Crypto and Information Security (2023)
Clausius Scientific Press, Canada

DOI: 10.23977/crypis.2023.020101
ISSN 2522-6436 Vol. 2 Num. 1

RSA Cryptosystem Speed Security Enhancement (Hybrid and Parallel Domain Approach)

Moise Ngendahimana[1,a,*], Wei Shen[1,b]

[1]*College of Information Science and Engineering, Zhejiang Sci-Tech University, Hangzhou, Zhejiang, 310018, China*
[a]*ngendahimana_moise@hotmail.com*, [b]*latitude@123.com*
**Corresponding author*

Keywords: Cryptography; Pseudo-random numbers; Fast Exponentiation; Chinese Remainder Theorem; RSA Encryption and Decryption; Linear Congruential Generator

Abstract: Encryption involves every aspect of working with and learning about codes. Over the last 40 years, it has grown in prominence to become a prominent scholarly discipline. Because most interactions now take place online, people require secure means of transmitting sensitive information. Several modern cryptosystems rely on public keys as a crucial component of their architecture. The major purpose of this research is to improve the speed and security of the RSA algorithm. By employing Linear Congruential Generator (LCG) random standards for randomly generating a list of large primes; and by employing other selected algorithms, such as the Chinese Remainder Theorem (CRT) in decryption, exponent selection conditions, the Fast Exponentiation Algorithm for encryption, and finally, a comparison of the enhanced RSA versus the normal RSA algorithm that shows an improvement will be provided.

1. Introduction

With the use of cryptography, sensitive information may be concealed from prying eyes. The protection provided by urban touchable and physical devices against data access by unauthorized parties is insufficient. Therefore, specialists and developers need to build and extend safety mechanisms to protect data and prevent attacks from starting at such a crucial point. For this reason, “encryption” was borrowed from elsewhere; it is a crucial component of any adequate safety system and a prerequisite for any practical means of influencing or creating such a system. Skill is required to keep the secret from random people. The importance of data encryption over simple message transport is growing as our collective knowledge base expands. Cryptography is used to protect this information, and it may be broadly classified into two subfields: secret-key and public-key cryptography [1].

Today’s cryptography relies significantly on the principles of mathematics and computer science. Due to the computational hardness assumptions used in their creation, cryptographic algorithms are very difficult for an adversary to crack in reality. It is conceivable in theory to crack such a system, but doing so in practice is currently impossible. Therefore, we call these methods ”computationally

safe," and we must always be adapting them to keep up with theoretical developments and quicker processing power. Some systems, such as the one-time pad, are provably safe in terms of information theory and cannot be broken even with infinite computing power [2]. These schemes, however, are more difficult to put into practice than the best theoretically breakable but computationally secure techniques.

The RSA algorithm relies on a certain prime number feature. The prime numbers: 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, . . . are positive integers that can't be written as the product of factors of integers. In other words, any natural number bigger than one and divisible solely by itself and one is considered prime.

Random numbers play a crucial role in cryptography, for example, in the generation of keys. Random numbers used in cryptography must be generated in an unpredictable and random manner. In other words, the process of creating new numbers should adhere to a regular pattern, and new sequences of numbers should be impossible to predict based on existing sequences. Keeping an ear out for radio or noise interference is a great approach to giving random number generation some real-world context. In contrast, some methods rely on a piece of software called a "Pseudo-random Number Generator" to generate numbers that are statistically very similar to random [3].

1.1. Problem statement

The parallel domain (implementation of a parallel or multi-threading scheme), the hybrid domain (combination of RSA and one or two other algorithms), and mathematical modifications in the "Core-Modifications" category are the most popular domains for RSA variant proposals, according to the systematic review [4]. With the common goal of improving overall execution speed and security, but no studies even attempt to explore how these big prime numbers are generated, despite the fact that this is one of the key causes of the RSA algorithm's slowness. If RSA were faster, it would be a very useful algorithm that could be employed in our everyday computing applications. As a result, we should apply any strategy available to all RSA stages, from prime generation, key generation to encryption-decryption, even if it is an addition or multiplication operation, as long as it is the quickest way to optimize its total execution time, memory usage, security, and so on. The goal of this research is to modify the Standard RSA Algorithm based on the public key exponent "e" and compute the modulus N. For security and speed enhancement, we will use additional algorithms and mathematical parallel implementations without making any changes to the Standard RSA basic mathematical structure. With the algorithms utilized, the comparison will be confined to the Standard RSA Algorithm and the Modified RSA Algorithm. The study's goal is to emphasize the importance of the RSA algorithm.

1.2. Our Contributions

A list of pseudo-random integers is generated using a linear congruential generator in order to perform RSA key creation using the Rabin-Miller test method. To speed up the transmission of the algorithm, this study introduced a unique RSA technique that employs LCG for prime generation, CRT for decryption, and a smaller exponent e as a Fermat number less than the square root of Φ (n). While using CRT to maintain the algorithm's security, we also provided other security measures to improve the algorithm's security, such as unencryptable messages, paired private and public keys, and the value of d having to be as large as the modulus N's magnitude. All other suggested methods and mathematical techniques were thoroughly discussed, and algorithm pseudo-codes and Python code were provided. The normal RSA method is compared to the proposed smaller-exponent LCG-CRT-RSA algorithm in the study. The results show that, when compared with normal RSA, the LCG-CRT-RSA algorithm increases RSA encryption speed and data security. As a result of using a hybrid

and parallel domain approach, the study contribution increases the speed and security of the RSA algorithm.

1.3. Organization

This research study is based on the speed and security of the RSA cryptosystem algorithm in hybrid and parallel domain techniques. The paper is broken into sections. The introduction, problem statement, outcomes, and contributions are all included in **Part 1**. **Section 2** addresses the technical background of cryptography; **Section 3** examines the literature review, and **Section 4** provides an in-depth look at the RSA. **Section 5** comprises all of the implemented methods and mathematical methodologies for the improved RSA algorithm. **Section 6** contains the improved RSA algorithm. **Section 7** delves deeper into the results and debate. Finally, **Part 8** will summarize the research with recommendations for future work, and the appendix section provides all of the Python codes used for creating our modified RSA and results on GitHub [5].

2. Background

Cryptography ensures data integrity in information security, providing assurance that data has not been altered with hash algorithms. One other benefit of using digital signatures is that they may be used for non-repudiation or authentication of the parties involved [6], they are two types of cryptography.

2.1. Symmetric key cryptography

Symmetric cryptography **Figure 1** shows the simplest type of encryption when both parties use the same secret key to encrypt and decrypt data. They are two distinct categories of symmetric key cryptography designated by the quantity of data they are able to process: block ciphers and stream ciphers. The block cipher organizes the input data into blocks or groups of consistent size (within a few bytes), facilitating efficient encryption and decryption, while the stream cipher converts the data format suitable for encryption and decryption.

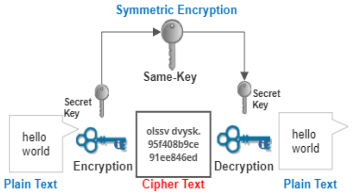


Figure 1: Symmetric Cryptosystem.

Examples of symmetric encryption algorithms include Blowfish, AES, RC4, DES, RC5, and RC6. One of the primary advantages of symmetric algorithms is their ability to be used in real-time systems, which asymmetric algorithms cannot because each user needs their own unique key.

2.2. Asymmetric key cryptography

The use of two distinct keys is at the heart of asymmetric cryptography, commonly known as public-key cryptography. When encrypting plain text with asymmetric encryption, two keys are used a public one that may be shared openly and a private one that shouldn't be revealed under any circumstances. Asymmetric cryptography provides not just the privacy that encryption does but also using digital signatures it is possible to acquire authentication, non-repudiation, and integrity. The context of a signed message is compared to the input used to generate the signature [7]. To counter

the fact that anybody in possession of the secret key may read an encrypted communication, asymmetric encryption makes use of a pair of keys that are mathematically connected to one another **Figure 2**.

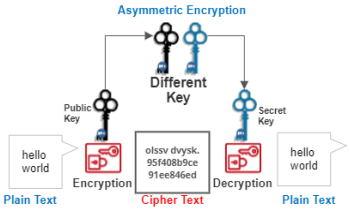


Figure 2: Asymmetric Cryptosystem.

RSA, DSA, and Diffie-Hellman are all examples of asymmetric algorithms. Using symmetric cryptography with a straightforward and safe key exchange system enabled by an asymmetric key method makes it possible to have a quick technique that generates compact cipher texts [8].

3. Related work

Many enhancements were made to the RSA algorithm by many researchers [4]. Some relevant studies are included below:

For the hybrid domain, which is the improvement of Classical RSA and one or two additional algorithms. Rebalanced RSA and Multi-Prime RSA were proposed by Boneh and Shacham [9]. A strategy for integrating two previously improved RSA versions was developed by Alison et al. [10]. To increase the security levels and overall execution speed of the method, Gupta and Sharma [11] integrated RSA with the Diffie-Hellman public key cryptographic technique. Based on an additive homomorphic property, Dhakar et al. [12] introduced the MREA (Modified RSA Encryption Algorithm), and they demonstrated that it is far more secure than the regular RSA and extremely resistant to brute force attacks. The RSA and El-Gamal cryptosystems were joined by Ahmed et al. [13] using the Discrete Logarithm Problem (DLP) for El-Gamal and the Integer Factorization Problem (IFP) for RSA. For asymmetric cryptosystems, the pairing of IFP with DLP gave a reasonable processing speed. The El-Gamal and RSA algorithms were less effective than the indicated system computations as a result. In order to secure the upload of data to the cloud, Mahalle and Shahade [14] presented an RSA variation that makes use of the AES method and three keys (a public key, a private key, and a secret key). The authors' conclusion is that this AES-RSA hybrid will effectively give cloud users data security.

In order to make factorization more difficult overall, Arora and Pooja [15] proposed a novel algorithm in 2015. This algorithm uses a hybrid of RSA and the El-Gamal algorithm. AES + RSA and Twofish + RSA, a hybrid implementation of one symmetric algorithm with another asymmetric algorithm, were recently proposed by Jintcharadze and Iavich [16]. They came to the conclusion that RSA + Twofish outperforms the abovementioned hybrid algorithms in terms of speed and memory use. Alamsyah and others [17] to increase the security of two-factor authentication, combined RSA and the one-time pad approach.

For multi-threading or parallel techniques. C. W. Chiou [18] compares a fresh modular exponentiation technique to reduce the time it takes for modular exponentiation to execute in 1993. Because of fewer operations, this method offered a roughly 33% greater throughput. Ayub et al. [19] developed an OpenMP-based parallel CPU-based RSA method implementation that parallelizes the algorithm's exponentiation phase to aid in speedy encryption and decryption. Their analysis concludes that the program's execution time has been improved. Rahat et al. [20] improved efficiency by using a unique parallel data structure termed a Concurrent Indexed List of character blocks in 2019. The essay presented three different simultaneous RSA implementations. With a possible speed-up

factor of 4.5.

For the CRT enhancements domain. Wu et al. [21] presented a CRT-RSA in 2001. Their proposal use Montgomery's algorithm rather than CRT, yielding better results for decryption and digital signatures. Blomer et al. [22] presented another CRT-RSA for using CRT to solve fault attack vulnerabilities on RSA-based signature algorithms. According to them, CRT-RSA is widely employed in smart card transactions. Sony et al. [23] proposed using multiple keys and CRT to improve data transmission security by increasing processing time and algorithm security. Quisquater and Couvreur [24] proposed a rapid decryption technique in 1982. Based on previously reported Standard RSA weaknesses, they developed an RSA deciphering method that employs an enhanced modular exponentiation methodology and is based on the CRT, with the goal of increasing overall performance time.

Finally, Aiswarya et al. [25] proposed a novel Binary RSA Encryption Algorithm (BREA) encryption method in 2017. Its security is further increased by converting the encrypted cipher text generated by the Modified RSA Encryption Algorithm (MREA) into binary code format. As a result, the intruder will struggle to decrypt the data. In the same year, Sahu et al. [26] presented a more secure approach than the original by making modulus n private as well.

4. RSA Algorithm

In 1977, Ron Rivest, Adi Shamir, and Leonard Adleman of the Massachusetts Institute of Technology revealed the public description of the Rivest Shamir Adleman (RSA) algorithm [27]. The system's safety comes from the difficulty of factoring the products of two large prime numbers. This difficulty is the foundation of the one-way RSA core function, which is straightforward to calculate in one direction but prohibitively so in the other. Consequently, RSA is secure since it is mathematically impossible to obtain such numbers or it would take too much time to do so, regardless of the available computing power. In addition, the encryption's safety is directly proportional to the size of the key. An algorithm's effectiveness increases exponentially when its size is doubled. Typical bit lengths for RSA keys are 2048 or 4096.

RSA isn't just used for encryption; it's also the basis for digital signatures, in which only the owner of a private key may "sign" a message, but anybody can check its authenticity using the public key. Several protocols, including SSH, OpenPGP, S/MIME, and SSL/TLS, rely on RSA signature verification [28]. Many firms utilize RSA for personnel verification. Cryptographic techniques are used in chip-based smart cards to ensure security by verifying the PIN code [29]. Pretty Good Privacy (PGP), a freeware that provides encryption and authentication for e-mail and file storage applications, also uses RSA for key transfer.

Furthermore, SSL provides data security by establishing an RSA key exchange during the SSL handshake client-server authentication at the end of the connection between the internet-based communications protocol TCP/IP and application protocols such as HTTP, Telnet, Network News Transfer Protocol (NNTP), or File Transfer Protocol (FTP). The determinism of the original RSA encryption means that the same plain text will always yield the same cipher text when using the same key pair. Due to this feature, the technique is susceptible to a "selected clear text attack," in which an attacker generates cipher texts randomly from a pool of known clear texts and checks whether or not they are equal to previously generated cipher texts. An adversary can learn about the original data without having to decrypt it by making this comparison [3].

4.1. Structure of Classical RSA algorithm

- Generate Large Prime numbers p and q.
- Calculate modulus N = P ·Q.

- Calculate Phi Φ (N) = (P − 1) · (Q − 1).
- Select exponent e such that 1 < e < Φ (N) and gcd (e, Φ (N)) = 1.
- Compute d ·e ≡ 1 (mod Φ (N)).
- Encryption C = M^e (mod N).
- Decryption M = C^d (mod N).

5. Implementation of Algorithms

We used many theories and algorithm techniques to improve the total performance of the RSA cryptosystem using our modified RSA algorithm from prime number creation through key generation, encryption, and decryption. All of the approaches applied for the enhancement are listed here.

5.1. Linear Congruential Generator

Pseudo-Random Numbers are created deterministically (meaning that they can be replicated) and must look independent, with no visible patterns in the numbers. We utilized this approach to quickly construct a big list of odd integers for a subsequent primality test. The algorithm function's Python code is on my Github [5].

$$X_{n+1} = P_1X_n + P_2 \text{ (mod m) where: } n \in Z^+ \quad (1)$$

Where X_n is the random integers generated, P_1 is the multiplier, P_2 is the increment, and m is the modulus, X_0 is the initial seed value of the series. As in [29] Linear congruential generator of **Equation (1)** has a full period (cycle length of m) if and only if the equation meets the following conditions:

- gcd(P_2, m) = 1;
- $P_1 \equiv 1$(mod p));
- $P_1 \equiv 1$(mod 4);

With $m = 2^k$, $P_1 = 4b + 1$, P_2 as an odd number where (b, k > 0) we will get a full period of length m, for our RSA prime generation approach:

- The modulus m should be a known nearest prime from $2^{(n+1)}$, n being the n-bits key, this value is fixed (constant) in our approach.
- P_1 is the multiplier with the condition of 4 k + 1 with k ϵ [1, 2, 3, …] in our approach:

$$P_1 = 4*10^{(n\_digits\ -\ 1)} + k \quad (2)$$

Where k ϵ [1, 3… 2i + 1]

- P_2 is the increment and this is a variable, for reference, we took:

$$P_2 = 10^{(n\_digits)} + k \quad (3)$$

Where k ϵ [1, 3… 2i + 1] hence P2 is an odd number.

- X_0 is the seed and in our approach, we took any odd integer number of the same magnitude size in our expected range,

$$X_0 = 2^{(n-bits)} + k \quad (4)$$

Where k ϵ [1, 3… 2i + 1]

$$n_{digits} = n_{bits} * \log 2 \quad (5)$$

5.2. Miller Rabin Prime Primality testing

The Miller-Rabin primality test [31] is a probabilistic primality test that determines if a particular number is likely to be prime, which is known to be the simplest and quickest test known; this test employs Fermat's Little Theorem [2]. We need to generate two huge prime numbers during the RSA key generation process, and after that, we need to ensure that they are indeed prime. Let us verify "n" primality ⇐⇒ n is an odd number.

Algorithm 1: Miller-Rabin primality test

Find integer k and m such that: $n - 1 = 2^k \cdot m$

Choose randomly any integer a ε [1, n − 1]

Compute $b_0 = a^m \bmod n$

Compute b_i,"k" times, $b_i = b_{i-1}^2 \bmod n$

The result must be ±1:

If the solution is 1, n is a composite number and if the solution is -1, n is a prime number.

Repeat the test "i" times. The probability of a composite n passing "i" tests is 1/4i; for our implementation the probability is 2^{-128}. GitHub repository [5] has a Python code of this algorithm.

5.3. Karatsuba Multiplication Algorithm

Algorithm 2: Karatsuba Recursive Algorithm

Begin

INPUT: P, Q

OUTPUT: Product ← P · Q

if P < 10 or Q < 10 then

return $P \cdot Q$

else

n ← val(max digits number of P or Q)

$P \leftarrow 10^{n/2} \cdot P_1 + P_0$

$Q \leftarrow 10^{n/2} \cdot Q_1 + Q_0$

$P \cdot Q \leftarrow 10^{2(n/2)} P_1Q_1 + 10^{n/2}(P_1Q_0 + P_0Q_1) + P_oQ_o$

$t_o \leftarrow$ karatsuba (P_o, Q_0)

$t_{12} \leftarrow$ karatsuba $(P_1 + Q_0, Q_1 + Q_0)$

$t_3 \leftarrow$ karatsuba (P_1, Q_1)

return $10^{2(n/2)} t_3 + 10^{n/2} t_{12} + t_0$

end if

End.

Karatsuba algorithm is a fast multiplication algorithm [32]. We choose this approach for computing the modulus N = P·Q and Φ (N) for fast computing. This algorithm reduces the multiplication of two n-digit numbers to three multiplications of n/2-digit numbers and, by repeating this reduction, to at most n $log_2^3 \approx n^{1.58}$ single-digit multiplications See **Algorithm [2]**. It is therefore asymptotically faster than the traditional algorithm, which performs n2 single-digit products.

5.4. Extended Euclidean algorithm

Euclid's algorithm if gcd (a, b) = d, then there exist integers x, y such that ax + by = d.

The first step in the Euclidean method is to divide the bigger integer, a, by the smaller number, b, to get the quotient, q_1, and the remainder, r_1(less than b),

$$a = q_1 b + r_1$$

Next, we'll use b and r_1 to go on in the same manner, eventually arriving at the value:

$$b = q_2 r_1 + r_2$$

$$r_n{-}2 = q_n r_n{-}1 + r_n$$

Finding the largest common divisor is complete after we have achieved rn|rn−1. This is shown by the evidence below: After solving for rn^{-2} = qnrn−1 + rn, in the previous step, we arrive at the following equation:

$$rn^{-2} = q_n r_n{}^{-1} + r_n$$

Now, suppose d is the gcd of a and b. So we get <d|a and d|b>.
Hence, we get r_n as the Greatest Common Divisor (gcd) of a and b.

5.5. Fermat's Little Theorem

Fermat's Little Theorem Let a ∈ N and p be a prime number, then:

$$a^p \equiv a \bmod p \tag{6}$$

$$a^{p-1} \equiv 1 \bmod p \tag{7}$$

5.6. CRT Chinese Remainder Theorem

A system of two or more linear congruencies does not necessarily have a solution, even though each of the individual congruencies does [33]. The Chinese Remainder Theorem CRT describes the solutions for a system of simultaneous linear congruencies.

$$x \equiv a \bmod m \text{ and } x \equiv b \bmod n$$

have a unique solution if the modules are relatively prime to each other, two to two. Where gcd (m, n) = 1.

Let $m_1, m_2 \ldots m_k$ be a set of relatively prime numbers two by two, gcd (mi, mj) = 1 where i ≠ j

Let $a_1, a_2 \ldots a_k$ be arbitrary integers. The system of simultaneous congruencies:

$$\begin{cases} x \equiv a_1 \bmod m_1 \\ x \equiv a_2 \bmod m_2 \\ \quad\quad \ldots \\ \quad\quad \ldots \\ \quad\quad \ldots \\ x \equiv a_k \bmod m_k \end{cases}$$

Let $M = m_1 \cdot m_2 \cdots m_k$ and $M_k = \frac{M}{m_k}$, gcd (M_k, m_k) = 1, Each M_k admits a unique inverse M_k^{-1} modulus mk below:

$$M_k \cdot M_k^{-1} = 1 \bmod m_k$$

Therefore, the system has a unique solution x = y mod M:

$$(a_1 M_1 M_1^{-1} + a_2 M_2 M_2^{-1} + \cdots + a_k M_k M_k^{-1}) \bmod M \tag{8}$$

5.7. Fast Exponentiation Algorithm

The speed with which we can calculate M^e (mod N) for integer n of this magnitude is a significant aspect of public-key cryptography. The integers used in modern RSA keys are at least 1024 bits long.

The traditional method for raising to a power, say x^8 is as follows:

$$x \xrightarrow{Sqr} x^2 \xrightarrow{Mul} x^3 \xrightarrow{Mul} x^4 \xrightarrow{Mul} x^5 \xrightarrow{Mul} x^6 \xrightarrow{Mul} x^7 \xrightarrow{Mul} x^8$$

Where Sqr means squaring and Mul means multiplying.

We squared first, followed by six successive multiplications, a total of seven operations. In RSA, we would have to execute 21024 multiplications for a power of 1024 bits, which is not at all convenient. The square and multiply algorithm is the fastest way to perform this exponentiation.

Consider the same example shown above to calculate x^8 using square and multiply algorithm:

$$x \xrightarrow{Sqr} x^2 \xrightarrow{Sqr} x^4 \xrightarrow{Sqr} x^8$$

We only need 3 squaring operations, note that the square and multiplication operation has the same time complexity.

To know the number of squaring and multiplying operations required we convert the exponent to its binary equivalent, where we have ”1” we square and multiply in that order, and where we have ”0” we apply square operation only, This method is called the Fast exponentiation algorithm [34]. See **Algorithm [3]**.

Algorithm 3: Fast Exponentiation Algorithm

Begin

INPUT: M, e, N

OUTPUT: $X \leftarrow M^e \pmod N$

Convert Exponent e to Binary Base2

$B \leftarrow b_{k-1}b_{k-2}\ldots b_i\ldots b_2b_1b_0$

$X \leftarrow M \pmod N$

$i \leftarrow k - 1$

while $i > 0$ do

$X \leftarrow X^2 \pmod N$

if $b_i = 1$ then

$X \leftarrow X \cdot M \pmod N$

end if

$i \leftarrow i - 1$

end while

Return (X)

End.

5.8. Recommendations to select the value “e”

We recommended using a very small public key value "e" less than $\sqrt{\Phi(n)}$. Therefore the trap $\Phi(n) = (p - 1)(q - 1)$, p and q being primes of at least 1024 bits, and the size of $\Phi(n)$ will be approximately equal to that of n, a little smaller but with the same magnitude(number of bits). Since the public key "e" and the private key "d" are inverses in the field Φ (n), that is, **d = inv [e, Φ (n)]** then we get the following relation **e ɗ (mod Φ (n)) = 1**, for this equality to hold, the product "e ɗ" must leave the field Φ (n) at least once so that the operation in that module returns the value 1. In other words, it will be true that **e ɗ = k Φ (n) + 1, with k = 1, 2, 3, 4...**

For this product to come out at least once from the body Φ (n), that is with k = 1, given that the public key “e” has for example 17 bits, the value of the private key d should be at least greater than 1007 bits, for the hypothetical case (and with almost null probability) that the equation is fulfilled for k = 1. In practice, that value of k = 1 or a low value of k, will be very unlikely and, therefore, we can

expect a private key “d” very close to or equal in bits to the value of n as it happens in the practice. In other words, it will be computationally difficult to guess the value of the private key d, since finding a number within a 1024-bit body means an intractable computation time, with an average of 21023 attempts [35]. Forcing then the public key e to being a small value, less than 20 bits within a body of 1024 bits or greater guarantees that the private key d is a very large value and, therefore very secure since it is computationally intractable to find it by brute force.

In our model, we will choose value “e” from Fermat’s numbers less than $\sqrt{\Phi\,(n)}$, and since all Fermat’s numbers are prime numbers we won’t necessarily need to check if gcd(e, ϕ(n)) = 1 which saves time. This relatively small value of “e” forces the private key “d” to have a size similar to the modulus n and makes a brute force attack impractical. For example the value 65537 a known prime number as Fermat number (F4) was used to create the SSL certificates **See Figure 3**.

$$[\text{Fermat's prime}]\ Fn = 2^{2^n} \tag{9}$$

In addition to the advantages mentioned, Fermat primes such as F4 have a significant feature that is worth highlighting. The binary representation of F4 has only two one bits equal to 65, 537_{10} = 10000000000000001_2 = 010001_{16}. This fact has great utility in exponentiation computational efficiency.

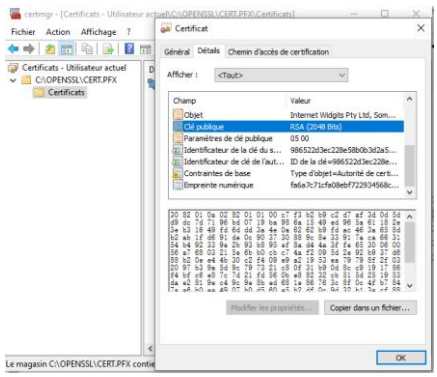


Figure 3: RSA2048 SSL Key Certificate.

5.9. Paired private and public keys

When an RSA key is generated, Euler’s thotient function Φ (N) is used as a trap to calculate the private key "d" knowing the public key "e". Since (e, Φ(N)) = 1, it is guaranteed that the inverse “d” exists and that it is the only inverse of “e” in that field Φ(N). The encryption is done afterward in the public body N so that anyone can use it. And in said body N it is no longer satisfied that the only inverse of the public key “e” is the private key “d”. There is at least one other value other than a “d” that allows deciphering what is encrypted with the public key.

These keys are called paired private keys. It has been said that an asymmetric cipher system has a single public key, and therefore also a single private key. For the RSA cryptosystem, this has turned out to be false. An example will better illustrate this.

Example: Let us take an RSA key with P = 37, Q = 57, e = 13, and d = 1861

$$N = P\ Q = 2109$$

$$\Phi(N) = (P - 1)\cdot(Q - 1) = 36 \cdot 56 = 2016$$

We will encrypt the Message 1001 with the public key (N, e) = (2109, 13) and then we will get.

$$\text{Cipher−text: } C = 1001^{13} \bmod 2109 = 1421$$

$$\text{Plain−text: } M = 1421^{1861} \bmod 2109 = 1001$$

We naturally get the original message. Let us now use the numbers 349, 853 and 1357 as if they were the private key d:

$$d = 349,\ M = 1421^{349} \bmod 2109 = 1001$$

$$d = 853,\ M = 1421^{853} \bmod 2109 = 1001$$

$$d = 1357,\ M = 1421^{1357} \bmod 2109 = 1001$$

That is, for the ciphered body N = 2109 with public key e = 13, the numbers 349, 853, and 1357 are paired private keys d_0 that fulfill my function more than the private key d. Every RSA key will have at least one matching private key. The number of even private keys depends strongly on the primes P and Q.

5.10. Unencryptable messages

One of the security vulnerabilities are non-recommended keys that either do not encrypt the information to protect or do so in a predictable way. For example, in the symmetric algorithm DES (Data Encryption Standard), there were weak or semi-weak keys, which did not satisfy the single-digit principle enunciated by Shannon; and gave solutions known as false solutions. Something similar happens in RSA, where there are unencryptable messages, or rather, unencryptable numbers.

$$0^{key} \bmod N = 0 \text{ and } 1^{key} \bmod N = 1 \tag{10}$$

Another value that is transmitted in the clear is (n - 1), since:

$$(N-1)^{key} \bmod N = (N-1) \tag{11}$$

Example: Consider the RSA key with n = 17 ·29 = 493 and e = 11, we have:

$$\begin{cases} 0^{11} mod\ 493 = 0 \\ 1^{11} mod\ 493 = 1 \\ 492^{11} mod\ 493 = 492 \end{cases}$$

In addition to these three numbers, in RSA there will always be another 6 numbers that are not encrypted.on our example:

$$\begin{cases} 86^{11} mod\ 493 = 86 \\ 203^{11} mod\ 493 = 203 \\ 204^{11} mod\ 493 = 204 \\ 289^{11} mod\ 493 = 289 \\ 290^{11} mod\ 493 = 290 \\ 407^{11} mod\ 493 = 407 \end{cases}$$

To locate these unencryptable numbers requires a brute force encryption attack on the space of the primes p and q, to verify the values of X that yield the following inequalities:

$$X^e \pmod P = X \text{ and } X^e \pmod Q = X \text{ for } 1 < X < Q - 1,\ P\text{-}1$$

Keys of 1024 bits or more make calculations within the primes p and q computationally intractable if each has at least 512 bits. Therefore, for those keys, it will not be possible to find the remaining unencryptable numbers. The equations to calculate the unencryptable numbers are shown below:

The number of unencryptable numbers σ_n within a field n is:

$$\sigma_n = [1 + (e-1, p-1)][1 + (e-1, q-1)] \tag{12}$$

The unencryptable numbers will be:

$$N = [q\ (\text{inv}\ (q, p))\ \cdot Np + p\ (\text{inv}\ (p, q))\ \cdot Nq] \bmod n \quad (13)$$

Where:

Np are the solutions of $N^e \bmod p = N$ and Nq are the solutions of $N^e \bmod q = N$

As can be seen, the only complicated calculation that occurs is in the last two equations, which means attacking by brute force all values of N candidates to be non-cipherable numbers, with

$$1 < N < (P - 1) \text{ for prime P and } 1 < N < (Q - 1) \text{ for prime Q}$$

6. Optimised RSA algorithm

For our modified RSA, we drew the following conclusions concerning the sizes of the operands:

- Carefully select the prime numbers p and q such that factoring them would be computationally unfeasible. As a rule of thumb, the bit lengths of these primes should be roughly comparable. For example, if the number n is 1024 bits in length, then the appropriate sizes for p and q are approximately 512 bits.
- The exponent is typically small in order to maximize the power of the exponentiation.

6.1. RSA Modified

- Generate Large Prime numbers p and q generate truly unpredictable random numbers list using LCG **Section 5.1**, then tests for primality we used Rabin-Miller probabilistic test **Section 5.2**.
- Calculate modulus **N = P ·Q**
- Calculate X to replace N. **X = N − (P + Q)**

The key length, which is commonly stated in bits, determines its size. We recommended the Karatsuba algorithm **Section 5.3** for N computation. The Karatsuba approach begins to pay off as n-digits increase, as it can multiply hundreds of digits quicker than the standard technique.

- Calculate Euler's totient function, which is defined as: **Φ(N) = X + 1**

Reduced multiplication of (P-1) (Q-1) of O (n^2) computing time to addition operation in O (n) computing time.

- Calculate exponent e as a fermat number **See Equation (9)** such that:

$$e < \sqrt{\Phi(N)}$$

- Compute private key **d·e mod Φ(N) = 1**

In practice, we applied Extended Euclidean algorithm **Section 5.4** for fast computation of this value. d bits should be close to N-bits so that it would not be easy to get any other paired private keys d_0 for security purposes.

- Generated Keys: **Public Key = (e, N) and Private Key = (d, N)**
- Encryption Cipher-Text **$C = M^e \pmod N$**

In practice, we used Fast Exponentiation Algorithm [4] for fast computation. $M \in [2, n - 2]$ as in **Equation [10] [11]** and M not an unencryptable numbers See **Equation [12]**.

- Decryption Plain-Text message **$M = C^d \pmod N$**

We proposed to use CRT **Section 5.6** and Fermat's Little Theorem 5.6 for fast computation and monitoring security.

6.2. RSA Modified Example

We begin by walking over on how to generate RSA encryption and decryption keys. After that, we'll go through a basic example to see how encryption and decryption work in practice.

• Two prime numbers “P” and “Q” are chosen. At present, such numbers must be of the order of 1024 bits, at least. For our example, Let us use a 16-bits prime for illustration

$$n\_digits = 16 \log 2 = 5$$

Using Equation (5) we should generate 5 digits prime numbers. Using our LCG approach **Section 5.1 Equation (1)**

$$X_0 = 2^{n\_bits-1} + 1 = 2^{15} + 1$$

$$P_1 = 4 \cdot 10^{n\_digits/2} + 1 = 4 \cdot k + 1$$

$$P_2 = 2^{n\_bits-1} + 7 = 2^{15} + 7$$

$$m = 2^{n\_bits+1} + k = 2^{17} + 29$$

where P_1 is variable, P_2 is an odd variable number, and m is the constant nearest prime. Running the Python code on [5] we get:

Real Primes: [2357, 23321, 83437, 33967, 118681, 96223, 44927, 129221, 35531, 67699] in 0.0008182525634765625 seconds.

We choose P = 23321 and Q = 67699.

• Calculate the modulus N,

$$N = P\ Q = 23321 \cdot 67699$$

$$N = karatsuba\ (23321, 67699) = 1578808379$$

$$X = N - (P + Q) = 1578717359$$

• We calculate the value of Φ (n).

$$\Phi(n) = X + 1 = 1578717360$$

$$\sqrt{\Phi(n)} = 39733$$

• A fermat's value of “e < 39733” is chosen. We will take Fermat number three.

$$e = F_3 = 2^{2^3} + 1 = 257 < 39733$$

, since all Fermat’s numbers are primes therefore gcd (257, 1578717360) = 1.

• Calculate d, the multiplicative inverse of module n.

Since gcd (e, Φ(n)) = 1, we will apply Extended Euclidian algorithm [1] to express 1 as a linear combination of integer u and v such that:

$$e \cdot u + \varphi(n) \cdot v = 1 \quad (14)$$

$$1 = 14 - 13$$

$$1 = 14 - (27 - 14)$$

$$1 = 2 \cdot 14 - 27$$

$$1 = 2 \cdot (257 - 9 \cdot 27) - 27$$

$$1 = 2 \cdot 257 - 19 \cdot 27$$

$$1 = 2 \cdot 257 - 19(1578717360 - 257 \cdot 6142869)$$

$$1 = 257(2 + 19 \cdot 6142869) + 1578717360 \cdot -19$$

$$1 = 257 \cdot 116714513 + 1578717360 \cdot (-19) \quad (15)$$

Equation [14] into [15] we get u = 116714513 and v = −19.
With the multiplicative inverse modulo n

$$d = 116714513$$

- The Message M Encryption $C = M^e \bmod N$.

For our illustration let us encrypt a message M = "123456":

$$C = 123456^{257} \bmod 1578808379$$

Applying **Algorithm [3]**
Step 1: First compute the binary representation of the exponent "257".
$257_{10} = 100000001_2$
Step 2: Read the binary representation of the exponent from left to right.
$100000001_2 = b_0b_1b_2b_3b_4b_5b_6b_7b_8$
Step 3: For every subsequent $b_i = 1$ apply square and multiplication while every subsequent $b_i = 0$ we apply the square operation only.
Step 4: $\mathbf{x}\ \overrightarrow{\boldsymbol{Sqr}}\ \mathbf{x^2}\ \overrightarrow{\boldsymbol{Sqr}}\ \mathbf{x^4}\ \overrightarrow{\boldsymbol{Sqr}}\ \mathbf{x^8}\ \overrightarrow{\boldsymbol{Sqr}}\ \mathbf{x^{16}}\ \overrightarrow{\boldsymbol{Sqr}}\ \mathbf{x^{32}}\ \overrightarrow{\boldsymbol{Sqr}}\ \mathbf{x^{64}}\ \overrightarrow{\boldsymbol{Sqr}}\ \mathbf{x^{128}}\ \overrightarrow{\boldsymbol{Sqr}}\ \mathbf{x^{256}}\ \overrightarrow{\boldsymbol{Mul}}\ \mathbf{x^{257}}$

Table 1: M^e (mod N) calculations.

$(257)_2$		
$b_0 = 1$	$1^2 \cdot 123456 \bmod 1578808379$	123456
$b_1 = 0$	$123456^2 \bmod 1578808379$	1032108525
$b_2 = 0$	$1032108525^2 \bmod 1578808379$	565223769
$b_3 = 0$	$565223769^2 \bmod 1578808379$	925099202
$b_4 = 0$	$925099202^2 \bmod 1578808379$	1485289910
$b_5 = 0$	$1485289910^2 \bmod 1578808379$	808794854
$b_6 = 0$	$808794854^2 \bmod 1578808379$	312569668
$b_7 = 0$	$312569668^2 \bmod 1578808379$	926477909
$b_8 = 1$	$926477909^2 \cdot 123456 \bmod 1578808379$	1080549941

We have to perform 8 squaring and 1 multiplication only. We have, therefore:

$$x^{257} = \chi^{100000001_2} = \chi^{b_0b_1b_2b_3b_4b_5b_6b_7b_8}$$

Using the fast exponentiation **Algorithm [4],** to loop through the bits of the exponent, starting at b_0 to b_8.
This completes the encryption process **See Table 1**, give Cipher-text:

$$C = 123456^{257} \pmod{1578808379} = 1080549941$$

- For Decryption $\mathbf{M = C^d \pmod N = 1080549941^{116714513} \pmod{1578808379}}$

Since both P and Q divide N, we can write the above equation as:

$$M_1 = 1080549941^{116714513} \pmod{23321}$$

$$M_2 = 1080549941^{116714513} \pmod{67699}$$

We now reduce the bases by their respective modules:

$$M_1 = 18048^{116714513} \pmod{23321}$$

$$M_2 = 6202^{116714513} \pmod{67699}$$

The exponents are also reduced:

$$116714513 = (23321 - 1) \cdot 5004 + 21233$$

$$116714513 = (67699 - 1) \cdot 1724 + 3161$$

Solving the first exponent:

$$M_1 = 18048^{5004^{(23321-1)}} \cdot 18048^{21233} \bmod 23321$$

Applying Fermat's Little Theorem [4], Fermat's **Equation (7)** we get:

$$M_1 = 18048^{21233} \bmod 23321 = 6851 \bmod 23321$$

Solving the second exponent:

$$M_2 = 6202^{1724^{(67699-1)}} \cdot 6202^{3161} \bmod 67699$$

Applying Fermat's Little Theorem [4], Fermat's **Equation (7)** we get:

$$M_2 = 6202^{3161} \bmod 67699 = 55757 \bmod 67699$$

We thus have the system of congruencies:

$$\begin{cases} M \equiv 6851 \; mod \; 23321 \\ M \equiv 55757 \; mod \; 67699 \end{cases}$$

Solving the above system using the CRT, we get:

$$\begin{cases} 6851 \cdot 67699 \cdot 9075 \equiv 6851 \bmod 23321 \\ 55757 \cdot 23321 \cdot 41355 \equiv 55757 \; mod \; 67699 \end{cases}$$

Compute:

$$(6851 \cdot 67699 \cdot 9075) + (55757 \cdot 23321 \cdot 41355) = 57983316650610 \bmod (23321 \cdot 67699) = 123456$$

Plain-text message M = 123456 as expected.

7. Results and discussion

7.1. Prime Generation Comparison

Table 2: Prime Generation Comparison.

KEY SIZE	LCG Run-time/sec	Random odd number Run-time/sec	LCG Generated primes	Random odd Generated
16 bits	0.00289559364	0.00202369054	9 primes / 10 odd numbers	1 prime
32 bits	0.00123167037	0.00262594223	5 primes/10 odd numbers	1 prime
512 bits	0.50483727455	0.28248929977	2 primes / 150 odd numbers	1 prime
1024 bits	7.26790904998	7.31390762329	7 primes / 200 odd numbers	1 prime
2048 bits	16.5216593742	17.60818386077	1 prime / 200 odd numbers	1 prime

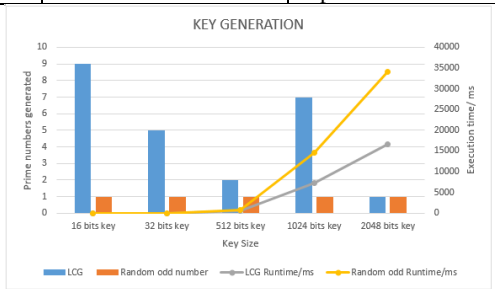


Figure 4: LCG Prime Generation versus Random Odd Number Prime Generation.

We also want to look into the temporal complexity of each method. In this situation, we're interested in the algorithm's efficiency, or how long it takes the function to generate at least two prime integers. As shown in **Table 2**, traditional prime generation always yields a single prime number by randomly selecting an odd number and testing for primality, however, our suggested LCG prime

generation method yields a list of primes in a short time. **Figure 4** illustrates a graph.

7.2. Key Generation

Consider n as the number of bits of P or Q both prime and N as the product of P and Q.

7.2.1. Modulus computation

The time complexity to compute the modulus N product of two big primes P and Q of the same bit size is $(log_2^p)^2$ using big O-natation that is O (n^2). By using the Karatsuba algorithm **Section [5.3]** the time complexity goes down to O ($nlog_2^3$) approximately O ($n^{1.58}$).

7.2.2. Thotient computation

The time complexity to compute Φ (N) product of two large integers (P − 1) and (Q − 1) of the same bit size is $(log_2^p)^2$ using big O-natation that is O (n^2), by substituting X = N − (P + Q), Φ (N) = X + 1 the time complexity goes down to O (n)

7.3. Encryption

7.3.1. Making the exponent smaller vs Classical RSA

In making an algorithm faster, we can actually use a small exponent and still RSA will be secure, In **Table 3** changing the public key exponent e to the binary representation we can tell the number of operations that are required for the square multiply algorithm using exponentiation, fast encryption is possible using this small exponent and RSA is still secure for short.

Table 3: Small e operations.

Public Key "e"	"e" in binary	Operations
$\mathbf{2^{2^0}}+1=3$	11_2	3
$\mathbf{2^{2^2}}+1=17$	10001_2	5
$\mathbf{2^{2^4}}+1=2^{16}+1$	10000000000000001_2	17
$\mathbf{2^{2^n}}+1$	$1(2^n-1\text{ zeros})1_2$	2^n+1

7.3.2. Cipher-text

$$C = M^e \pmod N$$

Normally e can be in range [1, Φ (N)] therefore the worst-case scenario e can also have n-bits.
Time complexity T: O ((log n^2) (log n)) = O (log n^3)
By applying fast modular exponentiation [4] and making e a small value in the range [1, $\sqrt{\Phi\ (N)}$].
Time complexity T: O (log $n^{2/2}$ log n) = O (log n^2).

7.4. Decryption

To compute $M = C^d \pmod N$ in general, d is a large integer almost the same size as N
Time complexity T: O (log n^2 log n) = O (log n^3)
By applying the Chinese Remainder Theorem (CRT) knowing P, and Q as factors of N we can compute M as:

$$M_1 = C^d \bmod p \text{ and } M_2 = C^d \bmod Q$$

$$M_1 = C^{d_1} \bmod P \text{ and } M_2 = C^{d_2} \bmod Q$$

Where: $d_1 = d \bmod (P - 1)$ and $d_2 = d \bmod (Q - 1)$

Lastly compute M as:

$$M = M_1 + P\,[(M_2 - M_1)\,P^{-1} \bmod Q]$$

The public exponent "e" can be smaller in this case, but "d" must be as big as "N" now, thus we applied the Chinese remainder theorem (CRT) to accelerate decryption. This technique lowers n_bits modular exponentiation into two n/2_bits modular exponentiations plus the CRT steps described above, while P^{-1} can be precomputed and saved. The Chinese remainder theorem, according to **Table 4**, is substantially faster at the expense of system parameters and memory, and it does not require any modular inversion implementation, saving development costs and memory. As illustrated in **Figure 6, 7 and 8** both the CRT and decreasing the exponent need less processing time than the standard RSA. Because it adds randomness to the cryptography method, RSA's improved performance increases speed and security.

Table 4: CRT system parameters memory cost.

Register Names	Bits	Number of registers
N,M	n	2
M_1, M_2, P, Q	n/2	4
P-inverse	n/2	1
Subtotal		9 n/2
P-inverse	n/2	1
Accumulator		
v	n/2	1
Subtotal		n/2
Total memory cost		**5n**

7.5. Comparison of Total Time Complexity

The time complexity study of the enhanced RSA method and the traditional RSA algorithm will now be described using big O notation. For time complexity operations, we have the following features, as described by [36] and [37]:

Sum (x + y), the sum of two n-bit values has a time complexity of O (n).

Subtraction (x - y), is an addition of a negative integer, its asymptotic time complexity is O (n).

Multiplication (x * y), the product of two n-bit values, has a time complexity of O (n^2).

Division (x / y), is a multiplication of an inverted integer, its asymptotic time complexity is O (n^2).

Module (A mod N), given an n-bit A, the algorithm's time complexity is O (n^2).

Exponentiation in modules (A^B mod N) has a time complexity of O (n^3).

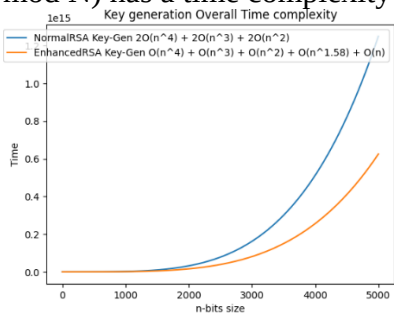


Figure 5: Key Generation Overall Time Complexity.

The Extended Euclid's Algorithm, Using the gcd (a, b) = gcd (b, a mod b) rule, Euclid's method computes the gcd of two integers a and b and yields the last value of "a" as gcd when b is zero. To examine the Extended Euclid's algorithm, assume a and b of n-bits, and a mod b < a/2 at each iteration. As a result, the algorithm will only make n recursive calls (since each division decreases one bit of

n). In addition, a division of complexity O (n^2) is done at every recursive step to extract the new argument b for the following iteration. When we add up the whole time, we obtain. O (n) ·O (n^2) = O (n^3)

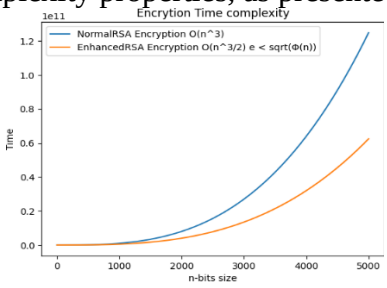
Multiplicative Inverse, Time complexity is O (n^3) due to the usage of the Extended Euclid method.

Primality Test, the N key is n bits long, while the prime integer p is (n/2) bits long. In practice, the primality test is performed using the probabilistic Miller-Rabin test, **Algorithm [5.2].** The most expensive operation in the Miller-Rabbin technique, given a number p of n bits, is modular exponentiation, which has an O (n^3) complexity. Furthermore, to ensure a high level of confidence, this test is repeated about ln (p) / 2 [38], resulting in a complexity of O(n). As a result, the final complexity is O (n^3) ·O (n) = O (n^4)

Table 5: Rsa Overall Time Complexity.

	Normal RSA	Enhanced RSA
Prime Generation		
Prime Generation	P in O(n4) Q in O(n4)	P and Q in O(n4)
Key Generation		
Modulus N = P ·Q	$O(n^2)$	Karatsuba $O(n^{1.58})$
Thotient Φ(n)	$O(n^2)$	O(n)
e	GCD(e, Φ(n)) = 1, $O(n^3)$	e is a Fermat's number $O(n^2)$
d · e ≡ 1 mod Φ(n))	$O(n^3)$	$O(n^3)$
Key Gen **Figure 5**	2 $O(n^4)$ + 2 $O(n^3)$ + 2 $O(n^2)$	$O(n^4)$ + $O(n^3)$ + $O(n^2)$ + $O(n^{1.58})$ + O(n)
Encryption		
$C = M^e \pmod N$	$O(n^3)$	$O(n^3/2)$
Decryption		
$M = C^d \pmod N$	$O(n^3)$	2 ·O((n/2)^3)
Overall RSA		
Overall Time	2$O(n^4)$ + 4$O(n^3)$ + 2$O(n^2)$	$O(n^4)$ + 7/4 $O(n^3)$ + $O(n^2)$+ $O(n^{1.58})$ + O(n)

We can examine the overall RSA performance in contrast to the suggested RSA from all of the above time complexity properties, as presented in **Table 5**.



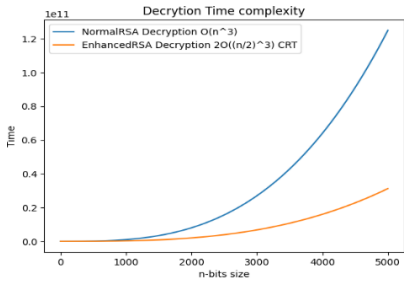


Figure 6: Encryption Overall Time Complexity. Figure 7: Decryption Overall Time Complexity.

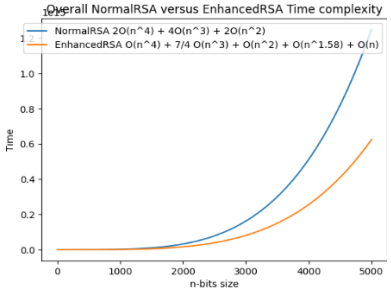


Figure 8: Standard RSA versus Proposed RSA.

8. Conclusion

This study proved helpful with a combination of the Chinese Remainder Theorem (CRT) algorithm, Fast exponentiation Algorithm, small exponent, and Linear Congruential Generator (LCG)

that has been proposed with the normal RSA algorithm for fast and secure communication for large data. We applied the proposed theorems, multiplication algorithms, and randomness techniques to redesign the execution of the RSA cryptosystem that improves speed on the RSA encryption side while the security of data was maintained and upgraded with the CRT on the decryption side. This research objective includes a detailed overview of the most successful classical cryptographic Technique RSA. The study proposed an RSA with the Chinese Remainder Theorem using a small value of the exponent, which helped enhance the cryptographic algorithm. Another contribution is the discussion on different multiplication algorithms and their pseudo-codes. Future work will discuss the practical implementation of the modified RSA algorithm's security attacks and threats. But unless we encounter a situation that puts our data in danger, we will continue to use what we have.

Acknowledgements

My heartfelt thanks go to my supervisor, Professor Shen Wei, and all the professors in the School of Information and Communication for their careful guidance. Their precepts and deeds will benefit me all my life. During the several years of cooperative research in the Department of Information and Communication of Zhejiang Sci-Tech University, I am grateful for the enthusiastic guidance and help of Professor Shen Wei. Thanks to my colleagues, classmates, the director of the School of Information and Communication, the dean, and the president of our school for directly or indirectly supporting this research work through scholarship, etc. This project was supported by the Zhejiang Sci-Tech University, and we would like to thank you for this.

References

[1] T. S. Obaid, Study a public key in rsa algorithm, European Journal of Engineering and Technology Research 5 (4) (2020) 395–398.

[2] S. A. Jaju, S. S. Chowhan, A modified rsa algorithm to enhance security for digital signature, in: 2015 international conference and workshop on computing and communication (IEMCON), IEEE, 2015, pp. 1–5.

[3] L. A. Koster, J. E. Meinardi, B. L. Kaptein, E. Van der Linden-Van der Zwaag, R. G. Nelissen, Two-year rsa migration results of symmetrical and asymmetrical tibial components in total knee arthroplasty: a randomized controlled trial, The Bone & Joint Journal 103 (5) (2021) 855–863.

[4] R. Imam, Q. M. Areeb, A. Alturki, F. Anwer, Systematic and critical review of rsa based public key cryptographic schemes: Past and present status, IEEE Access 9 (2021) 155949–155976.

[5] N. Moise, Rsa-public-key-codes, https://github.com/NMoise/RSA-public-key-codes (2023).

[6] S. Chandra, S. Paira, S. S. Alam, G. Sanyal, A comparative survey of symmetric and asymmetric key cryptography, in: 2014 international conference on electronics, communication and computational engineering (ICECCE), IEEE, 2014, pp. 83–93.

[7] J. N. Gaithuru, M. Bakhtiari, M. Salleh, A. M. Muteb, A comprehensive literature review of asymmetric key cryptography algorithms for establishment of the existing gap, in: 2015 9th Malaysian Software Engineering Conference (MySEC), IEEE, 2015, pp. 236–244.

[8] S. F. Mare, M. Vladutiu, L. Prodan, Secret data communication system using steganography, aes and rsa, in: 2011 IEEE 17th International Symposium for Design and Technology in Electronic Packaging (SIITME), IEEE, 2011, pp. 339–344.

[9] D. Boneh, H. Shacham, Fast variants of rsa, CryptoBytes 5 (1) (2002) 1–9.

[10] C. A. M. Paixao, D. L. Gazzoni Filho, An efficient variant of the rsa cryptosystem, in: Anais do V Simp´osio Brasileiro em Seguran¸ca da Informa¸c˜ao e de Sistemas Computacionais, SBC, 2005, pp. 14–27.

[11] S. Gupta, J. Sharma, A hybrid encryption algorithm based on rsa and diffie-hellman, in: 2012 IEEE International Conference on Computational Intelligence and Computing Research, IEEE, 2012, pp. 1–4.

[12] R. S. Dhakar, A. K. Gupta, P. Sharma, Modified rsa encryption algorithm (mrea), in: 2012 second international conference on advanced computing & communication technologies, IEEE, 2012, pp. 426–429.

[13] J. M. Ahmed, Z. M. Ali, The enhancement of computation technique by combining rsa and el-gamal cryptosystems, in: Proceedings of the 2011 International Conference on Electrical Engineering and Informatics, IEEE, 2011, pp. 1–5.

[14] V. S. Mahalle, A. K. Shahade, Enhancing the data security in cloud by implementing hybrid (rsa & aes) encryption algorithm, in: 2014 International Conference on Power, Automation and Communication (INPAC), IEEE, 2014, pp. 146–149.

[15] S. Arora, Enhancing cryptographic security using novel approach based on enhanced-rsa and elamal: Analysis and comparison, International Journal of Computer Applications 975 (2015) 8887.

[16] E. Jintcharadze, M. Iavich, Hybrid implementation of twofish, aes, elgamal and rsa cryptosystems, in: 2020 IEEE East-West Design & Test Symposium (EWDTS), IEEE, 2020, pp. 1–5.
[17] Z. Alamsyah, T. Mantoro, U. Adityawarman, M. A. Ayu, Combination rsa with one time pad for enhanced scheme of two-factor authentication, in: 2020 6th International Conference on Computing Engineering and Design (ICCED), IEEE, 2020, pp. 1–5.
[18] C. W. Chiou, Parallel implementation of the rsa public-key cryptosystem, International Journal of Computer Mathematics 48 (3-4) (1993) 153–155.
[19] M. A. Ayub, Z. A. Onik, S. Smith, Parallelized rsa algorithm: An analysis with performance evaluation using openmp library in high performance computing environment, in: 2019 22nd International Conference on Computer and Information Technology (ICCIT), IEEE, 2019, pp. 1–6.
[20] A. Rawat, K. Sehgal, A. Tiwari, A. Sharma, A. Joshi, A novel accelerated implementation of rsa using parallel processing, Journal of Discrete Mathematical Sciences and Cryptography 22 (2) (2019) 309–322.
[21] C.-H. Wu, J.-H. Hong, C.-W. Wu, Rsa cryptosystem design based on the chinese remainder theorem, in: Proceedings of the 2001 Asia and South Pacific Design automation conference, 2001, pp. 391–395.
[22] J. Bl¨omer, M. Otto, J.-P. Seifert, A new crt-rsa algorithm secure against bellcore attacks, in: Proceedings of the 10th ACM conference on Computer and communications security, 2003, pp. 311–320.
[23] K. Sony, D. Shaik, B. D. Sri, G. Anitha, Improvised asymmetric key encryption algorithm using matlab, IOSR Journal of Electronics and Communication Engineering 10 (2) (2015) 31–36.
[24] J.-J. Quisquater, C. Couvreur, Fast decipherment algorithm for rsa public-key cryptosystem, Electronics letters 21 (18) (1982) 905–907.
[25] P. Aiswarya, A. Raj, D. John, L. Martin, G. Sreenu, Binary rsa encryption algorithm, in: 2016 International Conference on Control, Instrumentation, Communication and Computational Technologies (ICCICCT), IEEE, 2016, pp. 178–181.
[26] J. Sahu, V. Singh, V. Sahu, A. Chopra, An enhanced version of rsa to increase the security, Journal of Network Communications and Emerging Technologies (JNCET) 7 (2017) 1–4.
[27] R. L. Rivest, A. Shamir, L. Adleman, A method for obtaining digital signatures and public-key cryptosystems, Communications of the ACM 21 (2) (1978) 120–126.
[28] M. Rouse, Digital signature network security, Search Security. Tech-Target, 2014.
[29] S. Nisha, M. Farik, Rsa public key cryptography algorithm, A Review. International Journal of Scientific and Technological Research 6 (2017) 187–191.
[30] T. E. Hull, A. R. Dobell, Random number generators, SIAM review 4 (3) (1962) 230–254.
[31] M. O. Rabin, Probabilistic algorithm for testing primality, Journal of number theory 12 (1) (1980) 128–138.
[32] L. C. C. Garcıa, Can sch¨onhage multiplication speed up the rsa decryption or encryption? Technische Universit¨at Darmstadt, 2007.
[33] C. Intila, B. Gerardo, R. Medina, A study of public key 'e'in rsa algorithm, in: IOP Conference Series: Materials Science and Engineering, Vol. 482, IOP Publishing, 2019, p. 012016.
[34] C. Thirumalai, Review on the memory efficient rsa variants, International Journal of Pharmacy and Technology 8 (4) (2016) 4907–4916.
[35] K. Moriarty, B. Kaliski, J. Jonsson, A. Rusch, Pkcs# 1: Rsa cryptography specifications version 2.2, Tech. rep. (2016).
[36] U. V. V. S. Dasgupta, C. H. Papadimitriou, Algorithms, McGraw-Hill Education, 2006.
[37] W. Stallings, G. Bressan, A. Barbosa, Criptografia e seguran¸ca de redes, Pearson Educacion, 2008.