Software Simulation of Variable Size Message Encryption Based RSA Crypto-Algorithm Using Ms. C#. NET

1 Hasan Al-Qadeeb, 1 Qasem Abu Al-Haija and 2 Noor A. Jebril

1 King Faisal University, Department of Electrical Engineering, Al-Ahsa 31982, P.O. Box 380
2 Computer Sciences Department, King Faisal University, Ahsa, 31982, P.O. 380, Saudi Arabia

ABSTRACT

RSA Crypto-Algorithm is well-known and widely used public key cryptographic algorithm due to the prominent level of proven security which depends on the difficulty of large integer factorization problem. RSA is heavily based on the modular arithmetic of large integers and the whole RSA includes three parts: key generation, encryption and decryption process. In this paper, we propose a software simulation version via Ms. C# 2013.NET of RSA cryptosystem and its internal modules independently using efficient arithmetic operations and robust number theories. The simulation was tested for 32-bit encryption and decryption keys. We observed that the message that will be encrypted should be limited to the range of the modulus. This implies that the range should begin from zero to the modulus minus one. The simulation showed a comparable delay results for encryption/decryption process especially when it configured with small message sizes 10-200 byte at a time.

KEYWORDS

Public Key Cryptography; RSA Algorithm; Visual Studio.NET, C#.NET 2013.

1. INTRODUCTION

Recently, the enormous neoteric technological revolution in telecommunication and data sharing such as cloud computing and Internet of things (IoT) has raised the demand for essential infrastructure to develop secure communication over insecure channels for different applications. Wide range of solutions have been proposed and investigated for the last couple of decades to constructing and analyzing algorithms of secret data sharing/storing by employing different mathematical schemes along with computer engineering techniques. The art of providing such solutions is called cryptography. Cryptography [5] provides different algorithms to prevent third parties (or the public) from reading/accessing the private messages communicated between two trusted parties. Cryptography provides wide range of services and applications [3] such as: Identification, Key establishment, Secret sharing, Electronic cash, Games [4], Certificates and Authentication, Digital Signature, Quantum Key Distribution and many others.

Classical cryptosystems are antique methods used to secure data in the medieval period. Unluckily, such kind of cryptography is too poor and simple to be broken by attackers, especially using computers. In contrast, it is such a terrific way to understand the main idea of cryptosystems. Therefore, more sophisticated methods were designed to assure various levels of security, these methods are categorized as: Symmetric Key Cryptography (SKC) and Public Key Cryptography (PKC). SKC algorithms use only one private key shared between the sender and the receiver while PKC algorithms use two keys, one is kept private for the decryption operation, and the other key is made public for all users. Generally, PKC algorithms are very powerful and secure, but they need high computational power compared with symmetric key algorithms [6]. RSA Crypto Algorithm is well-known example of PKC.

PKC is heavily based on digital arithmetic and number theory algorithms. The digital arithmetic [7] operations such additions, multiplications, divisions and modular operations form the core part of any processor or coprocessor design. On the other hand, the number theory [5] was largely
separated from other fields of Mathematics as it is
topically related to elementary arithmetic, except
that the concern now is with more advanced and
subtle aspects of the subject. It has rapidly grown
in recent years in practical importance through its
use in areas such as coding theory, cryptography
and statistical mechanics. Such advanced
algorithms comprise of modular arithmetic,
modular exponentiation, prime number theory,
Chinese Reminder Theorem, and Fermat's Little
Theorem and Euler's Theorem. It has attracted
many of the most outstanding mathematicians in
history including Euclid, Diophantus, Fermat,
Legendre, Euler, Gauss, Dedekind, Jacobi,
Eisenstein, Sieve of Eratosthenes and Hilbert all
made immense contribution to its development.
Many of these number theory algorithms and
digital arithmetic operations are contribution as
building blocks of RSA cryptosystem.
In the last years, many hardware/software
solutions tried to address the efficient design
public key algorithms such as RSA. The most
commonly used solutions were the FPGA design
and synthesize such as [8] and the microprogramming of Microcontroller chips such
as [9] as well as the high-level simulations. In this
paper, we propose a high-level simulation of 32-bit RSA Crypto-algorithm using Microsoft visual
studio 2012. Net suite - C# language. In addition
to RSA encryption/decryption processes, the
simulator encompasses several tools many
arithmetic and number theory algorithms that are
commonly used in designing several PKC
algorithms.
The remaining of this paper is organized as
follows: Section 2 re-visited the theoretical topics
of RSA Cryptosystem and Section 3 provides the
simulation environment of the proposed system.
Section 4, system analysis and evaluation, contains
simulation outcomes and experimental results
together with the discussions which include
performance measures the proposed simulation for
different message sizes. Finally, Section 6
concludes the paper.

2. RSA CRYPTO-SYSTEM-REVISITED

RSA which stands for Rivest, Shamir, and
Adleman, the three scientists who defined the
algorithm, is public key crypto algorithm that its
strength based on the difficulty of integer
factorization. Being a PKC algorithm means that
everyone can encrypt a message, but only
authorized users can decipher the message.

2.1 RSA Algorithm

RSA algorithm works as follows: Firstly, choose
two large distinct prime numbers (p) and (q). The
product of these numbers is assumed to be called
the modulus (n) which is the component of the
public key, but the number must be large enough
such that the numbers (p) and (q) cannot be
feasibly extracted to have a robust system. Then
generate the encryption key (e) which must be co-
prime to the Euler number \( \varphi(n) = (p-1) \times (q-1) \). Then create the decryption key (d) such
that \((e\times d)\mod \varphi(n) = 1\). And, finally both of
public and private are known. The complete steps
of RSA Algorithm is given in figure 1.

\[
\begin{align*}
\varphi(n) &= (p-1) \times (q-1) \\
\text{Where } \varphi(n) &< n
\end{align*}
\]

After that, pick up a number for the encryption key (e) which also called
the public key and the decryption key (d) which also called the private key,
where \(e\) is computed by generating random positive integers which must be co-
prime and \(d\) is the inverse of \(e\mod m\) as below:
\[
1 < e < \varphi(n);
\]
\[
\text{Where } \gcd(\varphi(n), e) = 1
\]
\[
d = e^{-1} \mod \varphi(n);
\]
\[
\text{Where } (e \times d) \mod \varphi(n) = 1
\]
\[
E = D^e \mod n
\]
\[
D = E^d \mod n
\]

2.2 Mathematical Proof of RSA Algorithm

RSA computations can be mathematically proofed
by forward substitution of the encryption process
of plaintext message M to get the ciphered
message C and then by backward substitution of
Ciphertext C to get back the plaintext message M
as shown in the figure 2 below.
2.3 RSA Challenge

The idea of RSA challenges [5] has been set before with lower-strength ciphers to encourage researcher with computational number theory and practical difficulty of factoring large integers. The RSA challenges were held by the RSA laboratories. They were concerned about factoring enormous number keys. That is, the keys are the product of two large prime numbers. However, the historical challenges were ranging between 576 bits to 2048 bits as eight RSA challenge numbers. They awarded a cash prize to the first person to factor each challenge number. Prizes ranged from $10,000 for the 576-bit challenge to $200,000 for 2048 bits. Previous RSA challenges have revealed that the US-government backed DES (Data Encryption Standard) was vulnerable to a brute force attack that yielded the result of a 56-bit key in a little over 22 hours [1].

2.4 RSA Factoring

Factoring a number means defining that number as a product of prime numbers. Examples of prime numbers are 2, 3, 5, 7, 11, and 13. Those numbers do not divide by any smaller number except 1. On the other hand, a composite number can be written as the product of smaller primes. Those are called prime factors, also. For instance, the number 665 is a product of the primes 5, 7, and 19. A number is said to be factored when all its prime factors are identified. The difficulty of factorization increases, as the length of the number increases. Factoring 100-bit number is not difficult with advanced algorithms. The first person to submit a correct factorization for any of the challenge numbers was eligible for a cash prize. Given the amount of computation required for such a factorization, the prizes were mainly symbolic. They served as a small incentive for public demonstrations of factoring on a large scale. In 2005, the largest number of RSA to be factored was 640 bits. RSA-2048 bit stands as the biggest number of RSA to be factored. The GNFS [10] is one of the best algorithms for factoring very large numbers. GNFS consists of a sieving phase that searches for a fixed set of prime numbers for candidates that have a particular algebraic relationship, modulo the number to be factored. This is followed by a matrix solving phase that creates a large matrix from the candidate values, and then solve it to determine the factors.

2.5 Possible Attacks Of RSA

Reasonably, there is no system perfect, but there are systems hard to be attacked. RSA is frequently used in applications such as e-mail, e-banking, remote login, etc., where security of digital data is a primary concern. Over years, numerous attacks on RSA illustrating RSA’s present and potential vulnerability have brought our attention to the security issues of RSA cryptosystem. Although twenty years of research have led to several fascinating attacks, none of them is devastating. They mostly illustrate the dangers of improper use of RSA. Indeed, securely implementing RSA is a nontrivial task. However, these possible attacks can be divided into four main disciplines that can be depicted in the following figure (figure 3) which summarizes the possible attacks on RSA as reported in the literature [2].
3. SIMULATION ENVIRONMENT

The completed design of RSA is comprised of six levels: random two prime numbers, parallel multiplication of the prime numbers and their decremented values, encryption key, decryption key, encryption and decryption levels. In fact, the implementation of RSA Cryptosystem is heavily based on modular arithmetic and exponentiation involving large prime numbers. This work proposes a software simulation for RSA Cryptosystem using MS. C#.Net 2013. The simulation and testing was conducted using high-performance computing platform which has the specifications shown in table 1.

Primarily, modular exponentiation with large modulus is considered as the core operation of RSA computation which is typically obtained by carrying out repeated modular multiplications. Unfortunately, this will consume extensively long-time delay. Consequently, the speed of the cryptosystem will totally depend on the speed of the modular multiplication as well as the number of modular multiplication components. Therefore, to increase the speed of the modular multiplication, Interleaved Algorithm [11] was particularly chosen as an efficient solution. Finally, this system can be utilized for several security claims such as network routers and security gateways using certain protocols.

3.1 Design Modules

The system design to be accomplished, it included multistage phases of mathematical computations using several arithmetic and number theory algorithms are taken into consideration. As seen from figure 4 below, the RSA Crypt-system passes through the following stages:

1. Stage #1: Plaintext message (M) Preparation.
   - The original message to be encrypted by the RSA simulator can be easily entered by the Keyboard and then the text is converted to string of numbers using Extended ASCII Coding tables [12].

2. Stage #2: Generating two large Primes (P, Q).
   - This process started by generate two large numbers using Pseudo Random Number Generation (PRNG) using Linear Feedback Shift Registers (LFSRs) as in [13] and then followed by primality testing module based on Rabin Miller primality testing algorithm [14].
3. Stage # 3: Arithmetic Multiplication. This includes Modulus Generation by Multiplication of two large primes (p, q) and Euler function Generation by Multiplication of (p-1 and q-1). We have used the sequential multiplication with redundant carry save audition which is considered as high-speed multiplier.

4. Stage # 4: Generation Public Encryption Key (e). This is accomplished by choosing random (e) using RNG between 2 and \( \varphi(n) \) with the condition of GCD \( (e, \varphi(n)) \) to assure that e and \( \varphi(n) \) are relatively prime. We have used Euclidean Algorithm [15] to compute the greatest common divisor (GCD) between two non-negative integers.

5. Stage # 5: Generation of Private Decryption Key (d). This is accomplished by using modular inverse operation and we have chosen Extended Euclidean Algorithm which is well-known extension of the Euclidean algorithm [14] to find the modular multiplicative inverse of two co-prime numbers.

6. Stage # 6: Encryption/ Decryption Stage. This is the most expensive operation of RSA algorithm and can be done by applying modular exponentiation. Modular exponentiation is a great technique to simplify modular arithmetic calculation. Its function is to calculate the modulus for very high-power exponents. In fact, we selected Right to Left Interleaved Modular Exponentiation Algorithm because it is an efficient method to be implemented practically, especially when the exponent becomes very large [11]. Finally, the numbers are converted back to text using the same ASCII tables for number-to-text.

![Figure 4. Complete process of RSA dataflow stages on both communication sides](image)

### 3.2 RSA Arithmetic

As seen from previous sub-section, RSA Arithmetic involves all the theorems and algorithms used in designing the RSA components. These include modular exponentiation, modular multiplication, modular inverse, multiplier, adders, primality testing, greatest common divider and random number generator.

Starting with the modular exponentiation, the Right to Left Modular Exponentiation Algorithm principle is to take the exponent in binary radix and starts from the least significant bit. Firstly, we initiate a register C with the value one. After that, if the corresponding bit of the exponent is ‘1’ then we put in that register the modular multiplication of the base by the register C. However, if the corresponding bit of the exponent is ‘0’ then the register C remains the same. This will pass through a loop with the length of the exponent iterations. As it is clear from its name that it takes the bits of the exponent from the right to the left.
iteration, there should be an initiated register S and calculate the modular multiplication of S2. Finally, the output will be ready at the end of the loop at register C.

Secondly, we used the Interleaved Algorithm as the modular multiplication. At the beginning, we initiate three registers P, I and J where P is set at zero. This will pass through a loop with the length of the first operand iterations. Then we put in register P the multiplication of two by P. After that, if the corresponding bit of the first operand is ‘1’ then we put in register I the second operand. However, if the corresponding bit of the exponent is ‘0’ then the register I remains the same. Then we put in register P the addition I and P. After that, we do two comparisons if the register P is greater than the modulus then we put in register P the subtraction of P minus the modulus. Finally, the output will be ready at the end of the loop at register P.

Third, we used the Extended Euclidean Algorithm as the modular inverse. At the beginning, we initiate three registers Q, T, B0, X1 and X0. If the modulus is ‘1’ then the output is ‘1’. We put the modulus in the register B0, zero in the register X0 and one in X1. After that, we do a loop until A becomes greater 1. Inside the loop, we do the following: we put in Q the division of a by B. The modular of A and B in the register T. Then we put T in A, X0 in T, X1 – Q * X0 in X0 and T in X1. After finishing from the loop, we do a comparison if the register X1 is less than 0 then we put B0 in X1. Finally, the output will be register X1.

We used the Carry Save Adder based Carry Look Ahead Adder. It is a parallel operation of full adders. The number of full adders depends on the operand’s length. So, each full adder can handle one-bit addition. After that, the results of the full adders should pass through carry look ahead adder to find the result. We used the sequential multiplier where it is based on the CSA. Here we used a shift register for the second operand and a multiple generator. The multiple generator is simply the ‘ANDing’ between the first operand the corresponding bit of the second operand. Each result of the multiple generator will be added in the CSA with the previous carry and the previous vector. We initiate the previous carry and the previous vector as zeroes. Finally, the output will be the addition of the previous carry and the previous vector by the carry look-ahead adder.

We used Euclidean Algorithm to find the greatest common divisor. Its principle can be done only by performing the modular multiplication. We simply do a loop until the second input becomes greater zero. Inside the loop, we do the following: the modular multiplication of the two inputs. Then we put exchange the input registers. After that, we put the reminder in the second input register. The output will be ready after finishing the loop in the first input register. Finally, we used the linear Feedback Shift Register as the random number generator. It is fast and so simply. Its operation lies on the ‘XOR’ gate. It takes the last two bits and pass them through the gate and the result is inserted as input to the generator. In addition, the linear Feedback Shift Register is initiated at one.

### 3.3 Overview Of MS-C# 2012.NET

C# [16] is a modern, general-purpose, object-oriented programming (OOP) language developed by Microsoft and approved by ECMA and ISO. OOP is an approach to software development in which the structure of the software is based on objects interacting with each other to accomplish a task. This interaction takes the form of messages passing back and forth between the objects. In response to a message, an object can perform an action. An object is a structure for incorporating data and the procedures for working with that data. For example, if you were interested in tracking data associated with products in inventory, you would create a product object that is responsible for maintaining and working with the data pertaining to the products.

C# is widely used professional language for several reasons such that: Modern, general-purpose programming language, Object oriented, Component oriented, Easy to learn, Structured language, it produces efficient programs, it can be compiled on a variety of computer platforms and it’s a Part of .Net Framework. C# application types include Windows Console applications, Windows Forms applications, ASP.NET Web applications, ASP.NET Web Service applications, Smart Device applications, ActiveX applications, and setup and deployment applications.
4. SYSTEM ANALYSIS AND EVALUATION

In this research, we provide a software implementation for the RSA architecture given in figure 4. There are many approaches and programming languages that can be used for this purpose. In this work, we have used the C#.NET programming language (part of Visual Studio 2013 suite) due to its proven efficiency and ease of interaction between objects [16]. The main window of our simulator is shown in figure 5 (a).

![RSA Software Main Menu](image1)
![Components List](image2)
![Text to Number Converter](image3)
![Modular Multiplication](image4)
![Primality Testing](image5)
![Sequential Multiplier](image6)
![Modular Exponentiation](image7)
![Carry LockAhead Adder](image8)

**Figure 5.** (a) RSA Software Main Menu   (b) All Program Components

![Sample Simulation Runs](image9)

**Figure 6.** Sample Simulation Runs
The simulation program of 32-bit RSA has been examined for different text message sizes as shown in the table 2 along with figure 7. The simulation runs showed that encryption/decryption of small message sizes (10-200 bytes) has recorded a lowest delay time with average delay time 10 – 170 ms. Comparing these results with our similar design using Arduino microcontroller in [9] will give better delay times for small size messages as Arduino based design relate between the message size and the required time for encryption/decryption linearly due to the block size of the encryption that used in this version. However, for larger block sizes, FPGA design in [8] is better option for hardware based RSA coprocessor.

<table>
<thead>
<tr>
<th>Size (Byte)</th>
<th>12</th>
<th>27</th>
<th>51</th>
<th>81</th>
<th>102</th>
<th>150</th>
<th>200</th>
<th>303</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time (ms)</td>
<td>1</td>
<td>2</td>
<td>8</td>
<td>20</td>
<td>33</td>
<td>83</td>
<td>170</td>
<td>490</td>
</tr>
</tbody>
</table>

**Table 2: Time characteristic of encryption using C#.NET for 32-bit RSA**

5. CONCLUSIONS AND FUTURE WORKS

In conclusion, a software program of RSA cryptographic system was successfully simulated via Ms. C# 2013.NET. Also, many other design set algorithms are simulated separately as independent components such as primality tester and modular exponentiation. The simulation was tested for 32-bit encryption and decryption keys. We observed that the message that will be encrypted should be limited to the range of the modulus. This implies that the range should begin from zero to the modulus minus one. Moreover, the attacks pointed out previously provided a clear idea on how to support and strengthen the structure of the RSA functions. In future, this design can be stretched out to higher encryption and decryption bit-size by revising and modifying the utilized algorithms. Also, ciphering audio and video data public key cryptography can be performed along with Steganography. Moreover, several public and symmetric key cryptographic algorithms could be simulated and verified using the design sets mentioned in this project; such as: El-Gamal Cryptosystem, Diffie-Helman Key exchange system, Data Encryption Standard (DES), Advanced Encryption Standards (AES), and many others.

REFERENCES


