which combines two algorithms one for multiplication and one for the remainder.
2. The Synthesis Method, which, employs one algorithm for both the multiplication the remainder.

Combination Method

The Combination Method performs modular multiplication of integers by combining two algorithms, one for multiplying the numbers and the other for reducing the resulting product. These algorithms are outlined in the following.

a. Multiplication Algorithms

To obtain the product of two large integers, several fast algorithms are available [4, 14, 18]. In this paper, two such algorithms are chosen, namely (1) *Convolution* and (2) *Divide and Conquer*. As for the convolution algorithm, the product P of two n-word integers x & y (for example, each word is of 16-bit length) can be calculated by:

$$P = x[0]*y[0] + \sum_{i=1}^{n=1} \sum_{j=0}^{i} x[i]*y[i-j]*2^{16j} \quad (3)$$

Where x[0], x[1], x[2],…,x[n-1] and y[0], y[1], y[2],…, y[n-1] are the individual words of the two numbers. Hence, words x and y are:

$$x = x[0] + x[1]*2^{16} + x[2]*2^{32} + ... + x[n-1]*2^{16(n-1)}$$
$$y = y[0] + y[1]*2^{16} + y[2]*2^{32} + ... + y[n-1]*2^{16(n-1)} \quad (4)$$

The number of multiplication operations in equation 3 is n^2.

However, for the Divide and Conquer algorithm, the problem is partitioned into smaller parts, solved and then recombined. For example if x and y are two n-word numbers (n is a power of 2), then they can be partitioned into two equal halves each, i.e. x_0, x_1, y_0 and y_1 of n/2 word length each, as follows:

$$x = x_0 + 2^{16n/2} * x1$$

$$y = y_0 + 2^{16n/2} * y1 \quad (5)$$

Then, the product P of x and y of equation 5, results into equation 6

$$P = x_0 y_0 + (x_0 y_1 + x_1 y_0) * 2^{16n/2} + x_1 y_1 * 2^{16n} \quad (6)$$

This algorithm reduces the number of multiplications required by 25%, because only three products of n/2 numbers are required instead of four. However, there is an extra time for the carry bits and overhead divisions and additions, which may not be ignored.

b. Modulus Algorithm

Two algorithms are also studied here [2]. They are (1) *Iterative division* and (2) *Modular reduction*. For the iterative division algorithm [14, 15], a 2n-word number x is reduced to n-word number x' by iteratively dividing by *N* and computing the remainder R_i until no remainder is left. i.e.

$$x' = x \bmod N$$
$$or \quad (7)$$
$$x' = (R_0 + R_1 + ... + R_i) \bmod N$$

Where *N* is an n-word number and R_0, R_1, … , R_i are the remainders after division by *N*.
The speed of running this algorithm depends on the property of the modulus *N*.

On the other side, to speed up the computation of modular reduction algorithm, "local" and 'global" fixed values are utilized to pre-compute "look up" table. The local fixed values depend upon a, in the form of a^s mod *N*, where a is fixed for the duration of the encryption computation and s is the number of bits in the modulus *N*, which depends on the key and therefore is fixed for a number of encryption operation. The table atab[i] consists of 512 entries; each consists of an (n+1)-words. The computation of this table may be achieved by simple combination of additions and comparisons. The idea of the algorithm is to reduce the length of 32n-bit number, x by 8 bits at a time to obtain 16n-bit number x'. This is done according to the following:

The table elements are

atab[0] = 0
atab[i] = g * *N* , (for i=1 to 511)
{where g is unique integer satisfying, int (g**N*/T) = i-1, and int ((g+1)**N*/T) = i, T = 2^s, s is number of bits in *N* or g = int (T*i/*N*)}

Then the individual computation records as follows:

$$j = \text{int}(x / 2^{8i} * T)$$
$$x' = x - 2^{8i} atab[j] \quad (8)$$

The table computed only at the time of modulus change. The time complexity for these algorithms will be computed later.

Synthesis Method

The modular multiplication is obtained by using one algorithm for both multiplication and modulus. Two algorithms where tested and their time complexity is compared. They are (1) *Repeated Shift and Add* algorithm and (2) *Combined* algorithm. The first, is suggested by Sloan [6], used to obtain modular multiplication by multiplying two n-word numbers a & b, then reduce the result modulo an n-word number *N* to obtain an n-word number C, by successive shifting and addition. The second Combines both convolution and modular reduction algorithms described earlier. The obtained time complexity for both techniques were compared and reported in later sections.

3.2. Modular Exponentiation Algorithms

RAS algorithms use a large integer number *N* exponential modulo to performed the encryption and decryption tasks. Two of the widely used algorithms that facilitate the software implementation of RSA cryptosystem are also compared here. They are: (1) *The Repeated Square and Multiply, (RSM) algorithm* [6] and (2) *The Modified Sign Digit, (MSD) algorithm* [7, 19].

For the RSM algorithm, if one is required to compute a^e mod *N*, where a, e and *N* are n-word input numbers, x is an n-word output number. Each word is s-bits long. X is computed by the following:

x =1;
For x = s-1 down to 0 do
x = x^2 mod *N* (9)
If e_i = 1 then
x = x * a mod *N*; (10)

Therefore, the number of multiplications in this algorithm depends on the number of 1's in e. However, for MSD algorithm, the number representation of the entries consists of –1, 0 and 1. i.e. any number may have many MSD representations as compared with the fixed binary system. Since the number of modular multiplications in the algorithm depends upon the number of non-zero entries in the MSD representation of e, which can be made less than binary by using weight minimization technique [7]. This has speeded up the process, as will be seen later in the comparison computation listed in section 5.

4. The Proposed Algorithms

To improve the modular exponentiation execution speed, two algorithms were suggested in this section. The first one is a *Modified Convolution Algorithm* which is suitable for modular multiplication, while the second one is a *Tabulated Modular Exponentiation* which is a modified version of Selby algorithm found suitable for the modular exponentiation computation [14]. Both algorithms, together with the pseudo-code description are outlined below.

$p = (a [0])^2$

$$P = p + \sum_{i=1}^{n-1} \{ (a[i])^2 * 2^{2*16i} + 2^{16i+1} * \sum_{j=0}^{\text{int}(i-1)/2} a[j] * a[i-j] \}$$

$$P = p + P = p + \sum_{i=n}^{2n-2} 2^{16i+1} \sum_{j=i-n+1}^{\text{int}(i-1)/2} a[j] * a[i-j]$$

```
Modified Convolution Algorithm
begin
let a(an n-words number) as input
n = size;
let p= procedure1 (n, a[])+procedure2
            (2*n+2, a[], size);
output: p (a (2n-1)-words number)
end

double procedure1 (an n-words number, a[])
begin
if n is less than zero then
return zero
else
let s= return call procedure3 (an n-words
            number, a[])*power(2, 16*n+1)
return s+ power (a[n], 2) * power (2, 2*16*n)
            + return call procedure1(n-1,
            a[]);
end

double procedure3 (an n-words number, a[])
begin
loop from i = 0 to number/2
s=a[i] * a[number -i]
end loop
return s
end

double procedure2 (i, a[], size)
begin
if n is greater than  (2*size+2)
return zero
else
begin
let s = 0;
loop from j=size +1 to  j<=(i-1)/2 steps 1
let s = s+ a[j]*a[i-j];
end
let s=s+ pow(2, 16*i+1);
return s + return call procedure2 (i-1, a[],
            size);
end
```

4.1. Modified Convolution Algorithm

This algorithm is based on the utilization of a symmetric property found in squaring a function. It applies the convolution principle; the square of any binary function follows a symmetric property that would reduce the number of multiplication operations as follows:

Suppose a is an n-bits binary number, whose decimal value is given by

$$a = a_0 + a_1 * 2 + a_2 * 2^2 + ... + a_{n-1} * 2^{n-1} \quad (11)$$

Finding the square of a, results into

$$\begin{aligned} a^2 &= (a_0)^2 + (a_0 a_1 + a_1 a_0) * 2 + (a_0 a_2 + a_1^2 + a_2 a_0) * 2^2 \\ &+ ... + (a_0 a_{n-1} + a_1 a_{n-2} + ... + a_{n-2} a_1 + a_{n-1} a_0) * 2^{n-1} + ... + \\ &(a_{n-2} a_{n-1} + a_{n-1} a_{n-2}) * 2^{2n-3} + (a_{n-1})^2 * 2^{2n-2} \end{aligned} \quad (12)$$

Notice that the number of multiplication operations for this convolution algorithm is n^2. By applying the commutative theorem to equation 12, we arrive at

$$\begin{aligned} a^2 &= (a_0)^2 + (a_0 a_1 shl1) * 2 + (a_0 a_2 shl1 + a_1^2) * 2^2 + \\ &... + ((a_0 a_{n-1} + a_1 a_{n-2} + ... + a_{(n/2)-1} a_{n/2}) * shl1) * 2^{n-1} + \\ &.... + (a_{n-2} a_{n-1} shl1) * 2^{2n-3} + (a_{n-1})^2 * 2^{2n-2} \end{aligned} \quad (13)$$

Where n is an even number and shl1 means shift left by 1 digit.

Suppose X be the required number of multiplication operations for equation 13, then,

$$X = 1+1+2+2+...+n/2+...+2+2+1+1 \\ = 4*(1+2+3+...+n/2)-n/2 \quad (14)$$

But (1 + 2 + 3 + . . . + n/2) = (n/2 * n/4 + n/4)

Therfore

$$X = (n^2+n)/2 \quad (15)$$

```
for i = 1 to n-1 do begin
      btab[i]  = btab [i-1] Shl 16
```

Form equations 13, 14, and 15, we managed to reduce the number of multiplications by applying commutative theorem from n^2 to $(n^2 + n)/2$, i.e. it is half the number of multiplication operations required convolution algorithm as well as for Blakley multiplication algorithm [16]. Furthermore, the same result will be obtained for
odd numbers.

4.2. Tabulated Modular Exponentiation Algorithm

This algorithm utilizes a tabulation technique in order to improve the computation speed of modular exponentiation. We will refer to it as Tabulated Modular Exponentiation, (TME**)** algorithm. It is achieved by implementing tables within the acceptable range of the microcomputer. The main idea in this algorithm is to reduce the size of multiplication results for n-word numbers from (2n-1)-word in convolution algorithm to (n+2)-word in our proposed algorithm. Since the modulus of (n+2)-word number is faster than the modulus of (2n-1)-word number, the TME algorithm is more efficient to use instead of convolution algorithm for multiplication. The table entries of this algorithm are of the form:

btab [i] [j], 0<= i <= n-1, 0<= j <= n-1 (16)

The pseudo-code and recursive representation for this algorithm are described below.

```
/* n represents the words number
   btab[][] represent the tabulated value */

let b (an n-words number ), a (an n-words number ),
      N (an n-words number) as input.
output: p (an (n+2)-words number ).
```

The table of this algorithm is computed by the following procedure:

```
table()
begin
btab [0] = b;
   else
btab [i] = btab [i] mod N;
(using the modular reduction algorithm)
The size of this table is n^2-words.
end
```

Using the table algorithm, the TME_algorithm is computed as follows:

```
double TME_Algorithm(n, a[], btab[][], size)
begin
   if (n ==0)
      return 0;

{
      s = 0;
   for(j=0; j<m; j++)
       s = s+a[n]*btab[n][j]* pow(2, j*16);
}
    return s+TME_Algorithm(n-1, a[], btab[][], size);
end
```

The correctness of this algorithm can be proven as follows:

$$P \bmod N = a*b \bmod N = (a[0]*b + a[1]*b*2^{16} + \\ .... + a[n-1]*b*2^{16(n-1)}) \bmod N$$

$$= (a[0]*b \bmod N + a[1]*b*2^{16} \bmod N + \\ ... + a[n-1]*b*2^{16(n-1)} \bmod N) \bmod N$$

$$= \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} a[i]*btab[i][j]*2^{16j} \bmod N \quad (17)$$

Hence $P == \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} a[i]*btab[i][j]*2^{16j}$

5. Results

The various algorithms outlined in this paper were studied and compared with each other by writing computer programs for each, with the aim of measuring the time required for their execution (the time complexity).

In order to insure fair comparison of the computation time between various algorithms considered in this work, the same software and hardware environment were used, namely, all calculations were performed by programs written in Turbo Pascal language that were

running on a Pentium II PC. The results are displayed in Figures 2-5 and briefly described in the following. The time complexity for modular multiplication of two 32-bit integer numbers using convolution algorithm and divide and conquer algorithm are shown in Figure 2. It shows that the convolution algorithm is superior to divide and conquer algorithm by a factor of 5, despite the fact that the number of multiplications in the latter is 25% less than the former. This is mainly attributed to the time needed for processing the carry bit in the divide and conquer algorithm.

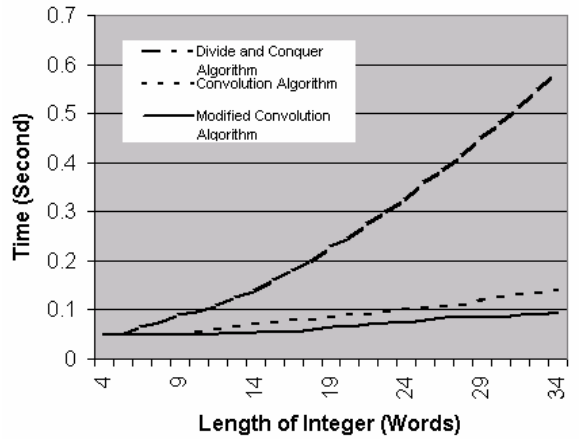

Figure 2. Time complexity for multiplication algorithm multiplication algorithm.

The time complexity for modulus algorithms used to reduce 2n-bits number to n-bits number by iterative division and modular reduction techniques are shown in Figure 3. It shows that the modular reduction algorithm is faster than the iterative division algorithm. For example, the time required to reduce 64-bits number to 32-bits number is 2.8 seconds and 0.11 seconds, respectively.

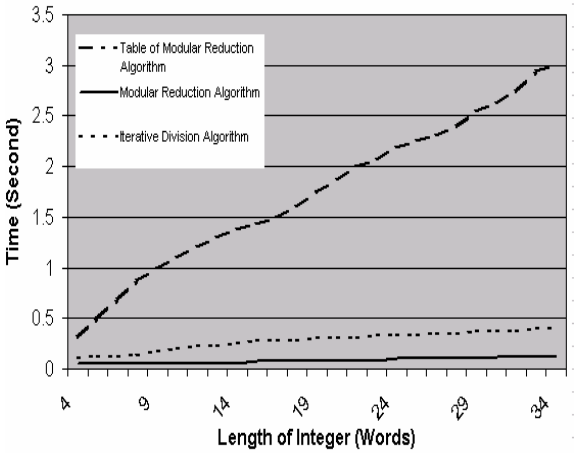

Figure 3. Time complexity for modulus algorithm.

The second method for modular multiplication considered here involves combined techniques for multiplication and modulus. It included two algorithms, namely: (1) Combined convolution and modular reduction algorithm and (2) Repeated shift and add algorithm.

Figure 4 shows that the time complexity measurements for these two techniques. It points out that convolution and modular reduction algorithm is much faster than repeated shift and add algorithm, for example the time required for one modular multiplication of 32-bits numbers is 0.16 seconds and 1.6 seconds for the two algorithms, respectively.

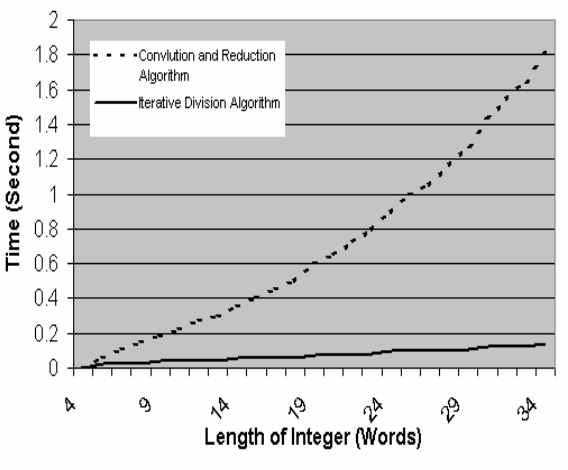

Figure 4. Time complexity of modular multiplication.

Therefore, the combined convolution and modular reduction algorithm is selected to be used in the modular exponentiation of the proposed algorithms.

Figure 5 shows the time complexity for the modular exponentiation algorithm outlined in section 3.2. It shows that the Modified Signed-Digit (MSD) algorithm is slightly faster than the Repeated Square and Multiply (RSM) algorithm. This can be attributed to the more efficient digit representation of MSD.

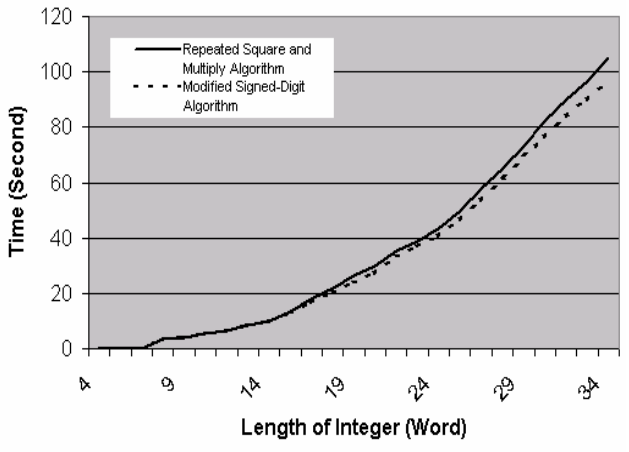

Figure 5. Time complexity for modular exponentiation.

When the suggested algorithm is implemented, the time complexity is calculated and plotted as curve marked 3 in Figure 2. In comparison with both divide and conquer algorithm and convolution algorithm, it shows an improvement of 16.4% in time complexity over the more efficient algorithm of the two, which is the convolution algorithm.

When the TME algorithm suggested in this paper is used for multiplication in the modified sign-digit algorithm, the execution time for one modular exponentiation became shorter, for example, it was 77.56 seconds instead of 88.85 seconds, using the

same computation environment. This improvement was of the order of 12.7%.

Finally, the total improvement in speed (or reduction in execution time) for the modular exponentiation when using both of the suggested algorithms together is 25.9%. This will obviously lead to a considerable improvement in the overall computation time required for any data security system.

6. Conclusion

Comparison study of the time complexity for fast algorithms suitable for RSA cryptosystem implementation is conducted. Two modified algorithms were suggested. The first is a modified convolution algorithm used for multiplication and the second is a modified Selby algorithm for modular exponentiation. The overall improvement obtained by using the suggested TME algorithm over the fastest studied algorithms was 25.9 %, which may be considered of great benefit.

However, the implementation of RSA cryptosystem is still slow which makes it more suitable for use in hybrid encryption schemes, where RSA can be used in conjunction with other algorithms. The best choice seams to be that RSA is used for key distribution and other schemes like DES, AES or IDEA are used for data security [10, 11, 17].

References

[1] Diffie W. and Hellman M.E., "New Direction in Cryptography," *IEEE Trans. Information Theary*, vol. IT-22, no. 6, pp. 644-654, 1976.

[2] Rivest, R., Shamir A., and Adleman L., "Method for obtaining Digital Signature and Public Key Cryptosystem," *Comm. of ACM*, vol. 21, no. 2, 1978.

[3] Mohan S. B., "Fast Algorithm for Implementing RSA Public-Key Cryptosystem," *Electronic Letters*, vol. 21, no. 17, 1985.

[4] Robert S., *Algorithms*, Addison Wesley, USA, 1983.

[5] Selby A., "Algorithms for Software Implementation of RSA," vol. 136, no. 3, 1989.

[6] Sloan, K. R., "A Computer Algorithm for Calculating A*B mod M," *IEE Trans. on Computers*, vol. C-34, no. 3, 1985.

[7] Davis D. W., *Security for Computer Networks*, Wiley, UK, 1989.

[8] Saloma A., *Public-Key Cryptography*, Springer, 1996.

[9] Jedwab J. and Mitchell C. J, "Minimum Weight Modified Signed-Digit Representations and Fast Exponentiation", *Electronic Letters*, vol. 25, no. 17, 1989.

[10] ElGamal T., "A Public Key Cryptosystem in a Signature Scheme Based on Discrete Logarithm," *IEEE Trans Information Theory*, vol. IT-31, no. 4, 1985.

[11] Boneh D. and Franklin M., "Efficient Generation of Shared RSA Keys," *J. of ACM*, vol. 48, no. 4, pp. 702-722, 2001.

[12] Schneier B., *Applied Cryptography*, John Wiley & Son, 1996.

[13] Naini S. R. and Wang H., "Broadcast authentication for Group Communication," *Theoretical Computer Science*, vol. 269, pp. 1-21, 2001.

[14] Rashid A. M. T., "Software Inplementation of RSA Public-Key Cipher for Microcomputer Protection and Authentication," *MSc Thesis*, Basrah University, 1992.

[15] Campbell R., "Basics of Computational Number Theory," http://www.math.umbc.edu.~campbell/NumbThy/Basic_NumberThy.html, last visited 11/5/2002.

[16] Cornam T., Leiserson C.E. and Rivest, R. L., *Introduction to Algorithms*, McGraw Hill Company, 1999.

[17] Bellare M; Garang J. A. and Rabin T., "Fast Batch Verification for Modular Exponentiation and Digital Signature," *Advances in Cryptography, Eurocrypt98 Proceedings, Lecture Notes in Computer Science*, vol. 1403, 1998.

[18] Aho A., Hopcroft J. and Ullman J., *The Design and Analysis of Computer Algorithms*, Addison-Wesley, 1974.

[19] Blum T. and Paar C., "*High-Radix Montgomery Modular Exponentiation on Reconfigurable Hardware*," *IEEE Trans. on Computers*, http://www.computer.org/tc/tc2001/t0759abs.htm

Hamza Ali received his BSc in physics from University of Basrah in 1968, MSc in electronics from University of London in 1973 and PhD in computer engineering from University of London in 1977. Previously he worked in Basrah University, Iraq, one year in Aizu University, Japan and currently associate professor at the Computer Science Department, Zarka Private University, Jordan. His research interest includes computer security and authentication, pattern recognition and neural networks.

Hamed Fawareh received his PhD in software engineering from University Putra Malaysia, 2001. Currently, he is an assistance professor at the Department of Computer Science, Zarka Private University, Jordan. His research areas include software engineering, algorithm security, and Biocomputing.