Jurnal Teknologi

Full Paper

COMPARATIVE OF RIVEST-SHAMIR-ADLEMAN CRYPTOSYSTEM AND ITS FOUR VARIANTS USING RUNNING TIME AND MEMORY CONSUMPTION ANALYSIS

Arif Mandangan[a], Muhammad Asyraf Asbullah[b]*, Syed Farid Syed Adnan[c], Mohammad Andri Budiman[d]

[a]Mathematics, Real-Time Graphics and Visualization Laboratory, Faculty of Sciences and Natural Resources, Universiti Malaysia Sabah, 88400 Kota Kinabalu, Sabah, Malaysia
[b]Institute for Mathematical Research, Universiti Putra Malaysia, 43400 UPM Serdang, Malaysia
[c]School of Electrical Engineering, College of Engineering, Universiti Teknologi Mara, 40450 Shah Alam, Malaysia
[d]Faculty of Computer Science and Information Technology, Universitas Sumatera Utara, Jl. Universitas No. 9-A, Kampus USU, Medan 20155, Indonesia

Article history
Received
13 July 2023
Received in revised form
7 January 2024
Accepted
7 January 2024
Published Online
23 June 2024

*Corresponding author
ma_asyraf@upm.edu.my

Graphical abstract

Abstract

The Rivest-Shamir-Adleman (RSA) algorithm, known for its slow single-precision multiplication (spm) and overall running time, is not commonly employed to encrypt user data directly. As a result, several researchers have developed various RSA-based cryptosystems to enhance the algorithm's performance while maintaining security. This paper presents a comparative analysis of different variants of the RSA cryptosystem, focusing on five specific cryptosystems: RSA, Somsuk-RSA, Modified-RSA (MRSA), Easy Simple Factoring-RSA (ESF-RSA), and Phony-RSA. The methodology involves evaluating the theoretical running time and memory usage through single-precision multiplication (spm) measurements, while the actual running time is estimated using Maple programming. The research has two primary objectives. Firstly, they examined each algorithm of the RSA variants and analysed them according to the proposed methodology. Secondly, to determine which cryptosystem consumes the most time and memory for key generation, encryption, and decryption. The results indicate that ESF-RSA and RSA are the fastest in terms of key generation, ESF-RSA is the quickest for encryption, and Phony-RSA excels in decryption speed. Additionally, ESF-RSA demonstrates the lowest memory usage, whereas MRSA requires the highest memory allocation for all processes.

Keywords: Cryptosystem, single precision, encryption, running time, memory consumption

Abstrak

Algoritma Rivest-Shamir-Adleman (RSA, dikenali dengan pendaraban berkepersisan-tunggal serta masa larian yang perlahan, lazimnya kurang digunakan secara terus untuk menyulitkan data pengguna. Natijahnya, para penyelidik telah membangunkan pelbagai sistemkripto berasaskan RSA untuk meningkatkan prestasi algoritma tersebut disamping mengekalkan tahap keselamatannya. Kertas kajian ini mencadangkan suatu analisis perbandingan

terhadap pelbagai varian sistemkripto RSA, dengan penumpuan khusus terhadap lima sistemkripto iaitu RSA, Somsuk-RSA, RSA-Terubahsuai (MRSA), RSA-Pemfaktoran Mudah Ringkas (ESF-RSA), serta Phony-RSA. Metodologi yang digunakan adalah merangkumi penilaian terhadap masa larian dan penggunaan memori secara teori melalui ukuran pendaraban berkepersisan-tunggal, disamping menentukan masa larian sebenar dengan menggunakan pengaturcaraan Maple. Kajian ini mempunyai dua objektif utama. Pertama, setiap algoritma varian RSA tersebut dinilai serta dianalisa mengikut metodologi yang dicadangkan. Kedua, untuk menentukan sistemkripto manakah yang menggunakan masa serta memori yang paling tinggi dalam proses penjanaan kekunci, penyulitan dan penyahsulitan. Keputusan yang diperolehi menunjukkan bahawa ESF-RSA dan RSA adalah sistemkripto yang terlaju bagi proses penjanaan kekunci, manakala ESF-RSA adalah varian yang terlaju bagi proses penyulitan dan Phony-RSA pula cemerlang dari segi kelajuan dalam proses penyahsulitan. Selain itu juga, ESF-RSA menunjukkan penggunaan memori terendah, manakala MRSA pula memerlukan peruntukan memori tertinggi untuk kesemua proses tersebut.

Kata kunci: Sistemkripto, berkepersisan tunggal, penyulitan, masa larian, penggunaan memori

1.0 INTRODUCTION

Cryptography word comes from a combination of two ancient Greek words, which are "Kryptos" and "graphene", with the meaning "hidden, secret" and "to write", respectively. The Concise Oxford English Dictionary defines cryptography as "the art of writing or solving codes" [1]. In general, cryptography is about constructing and analysing rules to secure communication between two parties with the presence of adversaries [2]. Cryptography and cryptanalysis are also linked to cryptology. Before the early 20th century, electronic devices became a popular medium to transport information. Since then, cryptography algorithms have been heavily based on mathematical theory and computer science practice [3].

Before the modern age, cryptography was effectively compatible with encryption and decryption. Encryption is converting ordinary information called plaintext into an unintelligible form called ciphertext. Decryption is the reversed process of encryption that converts ciphertext back to plaintext [4]. Both sender (encryptor) and receiver (decryptor) share a "key" to perform their operation successfully. The sender used the same key to encrypt the plaintext and the receiver to decrypt the ciphertext to plaintext. The key must be kept secret by both as no matter how obscure the algorithm for encryption and decryption is, it is troublesome if the key is not safe [5].

The widespread use of computers and communications networks in the 1960s prompted a need from the commercial sector for tools to secure digital information and provide security services [6]. Before the mid-1970s, all cypher systems implemented symmetric key algorithms. The sender and the recipient used the same cryptographic key with the underlying algorithm and had to keep it secret. The key in every such system has to be shared in a secure method beforehand. However, this technique could be more practical and quickly becomes unmanageable when the number of parties grows when secure channels are unavailable or when keys are often changed (as is common cryptographic practice) [7].

[8] proposed public-key cryptography (also called asymmetric cryptography) in the mid-1970s. In this system, the sender and receiver are the two participants in the asymmetric encryption workflow; each has its own public and private keys. The sender acquires the public key of the recipient. The sender then encrypts the plaintext or regular, readable text using the receiver's public key, resulting in the ciphertext. The ciphertext is then sent to the receiver, which decrypts the ciphertext with their private key and returns it to plaintext [9].

[10] published a public-key cryptosystem named RSA (Rivest–Shamir–Adleman) 1977. An RSA user creates and publishes a public key based on two large prime numbers and an additional value. The prime numbers are kept secret. Anyone can encrypt messages via the public key, but they can only be decoded by someone who knows the prime numbers.

The RSA algorithm has three steps: key generation, encryption and decryption. The RSA cryptosystem is defined as follows.

Algorithm 1.1 RSA Key Generation Algorithm

1: Choose distinct random prime p, q such that $2^k < p, q < 2^{k+1}$.
2: Compute $N = pq$ and $\phi(N) = (p-1)(q-1)$.
3: Choose e such that $3 \le e < \phi(N)$ and $\gcd(e, \phi(N)) = 1$.
4: Compute d such that $ed \equiv 1 \pmod{\phi(N)}$.
5: Return the public key (N, e) and the private key (N, d).

Algorithm 1.2 RSA Encryption Algorithm

1: Choose integer $0 < m < N$ such that $\gcd(m, N) = 1$.
2: Compute $c \equiv m^e \pmod{N}$.
3: Return the ciphertext c.

Algorithm 1.3 RSA Decryption Algorithm

1: Compute $m \equiv c^d \pmod{N}$.
2: Return the plaintext m.

RSA is a relatively slow algorithm in single-precision multiplication (spm) and actual running time. Because of this, it is not commonly used to encrypt user data directly [11]. Therefore, numerous variants of RSA-based cryptosystems are created by numerous researchers to improve the algorithm without sacrificing security. Thus, we want to know which algorithm runs faster in key generation, encryption and decryption among RSA cryptosystem and selected RSA variants cryptosystem. In addition, we will analyse memory consumption for RSA and selected variants in key generation, encryption and decryption processes.

After RSA was proposed, various researchers attempted to enhance RSA by improving its algorithm. Most of the time, researchers improve the RSA algorithm by applying various mathematical methods to the algorithm without changing the basic structure of RSA. Therefore, this section will review the RSA variants algorithms proposed before to improve the RSA cryptosystem.

[12] presents the new equation for RSA's decryption process to speed up the computation time. Note that the private key is usually created with a higher value than the public key to avoid intrusion from unintended parties. However, with a large private key, the decryption procedure takes longer. When a high private key is used, the new exponent becomes a tiny number in the proposed method. The experimental findings demonstrate that the proposed approach can quickly complete the RSA decryption process when the private key is large, especially near the Euler value [13]. However, care must be taken for parameter selection to avoid insecurity due to some efficient brute force attack.

[14] proposed a Modified RSA (MRSA) scheme that minimises the RSA system's key flaws. The main concern in most situations is that it is readily breakable due to the easy calculation of keys depending on N. Original RSA N is easily traceable since it is the only product of two prime integers. Therefore, MRSA used four large primes rather than two. The key generation of MRSA depends on a large factor value N; thus, it needs a higher key generation time. Encryption and decryption also take longer than the RSA method when involving four primes [15].

A new easy, and simple ESF-RSA public key cryptosystem was based on the factoring problem proposed by [16]. The new proposed cryptosystem requires no inverse modular operation during secret key generation. Moreover, if the public key and modulus in ESF-RSA are fixed, the size and value of the secret key are reduced. As a result, ESF-RSA encryption and decryption operations are more efficient than RSA [9].

[17] suggest an improved RSA method to overcome the limitation of an integer factorisation attack by increasing the complexities of the factorisation process by using a phoney/fake public key exponent f instead of e and a phoney modulus X instead of N. This method will provide better security than RSA by decreasing encryption and decryption time. Furthermore, [18] analyzes potential attacks that could be launched against the suggested system and deliberate on its efficacy.

There are two main objectives of this research. Firstly, to compare the running time and memory consumption of key generation, encryption and decryption for the RSA cryptosystem and four selected RSA variants. Secondly, to simulate all the cryptosystems using Maple programming and capture the actual running time in seconds. This research is limited to the original RSA cryptosystem, Somsuk-RSA cryptosystem, Modified-RSA (MRSA) cryptosystem, Easy Simple Factoring-RSA (ESF-RSA) cryptosystem and Phony-RSA cryptosystem.

This paper is organized in the following outline. In the next section, we provide the methodology of this study from both perspectives, single-precision multiplication and memory consumption. This comparative study's theoretical and experimental results are presented in Section 3, followed by a discussion in Section 4. Finally, Section 5 concludes the paper.

2.0 METHODOLOGY

The comparative analysis will be conducted both theoretically and experimentally in this study. The selected RSA variants cryptosystem are the RSA cryptosystem, Somsuk-RSA cryptosystem, Modified-RSA (MRSA) cryptosystem, Easy Simple Factoring (ESF) cryptosystem, and the Phony-RSA cryptosystem.

2.1 Single-precision Multiplication

We use the single-precision multiplication (spm) measurement to determine the running time in this study. This method compared the complexity of each RSA-based cryptosystem's key generation, encryption and decryption running time. Moreover, all the algorithms of the cryptosystem will be run in Maple software to find out which one of the schemes is faster.

Tables 2.1 and 2.2 will show how spm is applied in basic and modular arithmetic, respectively, as applied by [6] and [19].

Table 2.1 Single-precision Multiplication for Basic Arithmetic

Operation [6, 19]	Single-precision Multiplication (spm)
Addition /Subtraction	Add or subtract for two α bit integers is α spm.
Multiplication	Multiply α bit and β bit, the operation required $(\alpha+1)(\beta+1)$ spm. Hence, multiplication for two α bit integers is $\alpha^2+2\alpha+1$ spm
Squaring	Multiply α bit integers by itself requires $\frac{\alpha^2+\alpha}{2}$ spm.
Division	Division of α bit by β bit integers requires $(\alpha-\beta)(\beta+3)$ spm.

Table 2.2 Single-precision Multiplication for Modular Arithmetic

Operation [6, 19]	Single-precision Multiplication (spm)
Modular Reduction	Suppose a is an integer of α bit and n is an integer of β bit integer such that $a>n$. Let $a \pmod n$ perform a modular reduction, then this operation requires a $(\alpha+1)(\beta)$ spm.
Modular Multiplication	Suppose a,b,n are integers of α bit integers. Let $ab \pmod n$ performs a modular multiplication, and then this operation requires $4\alpha^2+4\alpha$ spm.
Modular Inversion	A modular inversion $a^{-1} \pmod n$ is like 30 times slower than a modular multiplication.
Modular Exponentiation	Suppose a,k,n are integers of α bit, γ bit and β bit, respectively. Let $a^k \pmod n$ be modular exponentiation. Then this operation requires $3(\alpha+1)(\gamma+1)(\beta)$ spm.

2.2 Memory Consumption

[20] introduced a method to calculate memory consumption using system parameters and accumulators, which also appeared in [19]. The descriptions below will be assessed to analyze the memory usage of system parameters and accumulators in an RSA-based cryptosystem.

System Parameter

A system parameter refers to a predetermined value that may remain unchanged before any computational operations. This parameter is permanently stored in memory, typically embedded in the hardware.

Accumulator

An accumulator represents a constant or variable value associated with ongoing computational tasks, and its resulting value is temporarily stored in memory. After completing all necessary computations, the accumulator's data is removed. Essentially, the memory allocation for an accumulator is dynamically created and cleared.

3.0 RESULTS

The running time evaluation is used for each cryptosystem's key generation, encryption, and decryption. Each subsection will display each process running time evaluation results in spm term. Aside from that, additional results on the memory cost evaluation of each process will be reported.

3.1 Running Time and Memory Cost Evaluation for RSA

3.1.1 Running Time Estimation for RSA Key Generation, Encryption and Decryption

Key Generation

We recall that the algorithm for the key generation process for RSA is as in Algorithm 1.1. For Step 2, N is the product of two $2k$-bit; thus, its running time is $4k^2+4k+1$ spm. Furthermore, $\phi(N)$ is the product of two $2k$-bit with running time $4k^2+4k+1$ spm. Step 4 involves a modular inversion of two $4k$-bit integers; the running time is $1920k^2+480k$ spm. Hence, the total running time for RSA key generation is $1928k^2+488k+2$ spm.

Encryption

We recall that the algorithm for the key generation process for RSA is as in Algorithm 1.2. From Step 1, we can see that the maximum size of m is $4k$-bit. Next, in Step 2, the modular exponentiation of a $4k$-bit integer with a $4k$-bit integer is needed. The running time in Step 2 is $192k^3+96k^2+12k$ spm, and the total running time for RSA encryption.

Decryption

We recall that the algorithm for the key generation process for RSA is as in Algorithm 1.3. The maximum size for c is $4k$-bit. Step 1, to compute m requires modular exponentiation of a $4k$-bit integer with a $4k$-bit integer. The running time for Step 1 is $192k^3+96k^2+12k$ spm. Hence, the total running time for RSA decryption is also $192k^3+96k^2+12k$ spm.

3.1.2 Memory Cost for RSA Key Generation, Encryption and Decryption

Table 3.1, Table 3.2, and Table 3.3 show the memory consumption of the system parameters and the accumulators for RSA key generation, encryption, and decryption, respectively.

Table 3.1 Memory Cost for RSA Key Generation

Category	Register Name	No. Of Register	Bits
System parameters	N, e, d	$3\times 4k$	$12k$
Accumulators	p, q	$2\times 2k$	$4k$
	$\phi(N)$	$1\times 4k$	$4k$
		Total	$20k$

Table 3.2 Memory Cost for RSA Encryption

Category	Register Name	No. Of Register	Bits
System parameters	e, N	$2\times 4k$	$8k$
Accumulators	c, m	$2\times 4k$	$8k$
		Total	$16k$

Table 3.3 Memory Cost for RSA Decryption

Category	Register Name	No. Of Register	Bits
System parameters	d, N	$2\times 4k$	$8k$
Accumulators	c, m	$2\times 4k$	$8k$
		Total	$16k$

3.2 Running Time and Memory Cost Evaluation for Somsuk-RSA

Suppose we consider the Somsuk-RSA key generation, encryption, and decryption as follows.

Algorithm 3.1 Somsuk-RSA Key Generation Algorithm

1: Choose distinct random prime p, q such that $2^k < p, q < 2^{k+1}$.
2: Compute $N = pq$ and $\phi(N) = (p-1)(q-1)$.
3: Choose d such that $2^{k/2} < d < 2^{k/2+1}$.
4: Compute x such that $x = \phi(N) - d$.
5: Compute e such that $ed \equiv 1 \pmod{\phi(N)}$.
6: Return the public key (N, e) and the private key (N, x).

Algorithm 3.2 Somsuk-RSA Encryption Algorithm

1: Choose integer $0 < m < N$ such that $\gcd(m, N) = 1$.
2: Compute $c \equiv m^e \pmod{N}$.
3: Return the ciphertext c.

Algorithm 3.3 Somsuk-RSA Decryption Algorithm

1: Compute $m \equiv (c^{-1})^x \pmod{N}$.
2: Return the plaintext m.

3.2.1 Running Time Estimation for Somsuk-RSA Key Generation, Encryption and Decryption

Key Generation

Let's refer to the Somsuk-RSA key generation process in Algorithm 3.1. For Step 2, N is the product of two $2k$-bit; thus, its running time is $4k^2 + 4k + 1$ spm. Furthermore, $\phi(N)$ is the product of two $2k$-bit with running time $4k^2 + 4k + 1$ spm. Step 4 involves subtracting a $4k$-bit integer with a $2k$-bit integer, so the running time is $2k$ spm. Step 5 involves a modular inversion of two $4k$-bit integers to obtain e; then, the running time is $1920k^2 + 480k$ spm. Hence, the total running time for Somsuk-RSA key generation is $1928k^2 + 492k + 2$ spm.

Encryption

Let's refer to the Somsuk-RSA encryption process in Algorithm 3.2. From Step 1, we can see that the maximum size of m is $4k$-bit. Next, in Step 2, the modular exponentiation of a $4k$-bit integer with a $4k$-bit integer is needed to obtain c. The running time in Step 2 is $192k^3 + 96k^2 + 12k$ spm, and it is the total running time for Somsuk-RSA encryption.

Decryption

Let's refer to the Somsuk-RSA decryption process as in Algorithm 3.3. The maximum size for c is $4k$-bit. In Step 1, computing m requires two modular exponentiations of a $4k$-bit integer with a $4k$-bit integer. First, the running time to obtain $c^{-1} \pmod{n}$ is $1920k^2 + 480k$ spm. Second, the running time to obtain $(c^{-1})^x \pmod{n}$ require $96k^3 + 72k^2 + 12k$ spm. Hence, the total running time for Somsuk-RSA decryption is $96k^3 + 1992k^2 + 492k$ spm.

3.2.2 Memory Cost for Somsuk-RSA Key Generation, Encryption and Decryption

Table 3.4, Table 3.5, and Table 3.6 show the memory consumption of the system parameters and the accumulators for Somsuk-RSA key generation, encryption, and decryption, respectively.

Table 3.4 Memory Cost for Somsuk-RSA Key Generation

Category	Register Name	No. Of Register	Bits
System parameters	N, e	$2\times 4k$	$8k$
	x	$1\times k$	k
Accumulators	p, q	$2\times 2k$	$4k$
	$\phi(N), d$	$2\times 4k$	$8k$
		Total	$21k$

Table 3.5 Memory Cost for Somsuk-RSA Encryption

Category	Register Name	No. Of Register	Bits
System parameters	e, N	$2\times 4k$	$8k$
Accumulators	c, m	$2\times 4k$	$8k$
		Total	$16k$

Table 3.6 Memory Cost for Somsuk-RSA Decryption

Category	Register Name	No. Of Register	Bits
System parameters	N	$1\times 4k$	$4k$
	x	$1\times k$	k
Accumulators	c,m	$2\times 4k$	$8k$
		Total	$13k$

3.3 Running Time and Memory Cost Evaluation for MRSA

Suppose we consider the MRSA key generation, encryption, and decryption as follows.

Algorithm 3.4 MRSA Key Generation Algorithm

1: Choose four distinct random prime w,x,y,z such that $2^k < w,x,y,z < 2^{k+1}$.
2: Compute $N = wxyz$ and $\phi(N) = (w-1)(x-1)(y-1)(z-1)$.
3: Choose e such that $1 < e < \phi(N)$ and $\gcd(e,\phi(N)) = 1$.
4: Choose f such that $1 < f < \phi(N)$ and $\gcd(f,\phi(N)) = 1$.
5: Compute d such that $ed \equiv 1 \pmod{\phi(N)}$.
6: Compute g such that $fg \equiv 1 \pmod{\phi(N)}$.
7: Return the public key (N,e,f), the private key (N,d,g).

Algorithm 3.5 MRSA Encryption Algorithm

1: Choose integer $0 < m < N$ such that $\gcd(m,N) = 1$.
2: Compute $c \equiv (m^e \pmod N)^f \pmod N$.
3: Return the ciphertext c.

Algorithm 3.6 MRSA Decryption Algorithm

1: Compute $m \equiv (c^d \pmod N)^g \pmod N$.
2: Return the plaintext m.

3.3.1 Running Time Estimation for MRSA Key Generation, Encryption and Decryption

Key Generation

Let's refer to the MRSA key generation process as in Algorithm 3.4. For Step 2, N is the product of four k-bit. Thus, its running time is $6k^2+8k+3$ spm. Furthermore, $\phi(N)$ is the product of four k-bit with a running time also $6k^2+8k+3$ spm. Step 5 and Step 6 involve a modular inversion of two $4k$-bit integers, and then the running time for both steps is $1920k^2+120k$ spm. Hence, the total running time for MRSA key generation is $3852k^2+256k+6$ spm.

Encryption

Let's refer to the MRSA encryption process as in Algorithm 3.5. From Step 1, we can see that the maximum size of m is $4k$-bit. Step 2 involves two modular exponentiations of integer with $4k$-bit integer to obtain c. The running time in Step 2 is $384k^3+192k^2+24k$ spm, and it is the total running time for MRSA encryption.

Decryption

Let's refer to the MRSA decryption process in Algorithm 3.6. The maximum size for c is $4k$-bit. In Step 1, computing m also required two modular exponentiations of a $4k$-bit integer with a $4k$-bit integer. The running time for Step 1 is $384k^3+192k^2+24k$ spm. Hence, the total running time for MRSA decryption is also $384k^3+192k^2+24k$ spm.

3.3.2 Memory Cost for MRSA Key Generation, Encryption and Decryption

Table 3.7, Table 3.8, and Table 3.9 show the memory consumption of the system parameters and the accumulators for MRSA key generation, encryption, and decryption, respectively.

Table 3.7 Memory Cost for MRSA Key Generation

Category	Register Name	No. Of Register	Bits
System parameters	N,e,d,f,g	$5\times 4k$	$20k$
Accumulators	w,x,y,z	$4\times k$	$4k$
	$\phi(N)$	$1\times 4k$	$4k$
		Total	$28k$

Table 3.8 Memory Cost for MRSA Encryption

Category	Register Name	No. Of Register	Bits
System parameters	e,f,N	$3\times 4k$	$12k$
Accumulators	c,m	$2\times 4k$	$8k$
		Total	$20k$

Table 3.9 Memory Cost for MRSA Decryption

Category	Register Name	No. Of Register	Bits
System parameters	N,d,g	$3\times 4k$	$12k$
Accumulators	c,m	$2\times 4k$	$8k$
		Total	$20k$

3.4 Running Time and Memory Cost Evaluation for ESF-RSA

Suppose we consider the ESF-RSA key generation, encryption, and decryption as follows.

Algorithm 3.7 ESF-RSA Key Generation Algorithm

1: Choose distinct random prime p,q such that $2^k < p,q < 2^{k+1}$.
2: Compute $N = pq$.
3: Find r,s where r/s is simple fraction of $(p-1)/(q-1)$.
4: Compute w such that $w = s(p-1)+1 = r(q-1)+1$.
5: Find u where $u \mid w$.
6:Obtain v where $w = uv$.
6: Return the public key (N,u) and the private key (N,v).

Algorithm 3.8 ESF-RSA Encryption Algorithm

1: Choose integer $0 < m < N$ such that $\gcd(m, N) = 1$.
2: Compute $c \equiv m^u \pmod{N}$.
3: Return the ciphertext c.

Algorithm 3.9 ESF-RSA Decryption Algorithm

1: Compute $m \equiv c^v \pmod{N}$.
2: Return the plaintext m.

3.4.1 Running Time Estimation for ESF-RSA Key Generation, Encryption and Decryption

Key Generation

Let's refer to the ESF-RSA key generation process as in Algorithm 3.7. For Step 2, N is the product of two $2k$-bit; thus, its running time is $4k^2 + 4k + 1$ spm. Step 3 involves the division of a $2k$-bit integer with a $2k$-bit integer, so the running time is k spm. In Step 4, computing w involves the multiplication of two $2k$-bit integers. Then the running time is $4k^2 + 4k + 1$ spm. Step 5 involves factorisation of w to find u since u is a divisor of w. However, no precise measurement for the factorisation process is available in the literature; therefore, it is represented as Ω spm. Step 6 requires $3k^2 + 9k$ spm for division of $4k$-bit with k-bit. Hence, the total running time for ESF-RSA key generation is $11k^2 + 17k + 2 + \Omega$ spm.

Encryption

Let's refer to the ESF-RSA encryption process as in Algorithm 3.8. From Step 1, we can see that the maximum size of m is $4k$-bit. Next, in Step 2, the modular exponentiation of a $4k$-bit integer with a $4k$-bit integer is needed. The running time in Step 2 is $48k^3 + 60k^2 + 12k$ spm, and it is the total running time for ESF-RSA encryption.

Decryption

Let's refer to the ESF-RSA decryption process as in Algorithm 3.9. The maximum size for c is $4k$-bit. In Step 1, to compute m it required modular exponentiation of a $4k$-bit integer with a $4k$-bit integer with $144k^3 + 84k^2 + 12k$ spm. Hence, the total running time for ESF-RSA decryption is $144k^3 + 84k^2 + 12k$ spm.

3.4.2 Memory Cost for ESF-RSA Key Generation, Encryption and Decryption

Table 3.7 Memory Cost for ESF-RSA Key Generation

Category	Register Name	No. Of Register	Bits
System parameters	N	$1 \times 4k$	$4k$
	u	$1 \times k$	k
	v	$1 \times 3k$	$3k$
Accumulators	p, q, r, s	$4 \times 2k$	$8k$
	w	$1 \times 4k$	$4k$
		Total	$20k$

Table 3.8 Memory Cost for ESF-RSA Encryption

Category	Register Name	No. Of Register	Bits
System parameters	u	$1 \times k$	k
	N	$1 \times 4k$	$4k$
Accumulators	c, m	$2 \times 4k$	$8k$
		Total	$13k$

Table 3.9 Memory Cost for ESF-RSA Decryption

Category	Register Name	No. Of Register	Bits
System parameters	v	$1 \times 3k$	$3k$
	N	$1 \times 4k$	$4k$
Accumulators	c, m	$2 \times 4k$	$8k$
		Total	$15k$

3.5 Running Time and Memory Cost Evaluation for Phony-RSA

Suppose we consider the Phony-RSA key generation, encryption, and decryption as follows.

Algorithm 3.10 Phony-RSA Key Generation Algorithm

1: Choose four distinct random prime w, x, y, z such that $2^k < w, x, y, z < 2^{k+1}$.
2: Compute $N = wxyz$ and $\phi(N) = (w-1)(x-1)(y-1)(z-1)$.
3: Find $X = \phi(N)v + 1$ where $X > N$ is a prime and $v \in \mathbb{Z}$.
4: Compute $\phi(X) = X - 1$.
5: Choose j, g such that $1 < j, g < 2^k$.
6: Compute private key exponent $d = jg$.
7: Find public key exponent $e = d^{-1} \pmod{\phi(X)}$.
6: Return the public key (N, u) and the private key (N, v).

Algorithm 3.11 Phony-RSA Encryption Algorithm

1: Choose integer $0 < m < N$ such that $\gcd(m, N) = 1$.
2: Compute $c \equiv m^u \pmod{N}$.
3: Return the ciphertext c.

Algorithm 3.12 Phony-RSA Decryption Algorithm

1: Compute $m \equiv c^v \pmod{N}$.
2: Return the plaintext m.

3.5.1 Running Time Estimation for Phony-RSA Key Generation, Encryption and Decryption

Key Generation

Let's refer to the Phony-RSA key generation process as in Algorithm 3.10. For Step 2, N and $\phi(N)$ is the product of four k-bit; thus, both running time is $6k^2 + 8k + 3$ spm each. In step 3, X is the product of $4k$-bit with k-bit, and the running time is $4k^2 + 5k + 1$ spm. In addition, there is also involved looping of primality testing to ensure X is prime and it represented as ξ spm. Finding

d in Step 6 required $2k^2+4k+2$ since it is the product of two k -bit. In step 7, e is the modular inversion of $5k$ -bit; therefore, the process requires $480k^2+240k$ spm. In step 8, finding f involves multiplying $5k$ -bit with k -bit integer and required $5k^2+6k+1$ spm. Consequently, the total running time for Phony-RSA key generation is $503k^2+271k+10+\xi$ spm.

Encryption

Let's refer to the Phony-RSA encryption process as in Algorithm 3.11. From Step 1, we can see that the maximum size of m is $4k$ -bit. Next, in Step 2, the modular exponentiation of a $4k$ -bit integer with a $5k$ -bit integer is needed. The running time in Step 2 is $360k^3+150k^2+15k$ spm, and it is the total running time for Phony-RSA encryption.

Decryption

Let refer to the Phony-RSA decryption process as in Algorithm 3.12. The maximum size for c is $5k$ -bit. In Step 1, to compute m it required modular exponentiation of a $5k$ -bit integer with a $5k$ -bit integer with $75k^3+90k^2+15k$ spm. Hence, the total running time for Phony-RSA decryption is $75k^3+90k^2+15k$ spm.

3.5.2 Memory Cost for Phony-RSA Key Generation, Encryption and Decryption

Table 3.10 Memory Cost for Phony-RSA Key Generation

Category	Register Name	No. Of Register	Bits
System parameters	X	$1\times5k$	$5k$
	g	$1\times k$	k
	f	$1\times6k$	$6k$
Accumulators	w,x,y,z	$4\times k$	$4k$
	$N,\phi(N)$	$2\times4k$	$8k$
	d	$1\times2k$	$2k$
	e	$1\times5k$	$5k$
	j	$1\times k$	k
		Total	$32k$

Table 3.11 Memory Cost for Phony-RSA Encryption

Category	Register Name	No. Of Register	Bits
System parameters	f	$1\times6k$	$6k$
	X	$1\times5k$	$5k$
Accumulators	c,m	$2\times4k$	$8k$
		Total	$19k$

Table 3.12 Memory Cost for Phony-RSA Decryption

Category	Register Name	No. Of Register	Bits
System parameters	g	$1\times k$	k
	X	$1\times5k$	$5k$
Accumulators	c,m	$2\times4k$	$8k$
		Total	$14k$

4.0 DISCUSSION

4.1 Overall Running Time in Single-precision Multiplication

This section studies running time in terms of spm for RSA and its selected variants. Table 4.1 below shows all collected data from the section before.

Table 4.1 Running Time in Single Precision Multiplication.

Cryptosystem	Key Gen. (spm)	Encryption (spm)	Decryption (spm)
RSA	$1928k^2+488k+2$	$192k^3+96k^2+12k$	$192k^3+96k^2+12k$
Somsuk-RSA	$1928k^2+488k+2$	$192k^3+96k^2+12k$	$96k^3+1992k^2+492k$
MRSA	$3852k^2+256k+6$	$384k^3+192k^2+24k$	$384k^3+192k^2+24k$
ESF-RSA	$11k^2+17k+2+\Omega$	$48k^3+60k^2+12k$	$144k^3+84k^2+12k$
Phony-RSA	$503k^2+271k+10+\xi$	$360k^3+150k^2+15k$	$75k^3+90k^2+15k$

RSA has the lowest spm measurement for a key generation, while MRSA has the highest. Even though ESF-RSA and Phony-RSA seem to have the lowest spm, both have Ω and ξ variables whose size is immeasurable. The higher spm required in MRSA is due to two modular inversion processes to find d and g during the key generation. A modular inversion is like 30 times slower than a modular multiplication.

Next, ESF-RSA has the lowest spm for encryption, which is $48k^3+60k^2+12k$, compared to MRSA, which has the highest spm, which is $384k^3+192k^2+24k$ spm. The lower spm for ESF-RSA to encrypt the plaintext was because the public key u is small. On the other hand, MRSA encryption involves two modular exponentiations causing the spm to be higher. For RSA and Somsuk-RSA, the spm reading shows the same measurement since both cryptosystems have the same encryption algorithm. Also, Phony-RSA is a second higher spm for encryption because it performs encryption using larger phony public key f.

For decryption, Phony-RSA has five times lower spm than MRSA because Phony-RSA uses only a small phony private key g rather than MRSA, which has the largest spm measurement requiring two modular exponentiations to decrypt the ciphertext. RSA and ESF-RSA almost have the same spm measurement

because the algorithm for decrypting is the same but different in the private key size.

4.2 Total Memory Cost and Actual Running Time

This subsection analyses total memory cost in bits and actual running time in seconds for RSA and its selected variants. Table 4.2 below shows all collected total memory costs and running times for all cryptosystems from the chapter before.

Actual running time is collected by running the algorithms in Maple software to acquire the average actual running time of the cryptosystems. Algorithms were run 200 times with constant m for each cryptosystem to acquire the average running time.

Table 4.2 Total Memory Cost and Actual Running Time.

Cryptosystem	Key Gen.		Encryption		Decryption	
	bits	second	bits	second	bits	second
RSA	$20k$	0.820	$16k$	0.026	$16k$	0.024
Somsuk-RSA	$21k$	0.880	$16k$	0.028	$13k$	0.010
MRSA	$28k$	7.276	$20k$	0.044	$20k$	0.043
ESF-RSA	$20k$	1.163	$15k$	0.003	$13k$	0.025
Phony-RSA	$32k$	4.901	$19k$	0.032	$14k$	0.009

From the table above, Phony-RSA has the largest total memory consumption in key generation. Phony-RSA key generation produces four types of key exponent and uses phony modulus X causing it to consume large memory. In addition, MRSA has a second larger total memory consumption, which is $28k$ -bits; consequently, MRSA also generates four key exponents. In the key generation process, RSA and ESF-RSA have the same total memory consumption at $20k$ -bits. RSA has the fastest key generation among other variants with 0.820 sec. ESF-RSA is slower than RSA since it involves finding u.

Similarly, Phony-RSA is even slower than ESF-RSA caused by the looping of the primality testing algorithm to obtain X. On the contrary, MRSA is the slowest algorithm to generate keys because MRSA generates four different keys rather than two. MRSA also has two modular inversions that make it even slower.

Then, in encryption, ESF-RSA consumes the smallest memory at $13k$ -bit because of storing a smaller public key u. Following ESF-RSA are RSA and Somsuk-RSA, which have $16k$ -bit in total memory consumption due to RSA and Somsuk-RSA having the same size as the key and modulus N. However, MRSA must store two different public key exponents during the encryption process, making MRSA the largest total memory consumption at $20k$ -bits. Although slower during key generation, ESF-RSA is the fastest algorithm to encrypt plaintext. The smaller public key u caused the ESF-RSA to compute quicker with an average of 0.003 sec only.

Nevertheless, MRSA has the slowest running time, almost 15 times slower than ESF-RSA for encryption, requiring two modular exponentiations. RSA and Somsuk-RSA have identical actual time taken because both have the same encryption algorithm. Using a larger public key f makes Phony-RSA come to the second slower algorithm.

Besides, in decryption, MRSA also has the largest total memory consumption at $20k$ -bits because MRSA stored different private keys d and g. Other RSA variants have slightly different sizes of total memory consumption because they vary in size of the private key and modulus N or X. Phony-RSA becomes the fastest decryption algorithm with an average of 0.009 sec. Using a small private key g caused Phony-RSA to take less decrypting time than other variants. They are following Phony-RSA is Somsuk-RSA which runs at an average of 0.010 sec and uses a small private key x. Then, RSA and ESF-RSA have the same running time even though their private key and modulus differ. Lastly, MRSA is the slowest algorithm because it uses two modular exponentiations when decrypting.

5.0 CONCLUSION

This study conducted a comparative analysis of RSA and selected variants of the cryptosystem. The selected RSA variants are Somsuk-RSA, MRSA, ESF-RSA, and Phony-RSA.

The first objective is to compare the running time and memory consumption of key generation, encryption, and decryption for the RSA cryptosystem and four selected RSA variants cryptosystem. Using single-precision multiplication (spm) and actual running time via Maple, this study found that RSA is the fastest cryptosystem for key generation, ESF-RSA for encryption, and Phony-RSA for encryption-decryption. MRSA is the slowest cryptosystem of all processes. Then, for memory consumption, it is proven by analysis that RSA and ESF-RSA consume the smallest memory for key generation. ESF-RSA also uses the least memory for encryption, whereas Phony-RSA uses the least memory for decryption. Although Phony-RSA consumes the smallest memory for decryption, it uses the largest memory space for the key generation process, whereas MRSA uses the largest for encryption and decryption.

The second objective, which is to simulate the RSA cryptosystem and four selected RSA variants cryptosystem using Maple programming, is achieved with several adjustments to the original algorithm. The change was made to adapt the algorithm with Maple coding since some algorithms cannot be translated into Maple coding. Even though some modifications were made, the modified algorithm still produces the same output as the original algorithm. However, the results of this study were acquired only using Maple programming. As a result, instead of utilizing Maple, various alternative programming languages such as MATLAB, Java and C programming can be used to improve the results.

Conflicts of Interest

The author(s) declare(s) that there is no conflict of interest regarding the publication of this paper.

Acknowledgement

The authors extend their appreciation to Universiti Malaysia Sabah for funding this work through Research Grant SBk0508-2021.

References

[1] Canetti, R., Halevi, S. and Katz, J. 2004. Chosen-ciphertext Security from Identity-based Encryption. Advances in Cryptology-EUROCRYPT 2004: *International Conference on the Theory and Applications of Cryptographic Techniques, Interlaken, Switzerland, May 2-6, 2004*. Springer Berlin Heidelberg. 23: 207-222. Doi: https://doi.org/10.1007/978-3-540-24676-3_13.

[2] Barakat, M., Eder, C. and Hanke, T., 2018. An Introduction to Cryptography. Timo Hanke at RWTH Aachen University. 1-145. Retrieved: 1 July 2023.

[3] Alqad, Z., Oraiqat, M., Almujafet, H., Al-Saleh, S., Al Husban, H. and Al-Rimawi, S. 2019. A New Approach for Data Cryptography. *International Journal of Computer Science and Mobile Computing*. 8(9): 30-48.

[4] Saloma, A. 2006. *Public-key Cryptography*. 2nd edition. Springer Science & Business Media.

[5] Pal, S. K. and Mishra, S. 2019. An TPM based Approach for Generation of Secret Key. *International Journal of Computer Network and Information Security*. 11(10): 45-50. Doi: https://doi.org/10.5815/ijcnis.2019.10.06.

[6] Menezes, A. J., Van Oorschot, P. C. and Vanstone, S. A. 2018. *Handbook of Applied Cryptography*. CRC Press. Doi: https://doi.org/10.1201/9781439821916.

[7] Asbullah, M. A. and Ariffin, M. R. K. 2014. Comparative Analysis of Three Asymmetric Encryption Schemes based Upon the Intractability of Square Roots Modulo N= p^ 2 q. *4th International Cryptology and Information Security Conference*. 24-26.

[8] Diffie, W. and Hellman, M. E. 2022. New Directions in Cryptography. *Democratizing Cryptography: The Work of Whitfield Diffie and Martin Hellman*. 365-390. Doi: https://doi.org/10.1145/3549993.3550007.

[9] Hue, S. Y., Sarmin, N. H., Ismail, E. S. and Chin, J. J. 2020. December. Easy Simple Factoring-based Digital Signature Scheme. *2020 15th International Conference for Internet Technology and Secured Transactions (ICITST)*. 1-4). IEEE. Doi: https://doi.org/10.23919/ICITST51030.2020.9351341.

[10] Rivest, R.L., Shamir, A. and Adleman, L. 1978. A Method for Obtaining Digital Signatures and Public-key Cryptosystems. *Communications of the ACM*. 21(2): 120-126. Doi: https://doi.org/10.1145/359340.359342.

[11] Al Hasib, A. and Haque, A. A. M. M. 2008. A Comparative Study of the Performance and Security Issues of AES and RSA Cryptography. *2008 Third International Conference on Convergence and Hybrid Information Technology*. IEEE. 2: 505-510. Doi: https://doi.org/10.1109/ICCIT.2008.179.

[12] Somsuk, K. 2017, November. The New Equation for RSA's Decryption Process Appropriate with High Private Key Exponent. 2017 21st International Computer Science and Engineering Conference (ICSEC). IEEE. 1-5. Doi: https://doi.org/10.1109/ICSEC.2017.8443858.

[13] Somsuk, K. 2021. A New Methodology to Find Private Key of RSA Based on Euler Totient Function. *Baghdad Science Journal*. 18(2): 338-348. Doi: https://doi.org/10.21123/bsj.2021.18.2.0338.

[14] Islam, M. A., Islam, M. A., Islam, N. and Shabnam, B. 2018. A Modified and Secured RSA Public Key Cryptosystem based on "n" Prime Numbers. *Journal of Computer and Communications*. 6(03): 78. Doi: https://doi.org/10.4236/jcc.2018.63006.

[15] Imam, R., Anwer, F. and Nadeem, M. 2022. An Effective and Enhanced RSA based Public Key Encryption Scheme (XRSA). *International Journal of Information Technology*. 14(5): 2645-2656. Doi: https://doi.org/10.1007/s41870-022-00993-y

[16] Ismail, E. S., Zaharidan, M. Z. and Samat, F. A. I. E. Z. A. 2018. ESF: Suatu Kriptosistem Mudah Ringkas Berasaskan Masalah Pemfaktoran. *Journal of Quality Measurement and Analysis JQMA*. 14(2): 81-89.

[17] Raghunandan, K. R., Aithal, G. and Shetty, S. 2019. Secure RSA Variant System to Avoid Factorization Attack using Phony Modules and Phony Public Key Exponent. *Int J Innov Technol Exploring Eng (IJITEE)*. 8(9). Doi: https://doi.org/10.35940/ijitee.I7807.078919.

[18] Puneeth, B. R., Raghunandan, K. R., Bhavya, K., Shetty, S., NS, K. R., Dodmane, R. and Islam, S. M. 2022. Preserving Confidentiality against Factorization Attacks using Fake Modulus ($\zeta $) Approach in RSA and its Security Analysis. *2022 IEEE 9th Uttar Pradesh Section International Conference on Electrical, Electronics and Computer Engineering (UPCON) IEEE*. 1-6.

[19] Asbullah, M. A., Ariffin, M. R. K. and Mahad, Z. 2016. Analysis on the Rabin-p Cryptosystem. *AIP Conference Proceedings AIP Publishing*. 1787(1). Doi: https://doi.org/10.1063/1.4968151.

[20] Vuillaume, C. 2003. Efficiency Comparison of Several RSA Variants Master Thesis. Fachbereich Informatik der TUDarmstadt.