A Memory Optimized Public-Key Crypto Algorithm Using Modified Modular Exponentiation (MME)

A. O. Oluwatope, B. A. Ojo, and G. A. Aderounmu
Comnet Laboratory, Computer Science And Engineering,
Obafemi Awolowo University Ile-Ife. Nigeria

M. O. Adigun
Computer Science, University of Zululand. South-Africa

aoluwato@oauife.edu.ng, ojo_d@yahoo.com,
gaderoun@oauife.edu.ng & madigun@pan.uzulu.ac.za

Abstract

Since the advent of data communication over networks, it has become imperative to ensure security of information. Cryptography is a technique that is being employed. This paper takes a look at an important aspect of the public key encryption scheme, the modular exponentiation technique, with the view of optimizing it. Taking a look at some public key encryption schemes, it would be observed that the modular exponentiation process is primal to achieving high speed algorithms in data encryption. With special emphasis on the Montgomery exponentiation algorithm, a blend of this algorithm with the sliding window method of exponentiation is proposed. A detailed complexity analysis of the proposed and selected algorithms was carried out. Both algorithms were implemented and simulated using MATLAB 6.5. While the proposed algorithm did not prove to be faster than the classical Montgomery exponentiation algorithm, it was rather observed that it makes lesser number of calls to the Montgomery reduction sub-function. This means 10% lesser number of loops during execution and thus better optimized for lower memory applications.

Keywords Public-key encryption, modular multiplication, modular exponentiation, sliding window, algorithm

Introduction

With the awareness that the world is fast becoming a global village due to the increased application of the internet especially in the area of e-commerce and m-commerce, information security is an issue that cannot be overemphasized. Since the onset of e-commerce in the mid 90s, statistics have it that e-commerce is expected to generate a total sales of about $105 billion in the United States alone by 2007. Statistics further shows that as of 2001, over $700million has been lost to online sales (Holcombe, 2006). Most often times, ecommerce customers are often concerned about the authenticity of the online trader they are dealing with (data origin authenticity) and more importantly, the privacy of their information

(data confidentiality); such as credit card details, which is being used for the transaction.

Based on these statistics, it has become very important to improve techniques used for information security. New techniques have been developed and newer techniques are evolving. Cryptography is one of such techniques being used to ensure information security. It would be pertinent to seek ways of improving this technique as adversaries are daily developing newer techniques of breaking information security systems. Based on this, it has become pertinent to ensure security of information. Data encryption using cryptographic techniques is a means of achieving this.

Furthermore, with respect to an already developed application in our laboratory- (the M^2 agent for the stock market) (Oluwatope, Aderounmu, Akande, & Adigun, 2004), an important aspect yet to be added is security of data traversing the network. The mobile agent in the stock market application traverses the Internet searching for stock deals based on certain criteria set by the stock broker. When such criteria are met, the mobile agent reports back to the static agent. Whenever a mobile agent is reporting back to the static agent, the data it is carrying along can be intercepted and altered if proper encryption scheme is not put in place. It will not be a out-of-place to mention that the proposed algorithm is applicable in digital signal processing (DSP) (Tang et al, 2003). In DSP, coding of digitised signals for optimised transmission, higher bandwidth utilisation and efficient processor-time utilisation is highly desirable. The outcome of this work will form a baseline input into algorithms used for coding and decoding in DSP.

Cryptography is about communication in the presence of adversaries (Rivest, 1990). The reason for cryptography is to ensure security as information traverses an unsecured channel. Some of the objectives of information security include privacy, confidentiality, data integrity, data origin authentication, entity authentication, non-repudiation (Rivest, 1990). Basically, cryptography seeks to address these objectives in theory and in practice. Cryptography is defined as the study of mathematical techniques related to aspects of information security such as confidentiality, data integrity, entity authentication, and data origin authentication (Menezes, Van Oorschot, & Vanstone, 1997).

Motivation

As mentioned earlier, in the course of trying to achieve security of information, some cryptographic tools are used. These tools transform the information such that the encrypted information becomes meaningless to an adversary. The transformation process involves the execution of some computational steps. It has also been observed that the computational speed of some of these public-key cryptosystems is based on the speed of the modular exponentiation algorithm, especially when considering public-key encryption schemes such as RSA, Rabin and EIGamal. Furthermore, based on the fact that part of the security of these systems is the use of very large integers in the range of 1024bits and above, it usually requires enormous amount of processor time to complete these computational steps.

In the study of complexity theory (classification of computational problems based on the resources required to solve them), one of the resources used to classify computational problems is time. It is a thing of interest to look for the most efficient algorithms for solving computational problems because time is a very critical resource. More so in the e-commerce world that deals with transaction of businesses over the internet, *time can be equated to money.* This serves as the basis of motivation for this work.

Public-key encryption is slower than symmetric-key encryption which is been used for bulk data encryption. This is also due to the encryption/ decryption processes. It would be pertinent to optimize the former because if it has a comparable speed to symmetric-key encryption, lesser resources would be utilized in the transportation of private information over the Internet. In Tang Tsui, and Leong (2003) montgomery multiplier and a semi-systolic pipeline scheme have been

used to increase the clock rate and enhanced parallelism in Field-Programmable Gate Arrays (FPGAs) devices. In Hachez and Quisquater (2001) the authors showed from software perspective montegomery exponentiation speed can be increased as well. In Joseph and Penzhorn (2004), the authors extended analysis to previous works and reduced running time of modular exponentiation step required for public–key encryption algorithms. Yesufu and Amao (2004) succeeded in producing a hybrid scheme of Knapsack algorithm, symmetric key cipher and pseudo-random sequencing. We will show, from a software view, a modified montgomery exponentiation that makes lesser number calls to sub-functions. Also, we will show analytically that the algorithm is optimized arithmetically.

Hence, this paper seeks to develop an optimized public-key crypto algorithm. In order to achieve our goal, we set these tasks: - propose and model an optimized montgomery exponentiation algorithm, and compare the proposed algorithm with the classical montgomery exponentiation.

Review of Selected Public-Encryption Schemes

To show the relevance of this work to the optimization of public-key encryption schemes, a review of selected public-key encryption schemes was carried out. It would be noticed that the modular exponentiation process is a core aspect these encryption schemes. The encryption schemes are RSA, Rabin and EIGamal public-key encryption schemes.

RSA Public-key Encryption Scheme

The algorithms used for the RSA public-key encryption are shown below (Menezes et al, 1997)

Key generation algorithm

Each entity is expected generate a key pair (public and private key). Ayo does the following:

1. Generate two large random (and distinct) primes p and q, each roughly the same size.
2. Compute $n = pq$ and $\Phi = (p - 1)(q - 1)$.
3. Select a random integer e, $1 < e < \Phi$, such that gcd $(e, \Phi) = 1$.
4. Use the binary extended gcd algorithm to compute the unique integer d, $1 < d < \Phi$, such that $ed \equiv 1 \pmod{\Phi}$.
5. Ayo's public key is (n, e); the corresponding private key is d.

Encryption/ Decryption algorithm

---

1. *Encryption:* Ben does the following:
   a) Obtain an authentic copy of Ayo's public key (n, e).
   b) Represent the message as an integer m in the interval $(0, n - 1)$.
   c) Compute $c = m^e \bmod n$.
   d) Send the ciphertext c to Ayo.
2. *Decryption:* To recover plaintext m from c, Ayo should do the following:
   a) Use the private key d to recover $m = c^d \bmod n$.

Rabin Public-key Encryption Scheme

The algorithms used for the Rabin public-key encryption are shown below (Menezes et al, 1997).

Key generation algorithm

Each entity is expected generate a key pair (public and private key). Ayo does the following:

1. Generate two large random (and distinct) primes p and q, each roughly the same size.
2. Compute $n = pq$.
3. Ayo's public key is n; his corresponding private key is (p, q).

Encryption/ Decryption algorithm

1. *Encryption:* Ben does the following:
   a) Obtain an authentic copy of Ayo's public key n.
   b) Represent the message as an integer m in the range $\{0, 1, \ldots, n-1\}$.
   c) Compute $c = m^2 \bmod n$.
   d) Send the ciphertext c to Ayo.
2. *Decryption:* To recover plaintext m from c, Ayo does the following:
   a) Use an algorithm for finding square root modulo n to find the four square roots m_1, m_2, m_3, and m_4 of c modulo n.
   b) The message sent was either m_1, m_2, m_3, or m_4. Ayo somehow determines which is m.

ElGamal Public-key Encryption Scheme

The algorithms of this scheme as regards to its functionality are shown below.

Key generation algorithm

Each entity is expected generate a key pair (public and private key). Ayo does the following:

1. Generate a large random prime p and a generator α of the multiplicative group Z^*_p of the integers modulo p.
2. Select a random integer a, $1 \le a \le p-2$, and compute $\alpha^a \bmod p$.
3. Ayo's public key is (p, α, α^a); his corresponding private key is a.

Encryption/ Decryption algorithm

1. *Encryption*: B should do the following:
   a) Obtain an authentic copy of Ayo's public key (p, α, α^a).
   b) Represent the message as an integer m in the range $\{0, 1, \ldots, p-1\}$.
   c) Select a random integer k, $1 \le k \le p-2$.
   d) Compute $\gamma = \alpha^k \bmod p$ and $\delta = m \cdot (\alpha^a)^k \bmod p$.
   e) Send the ciphertext $c = (\gamma, \delta)$ to Ayo.
2. *Decryption*: To recover plaintext m from c, Ayo should do the following:
   a) Use the private key a to compute $\gamma^{p-1-a} \bmod p$ (note: $\gamma^{p-1-a} = \gamma^{-a} = \alpha^{-ak}$).
   b) Recover m by computing $(\gamma^{-a}) \cdot \delta \bmod p$.

As seen in the algorithms described above, the modular exponentiation process plays an important role in the speed of these algorithms.

Review of Exponentiation Methods

In order to get a proper understanding of the subject matter of this paper, we are taking a look at some of the existing exponentiation methods.

M-ary Method

The *m*-ary method is an optimization of the binary method. Bits of the exponent are scanned in groups as against the binary method in which a bit is scanned per iteration. A *t*-bit exponent is divided into *s* words of length *r* bits each. If r doesn't divide *t*, the exponent is padded with at most $(r-1)$ 0s just before the leftmost bits. The actual *t*-bit word is defined in equation (1).

$$f_i = \left(e_{ir+r-1}, e_{ir+r-2}, \ldots, e_{ir}\right) = \sum_{j=0}^{t-1} e_{ir+j}\, 2^j \qquad (1)$$

This method does a precomputation of $g^w \pmod n$ for all $w = 2, 3, \ldots, m-1$ after which the bits of the exponent are scanned *r* bits per iteration from left to right. For each iteration, modular multiplications are performed based on the bits scanned (Koc, 1994). The algorithm for this technique is shown below.

---

m-ary modular exponentiation

INPUT: positive integer *g*, exponent $e = (e_{t-1}, \ldots, e_1, e_0)_2$ and a modulus *n*.

OUTPUT: $A = g^e \bmod n$.

1. Precomputation
   1a. Compute and store $g^w \pmod n$ for all $w = 2, 3, \ldots, m-1$.
   1b. Decompose *e* into *r*-bit words f_i for i from 0 to $s-1$.
2. $A \leftarrow g^{fs-1} \pmod n$.
3. For *i* from $s-2$ down to 0, do;
   3a. $A \leftarrow A^{2^r} \left(\bmod n\right)$
   3b. If $f_i \neq 0$, $A = A \cdot g^{fi} \pmod n$.
4. Return (*A*).

This technique can be optimized to do precomputations in the powers of only the partitioned exponent bit value. This helps to save unnecessary precomputations that are not used. A generalized analysis can be given as follows; for a *t*- bit exponent, with the assumption that $2^r = m$ and t/r is always an integer, this method requires;

- $2^r - 2$ number of precomputations,
- $(t/r - 1)\, r = (t - r)$ number of squarings (step 3a),
- $(t/r - 1)(1 - 2^{-r})$ number of multiplications.
- $2^r - 2 + t - r + (t/r - 1)(1 - 2^{-r})$ average number of multiplications.

Drawbacks – Each iteration step involves costly multiple precision modular multiplications. Also for larger exponent values, it would involve too many iterations.

Sliding Window Method

Siding window method attempts to partition the exponent bits into zero and nonzero words (window) f_i . It then does as many multiplications as there are nonzero words (windows). The win-

dows could be of variable length but its least significant bit must be equal to 1. For an exponent partitioned into *d*-bit words, the longest window length is $d = \max L(f_i)$ for $i = 0$ to $(t-1)$. This method also reduces the number of precomputations by computing only g^w for only odd values of *w*. (Koc, 1994). The algorithm is shown below where *p* is the number of windows.

Sliding Window modular exponentiation

INPUT: positive integer *g*, exponent $e = (e_{t-1}, \ldots, e_1, e_0)_2$ and a modulus *n*.

OUTPUT: $A = g^e \bmod n$.

1. Precomputation
   1a. $g_1 \leftarrow g, g_2 \leftarrow g^2$.
   1b. For *i* from 1 to $(2^{d-1} - 1)$, do; $g_{2i+1} \leftarrow g_{2i-1} \cdot g_2$.
   1c. Decompose *e* into zero and nonzero windows f_i of length $L(f_i)$ for *i* from 0 to $p - 1$.
2. $A = g^{f_{p-1}} \pmod n$
3. For *i* from $p - 2$ down to 0,
   3a. $A = A^{2^{L(fi)}} \pmod n$
   3b. If $f_i \neq 0$, then $A = A \cdot g^{f_i} \pmod n$.
4. Return (*A*)

In partitioning the bit, two techniques are being proposed; constant length nonzero windows and variable length nonzero windows (Koc, 1994).

Montgomery's Method

Montgomery method allows the implementation of modular multiplication without directly dividing by the modulus. Instead, it replaces that division operation of the modulus by a power of 2 which is much faster because of the binary representation used by computers (Koc, 1994). This is known as the Montgomery reduction technique. For a k-bit modulus, $2^{k-1} \le n < 2^k$, r is defined to be 2k. This technique requires that gcd (r, n) = 1. It computes the n-residue of the product of two integers whose n-residues are given. The n-residue of an integer $a < n$ with respect to *r* is given as;

$$\overline{a} = a \cdot r \bmod n.$$

Thus the Montgomery reduction is defined as;

$$\overline{R} = \overline{a} \cdot \overline{b} \cdot r^{-1} \bmod n,$$

where r^{-1} is the multiplicative inverse of *r* modulo *n* and $\overline{R}$ is the *n*-residue of the product

$$R = a \cdot b \bmod n$$

An additional quantity n' is required to describe the Montgomery reduction algorithm. It has the property

$$r \cdot r^{-1} - n \cdot n' = 1 \text{ and } n' = -n^{-1} \bmod r$$

r^{-1} and n' are computed using the binary gcd algorithm. The Montgomery reduction algorithm is stated as follows (Koc, 1994);

Montgomery reduction

INPUT: *n*-residues a and b, integer $n = (n_{k-1},...,n_1,n_0)_2$, $r = 2^k$ with gcd(*r*, *n*) = 1 and n'.

OUTPUT: $u = \bar{a} \cdot \bar{b} \cdot r^{-1} \bmod n$.

1. $t = \bar{a} \cdot \bar{b}$.
2. $m = t \cdot n' \bmod r$.
3. $u = (t + m \cdot n) / \ r$.
4. If $u \geq n$ then return $u - n$ else return u.

This Montgomery reduction can be used to obtain a faster algorithm for computing modular multiplication.

Montgomery exponentiation

Montgomery reduction is very suitable for computations that involve several modular multiplications with a fixed modulus. Based on this idea, the Montgomery reduction algorithm is now incorporated into the square and multiply binary algorithm to give an optimized method of computing modular exponentiations called the Montgomery exponentiation. This method circumvents the costly multiple precision multiplications involved in the binary square and multiply method (Koc, 1995). The algorithm is shown below.

Montgomery exponentiation

INPUT: $n = (n_{k-1},...,n_1,n_0)_2$, $r = 2^k$ with gcd(*r*, *n*) = 1, $n' = -n^{-1} \bmod r$, $e = (e_{t-1},...,e_1,e_0)$, and integer $g, 1 \leq g < n$.

OUTPUT: $A = g^e \bmod n$

1. $\bar{g} \leftarrow g \cdot r \bmod n$
2. $\bar{A} \leftarrow 1 \cdot r \bmod n$
3. For *i* from *t* – 1 down to 0, do;
   3a. $\bar{A} = mont(\bar{A}, \bar{A})$
   3b. if $e_i = 1$, then $\bar{A} = mont(\bar{g}, \bar{A})$
4. $A = mont(\bar{A}, 1)$
5. Return (A)

Proposed Algorithm: Montgomery Exponentiation Using Sliding Windows

The proposed algorithm is an improvement on the classical Montgomery exponentiation technique. The classical Montgomery exponentiation carries out the exponentiation operation based on the bits of the exponent in a bitwise fashion. The algorithm been proposed, combines the Montgomery reduction algorithm with the sliding windows exponentiation algorithm. The expo-

nentiation process is carried out in groups of bits (synchronously), thus reducing the number of iterations and improving on the speed of the exponentiation process. The algorithm is shown below.

Montgomery exponentiation using sliding windows

INPUT: $n = (n_{k-1},...,n_1,n_0)_2$, $r = 2^k$ with gcd(r, n) = 1, $n' = -n^{-1} \bmod r$, $e = (e_{t-1},...,e_1,e_0)$, integer $g, 1 \le g < n$, and integer $d \ge 1$.

OUTPUT: $A = g^e \bmod n$

1. $\overline{g} \leftarrow g \cdot r(\bmod n)$.
2. Precomputation
   a) $\overline{g_1} \leftarrow \overline{g}, \overline{g_2} \leftarrow mont(\overline{g}, \overline{g})$.
   b) For i from 1 to ($2^{d-1} - 1$), do; $\overline{g}_{2i+1} \leftarrow mont(\overline{g}_{2i-1}, \overline{g}_2)$.
   c) Decompose e into zero and nonzero windows f_i of length $L(f_i)$ for i from 0 to $p - 1$.
3. a) $\overline{A} \leftarrow r(\bmod n)$
   b) $\overline{A} = mont(\overline{A}, \overline{g}_{f_{p-1}})$
4. For i from $p - 2$ down to 0,
   a) For $j = L(f_i) - 1$ down to 0, do;
   $$\overline{A} \leftarrow mont(\overline{A}, \overline{A})$$
   b) If $f_i \neq 0$, then $\overline{A} \leftarrow mont(\overline{A} \cdot \overline{g}_{f_i})$.
5. $A = mont(\overline{A}, 1)$.
6. Return (A).

A performance comparison of this algorithm is not given in this paper. This is because work is still in progress with respect to the algorithm.

Montgomery Exponentiation (Analysis)

A generalized analysis can be given as follows; for an n-bit exponent, this method requires, where Single Precision Multiplication (SPM) and Multiple Precision Modular Multiplication (MPMM);

Step 1: One MPMM = $(2n^2 + 2)$ SPMs,

Step 3a: $n - 1$ mont-red calls and,

Step 3b: wt(e) -1 mont-red calls, where wt(e) is the hamming weight (number of 1s in the binary representation) of the exponent.

Step 4: One mont-red call = $(n^2 + n)$ SPMs

For $0 \le$ wt(e) -1 $\le n - 1$, there can be a maximum of $n - 1$ ones and a minimum of 0 ones in the binary representation of e. Assuming $e_{n-1} = 1$, this implies that we can have;

- $(n-1) + (n-1) = 2(n-1)$ maximum number of mont-red calls,
- $(n-1) + 0 = (n-1)$ minimum number of mont-red calls,

½ $[2(n-1) + (n-1)] = {}^{3}\!/_{2}\,(n-1)$ average number of mont-red calls (for step 3a).

Since one mont-red call is made up of $n^2 + n$ SPMs, then the total number of SPM calls would be;

$$\left(2n^2+2\right) + \left(\left({}^{3}\!/_{2}\,(n-1)\right) * \left(n^2+n\right)\right) + \left(n^2+n\right)$$

$$= 1.5n^3 + 3n^2 - 0.5n + 2 \text{ SPMs}$$

Montgomery Exponentiation Using Sliding Windows (Proposed Algorithm)

A generalized analysis can be given as follows; for an *n*-bit exponent, this method requires;

Step 1: One MPMM = $\left(2n^2+2\right)$SPMs,

Step 2a: One mont-red call = $\left(n^2+n\right)$ SPMs,

Step 2b: $\left(2^{d-1}\right)$ mont-red calls = $\left(\left(2^{d-1}\right) * \left(n^2+n\right)\right)$ SPMs,

Step 3b: One mont-red call = $\left(n^2+n\right)$ SPMs,

Step 4a: $\left(n-d\right)$ mont-red calls = $\left(\left(n-d\right) * \left(n^2+n\right)\right)$ SPMs,

Step 4b: C_1 mont-red calls = $\left(C_1 * \left(n^2+n\right)\right)$ SPMs (where C_1 is average number of transitions from state 0 to state 1 after *n* iterations (constant length non-zero window) as modeled using Markov chain (Koç, 1995)).

Step 5: One mont-red call = $\left(n^2+n\right)$ SPMs.

Total number of SPMs =

$$\left(\left(2^{d-1}\right) * \left(n^2+n\right)\right) + \left(n^2+n\right) + \left(\left(n-d\right) * \left(n^2+n\right)\right) + \left(C_1 * \left(n^2+n\right)\right)$$

$$= n^3 + 5n^2 + 2^{d-1}n^2 - dn^2 + C_1 n^2 + 2n + 2^{d-1}n - dn + C_1 n + 2^{d-1} + 2$$

In comparing the various exponentiation techniques analyzed, the coefficients of the highest degree of the various polynomials are considered. This is so because all exponentiation algorithms are polynomial time algorithms of the cubic order.

Experimental Results and Discussion

The algorithms were implemented in MATLAB 6.5 codes, run on Pentium III 500Mhz IBM PC. The Classical and the modified algorithms were run for 100 times each for four different exponent bit sizes. This gives a total of 400 runs for each of the algorithms. The execution time for each run was recorded and used to compute the mean and standard deviation for both algorithms. Contrary to expectation, the Classical Montgomery algorithm was seen to have a shorter execution time than the Montgomery modular exponentiation using sliding windows. Figure 1 shows a graph of exponent bit size against the mean execution time. All graphs were also plotted using MATLAB.

Apart from the execution times of the algorithms, an interesting observation was made which forms the strength of this paper. Despite the fact the classical algorithm was seen to run faster

than the modified one, the modified algorithm was seen to make lesser number of calls to the Montgomery reduction sub-function than the classical algorithm. This observation is shown in Figure 2. This implies that although the Montgomery exponentiation using sliding windows algorithm could be slower than the Classical Montgomery algorithm, it makes use of lesser memory space (space complexity). Further statistical analysis revealed that Montgomery exponentiation with sliding windows algorithm required 10% lesser memory referencing. Lesser number of calls to the Montgomery reduction sub-function implies lesser number of loops but why it does not translate to a reduced execution time is yet to be ascertained. In other word, we can say that the Montgomery exponentiation using sliding windows algorithm is arithmetically optimized

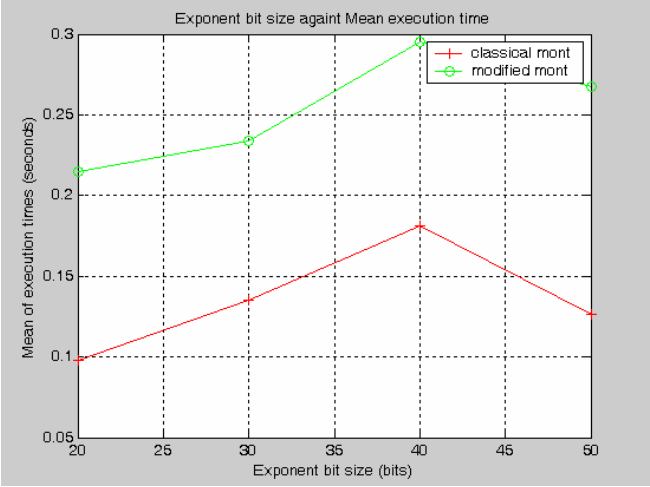


Figure 1: Graph of Exponent bit size against mean execution time

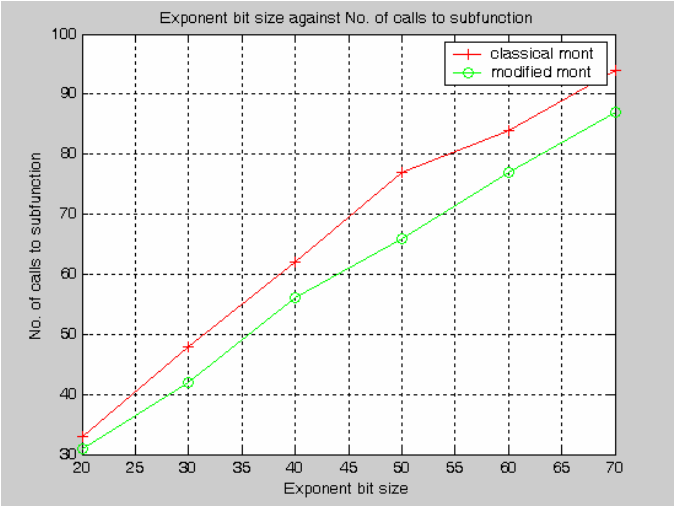


Figure 2: Exponent bit size against No. of calls to sub- function

Conclusion

In large computer complexes, information security becomes a very pertinent issue. Furthermore in applying layers of security, it is also crucial to conserve network resources such as processor time and memory. This project work attempted to propose an optimized algorithm for a core process in cryptographic procedures (exponentiation). As it can be seen from the simulation results, contrary to expectation, the Classical Montgomery modular exponentiation algorithm was

seen to run faster than the proposed Montgomery modular exponentiation using sliding windows. On the other hand, the proposed algorithm was seen to make lesser number of calls to the Montgomery reduction sub-function. This implies it carries out lesser number of memory activities (storage and retrieval) and thus it is optimized for smaller memory applications.

It should be noted that exponentiation is only one of the many algorithms involved in cryptography. In spite of the optimization of this algorithm, it would be imperative to look into the optimization of other algorithms involved. In future, we will find out why the new algorithm did not run faster, despite reduced number of sub-function calls. Thereafter modified the Montgomery exponentiation with sliding windows algorithm for higher speed.

References

Hachez & Quisquater, J. J. (2001). Exponentiation with no final subtractions: Improved results. *Lecture Notes in Computer Science, 1965*, 293-301.

Holcombe C. (2006). What is ecommerce? Retrieved from www.ecommerce-digest.com/tut.html

Joseph, G. & Penzhorn, W. T. (2004). High-speed algorithms for public-key cryptosystems in an e-commerce environment. Proceedings of the *Southern African Telecomunication Networks and Applications Conference SATNAC 2004, 2*, 205-210.

Koç, Ç. K. (1994). High-speed RSA implementation, Version 2.0. *RSA Laboratories*, November 1994.

Koç Ç. K. (1995). Analysis of sliding window techniques for exponentiation. *Computer and Mathematics with Applications, 30*(10), 17-24.

Menezes, P., Van Oorschot, & Vanstone, S. (1997). *Handbook of applied cryptography.* CRC Press.

Oluwatope, A. O., Aderounmu, G. A., Akande, O. O. & Adigun, M. O. (2004). An auction scenario in a stock market using M^2 Agent. Proceedings of *Southern Africa Telecommunication Networks and Applications Conference (SATNAC). 1,* 229 – 234.

Rivest, R. L. (1990). *Cryptology*. MIT Laboratory for Computer Science.

Tang, S. H., Tsui, K. S. & Leong, P. H. W. (2003). Modular exponentiation using parallel multipliers. *Proceedings of the 2003 IEEE International Conference on Field Programmable Technology (FPT)*, Tokyo, 52-59.

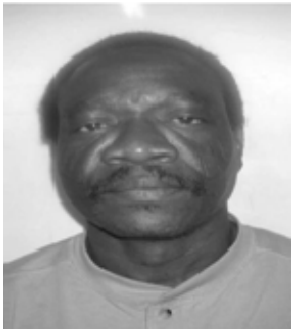
Yesufu, T. K. & Amao, O. R. (2004). A cryptosystem based on a combination of three computationally secure algorithms. *Southern African Telecomunication Networks and Applications Conference SATNAC 2004 Proceedings, 2*, 199-204.

Biographies

A. O. Oluwatope obtained his Bachelors and Masters degree in Computer Engineering and Computer Science from Obafemi Awolowo University, Ile-Ife, in 1995 and 2003, respectively. He is a registered computer engineer with Council for the Regulation of Engineering Practice in Nigeria (COREN), a member of Nigerian Society of Engineers (NSE) and an Associate member of Nigeria Computer Society (NSC). He has co-authored several articles published in Nigeria and abroad. His current research interests include internet congestion control, internet protocol development, algorithm modeling and simulation and grid computing. He has about seven years of teaching and research experience. He is currently a doctoral student and a lecturer in the Department of Computer Science and Engineering of the same university.

B.A. Ojo obtained his Bachelor of Science degree in Computer Engineering from Obafemi Awolowo University, Ile-Ife, in 2005. He is a student member of Nigerian Society of Engineers (NSE). His current research interests are in the areas of computer communications and teletraffic engineering. He is also into protocols design and simulations of wireless communications. Mr. Owojori is currently on National Youth Service.

G.A. Aderounmu holds a research degree M.Sc./PhD in Computer Science from Obafemi Awolowo University, Ile-Ife, Nigeria (1997 and 2001, respectively). He is a member of the Nigerian Society of Engineers (NSE) and is also a registered computer engineer with Council for the Regulation of Engineering Practice in Nigeria (COREN). He is also a Member of Nigerian Computer Society (NCS) and Computer Professional Registration Council of Nigeria (CPN). He has over 13 years of experience in teaching and research. He is an author of many journal articles in Nigeria and abroad. His special interests include engineering education in Nigeria, curriculum development, and computer communication and network. He is a Visiting Research Fellow to the University of Zululand, Republic of South Africa. He is currently a Senior Lecturer, Deputy Director of the Information Technology and Communication Unit (INTECU) and the Acting Head of the Department of Computer Science and Engineering of the same university.



M. O. Adigun holds a research degree PhD in Computer Science from Obafemi Awolowo University, Ile-Ife, Nigeria which he obtained in 1989. Currently, he is a Professor and Head, Department of Computer Science, University of Zululand, Republic of South Africa. He is an author of many journal articles in Nigeria, Republic of South Africa and abroad. His research interests include Software Engineering, Mobile Computing, Modeling and Simulation, and Performance Analysis of Computer System.