Enhancing RSA Security Through Random Key Generation in C Language

MAJOR PROJECT REPORT

SEMESTER 1

FOUR-YEAR UNDERGRADUATE PROGRAM

(DESIGN YOUR DEGREE)

SUBMITTED TO

UNIVERSITY OF JAMMU, JAMMU

SUBMITTED BY

NAME	ROLL N0.
Anushka Sharma	DYD-24-05
Bhaviya Koul	DYD-24-08
Harshita	DYD-24-11
Himanshi Amla	DYD-24-12
Krishna Sharma	DYD-24-17

UNDER THE MENTORSHIP OF

Prof. K.S. Charak

Dr. Jatinder Manhas

Dr. Sunil Kumar

University of Jammu, Jammu

SUBMITTED ON: 29th JANUARY, 2025

CERTIFICATE

The work encompassed in this report titled **"*Enhancing RSA Security Through Random Key Generation in C Language*"** has been done by Anushka Sharma, Bhaviya Koul, Harshita, Himanshi Amla, Krishna Sharma as a Major Project for Semester 1. This work was carried out under the guidance of Prof. K.S. Charak, Dr. Jatinder Manhas and Dr. Sunil Kumar for the partial fulfilment for the award of the Design Your Degree, Four Year Undergraduate Program, University of Jammu. This project report has not been submitted anywhere for the award of any degree, diploma.

Students:

Anushka Sharma

Bhaviya Koul

Harshita

Himanshi Amla

Krishna Sharma

Signature of Mentors:

Prof. K.S. Charak Dr. Jatinder Manhas Dr. Sunil Kumar

Prof. Alka Sharma

Director, SIIEDC, University of Jammu

ACKNOWLEDGEMENT

We, the students of the "Design Your Degree" (Four-Year Undergraduate Program) at University of Jammu, are deeply grateful to all the individuals and organizations whose support, guidance, and encouragement have been instrumental in the successful completion of our project, titled, ***"Enhancing RSA Security Through Random Key Generation in C Language."***

First and foremost, we would like to extend to our sincere gratitude to our esteemed mentors, **Prof. K.S. Charak**, **Dr. Jatinder Manhas**, and **Dr. Sunil Kumar**, whose invaluable guidance, constant support, and expertise have been a source of inspiration throughout the duration of this project.

We would also like to express our heartfelt thanks to the faculty members of the **University of Jammu** for providing us with a comprehensive academic foundation and for fostering an environment that encouraged intellectual curiosity. Their insights and knowledge were instrumental in shaping the directions of the research.

Lastly, we are deeply grateful to our families and friends for their constant support, understanding, and understanding.

ABSTRACT

The **Rivest-Shamir-Adleman (RSA) algorithm** is one of the most widely used cryptographic systems, ensuring secure communication in the digital world. This project focuses on the mathematical foundation and practical implementation of RSA encryption, emphasizing key generation, prime number randomness, and statistical analysis.

Key generation as part of the process is the very first aspect of RSA and is based upon mathematical principles that use two large prime numbers. Their product forms the modulus, while the public and private keys are generated. It is through these processes that a secure encryption system can be established.

In the practical implementation, a C language program was generated to create RSA key generation in which the prime numbers are randomly selected from a specific range of numbers. Such a program uses efficient algorithms which explains why it is essential to maintain the robustness of the RSA encryption and the key generation process.

In addition, the project involves statistical analysis of the last digits of the generated prime numbers to check their distribution. This analysis provides insight into the randomness and uniformity of prime number selection, which contributes to the reliability of RSA encryption.

It emphasizes randomness and statistical properties in key generation for cryptographic security. This project shows the interaction between theoretical ideas and practical implementation in the RSA algorithm, giving much insight into further research and applications in secure communication.

TABLE OF CONTENT

CHAPTER 1

INTRODUCTION

RSA (Rivest–Shamir–Adleman) is a well-known asymmetric cryptography algorithm used for secure data transmission. Developed in 1977 by Ron Rivest, Adi Shamir, and Leonard Adleman [1]. RSA's security is based on the computational difficulty of factoring large composite numbers. The algorithm involves modular exponentiation with extremely large numbers, typically ranging from 1024 to 2048 bits, for both encryption and decryption.

This project focuses on exploring the mathematical principles behind RSA and enhancing RSA security through random key generation in C language.

1.1. CRYPTOGRAPHY: HISTORY

The first known evidence of the use of cryptography (in some form) was found in an inscription carved around 1900 BC, in the main chamber of the tomb of the nobleman **Khnumhotep II, in Egypt.** The scribe used some unusual hieroglyphic symbols here and there in place of more ordinary ones. The purpose was not to hide the message but perhaps to change its form in a way which would make it appear dignified. Though the inscription was not a form of secret writing but incorporated some sort of transformation of the original text and is the oldest known text to do so. Evidence of some use of cryptography has been seen in most major early civilizations. "**Arthshashtra**", a classic work on statecraft written by Kautalya, describes the espionage service in India and mentions giving assignments to spies in "secret writing" - sounds like an ancient version of James Bond.

Fast forwarding to around 100 BC, Julius Caesar was known to use a form of encryption to convey secret messages to his army generals posted in the war front. This substitution cipher, known as Caesar cipher, is perhaps the most mentioned historic cipher in academic literature. (A cipher is an algorithm used for encryption or decryption.) In a substitution cipher, each character of the plain text (plain text is the message which has to be encrypted) is substituted by another character to form the cipher text (cipher text is the encrypted message). The variant used by Caesar was a shift by 3 ciphers. Each character was shifted by 3 places, so the character 'A' was replaced by 'D', 'B' was replaced by 'E', and so on. The characters would wrap around at the end, so 'X' would be replaced by 'A'.

It is easy to see that such ciphers depend on the secrecy of the system and not on the encryption key. Once the system is known, these encrypted messages can easily be decrypted. In fact, substitution ciphers can be broken by using the frequency of letters in the language.

During the 16th century, Vigenere designed a cipher that was supposedly the first cipher which used an encryption key. In one of his ciphers, the encryption key was repeated multiple times spanning the entire message, and then the cipher text was produced by adding the message character with the key character modulo 26. In one of his ciphers, the encryption key was repeated multiple times spanning the entire message, and then the cipher text was produced by adding the message character with the key character modulo 26. (Modulo, or mod, is a mathematical expression in which you calculate the remainder of a division when one number is divided by another.) As with the Caesar cipher, Vigenere's cipher can also easily be broken; however, Vigenere's cipher brought the very idea of introducing encryption keys into the picture, though it was poorly executed. Comparing this to Caesar cipher, the secrecy of the message depends on the secrecy of the encryption key, rather than the secrecy of the system.

At the start of the 19th century when everything became electric, Hebern designed an electro-mechanical contraption which was called the Hebern rotor machine. It uses a single rotor, in which the secret key is embedded in a rotating disc. The key encoded a substitution table and each key press from the keyboard resulted in the output of cipher text. This also rotated the disc by one notch and a different table would then be used for the next plain text character. This was again broken by using letter frequencies.

The Engima machine was invented by German engineer Arthur Scherbius at the end of World War I and was heavily used by the German forces during the Second World War. The Enigma machine used 3 or 4 or even more rotors. The rotors rotate at different rates as you type on the keyboard and output appropriate letters of cipher text. In this case the key was the initial setting of the rotors.

The Enigma machine's cipher was eventually broken by Poland and the technology was later transferred to the British cryptographers who designed a means for obtaining the daily key.

Up to the Second World War, most of the work on cryptography was for military purposes, usually used to hide secret military information. However, cryptography attracted commercial attention post-war, with businesses trying to secure their data from competitors.

In the early 1970's, IBM realized that their customers were demanding some form of encryption, so they formed a "crypto group" headed by Horst-Feistel. They designed a cipher called Lucifer. In 1973, the Nation Bureau of Standards (now called NIST) in the US put out a request for proposals for a block cipher which would become a national standard. They had obviously realized that they were buying a lot of commercial products without any good crypto support. Lucifer was eventually accepted and was called DES or the Data Encryption Standard. In 1997, and in the following years, DES was broken by an exhaustive search attack. The main problem with DES was the small size of the encryption key. As computing power increased it became easy to brute force all different combinations of the key to obtain a possible plain text message.

In 1997, NIST again put out a request for proposal for a new block cipher. It received 50 submissions. In 2000, it accepted Rijndael and christened it as AES or the Advanced Encryption Standard. Today AES is a widely accepted standard used for symmetric encryption.

In recent times, advancements in Quantum computers have led us to think about Post Quantum Cryptography. In 2016 NIST declared a "call for proposals" seeking public help in designing quantum resistant algorithms which could help us "withstand the assault of a future quantum computer". In 2020 NIST announced four finalists for the same [2].

To conclude, history teaches us:

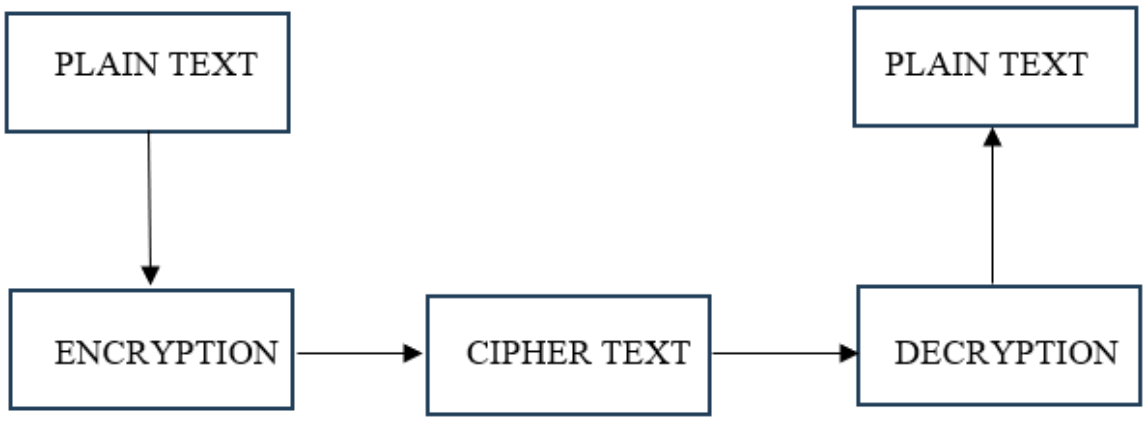
1. The secrecy of your message should always depend on the secrecy of the key, and not on the secrecy of the encryption system. (This is known as Kerckhoffs's principle.)
2. Related to the above, always use ciphers which have been publicly reviewed and have been established as a standard. Using "secret crypto" is bad, because just like the Caesar cipher, once the system is known, all messages can be decrypted. For example, if your key is compromised, an attacker could access your messages; however, if the attacker can compromise the crypto system itself, they can obtain the plain text of every message (not just for a single person) encrypted by that system.

1.2. CRYPTOGRAPHY: DEFINITION

Cryptography is a technique of securing communication by converting plain text into cipher text. It involves various algorithms and protocols to ensure data confidentially, integrity, authentication, and non-repudiation.

It's a technique where the communication is secured through the use of the codes so that only those persons for whom the information is intended can understand and process it. Thus, preventing the unauthorized access to information. The prefix "Crypt" means "hidden" and the suffix "graphy" means "writing" [3]. In Cryptography, the techniques that are used to protect information are obtained from mathematical concepts and a set of rule-based calculations known as algorithms to convert messages in ways that make it hard to decode them.

These algorithms are used for cryptographic Key Generation, digital signing and verification to protect data privacy, web browsing on the internet and to protect confidential transactions such as credit card and debit card transaction.



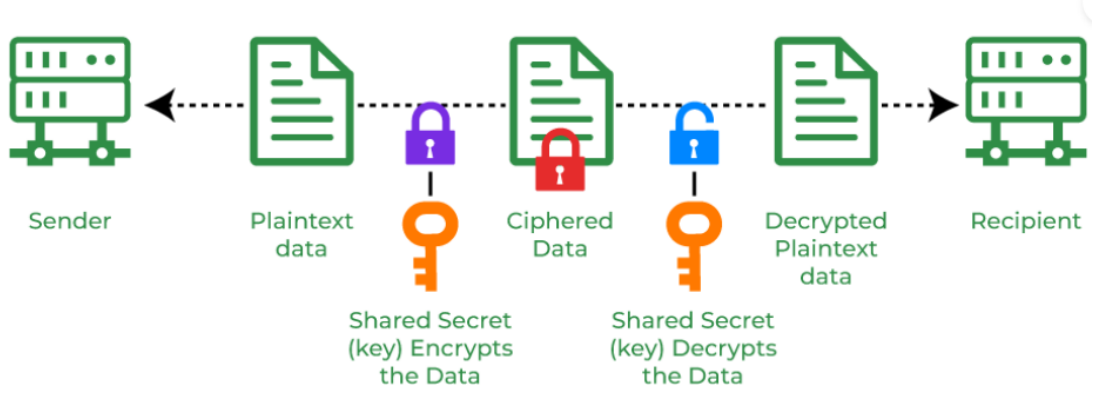
1.3. FEATURES OF CRYPTOGRAPHY

- **Confidentiality:** Information can only be accessed by the person for whom it is intended and no other person except him can access it.
- **Integrity:** Information cannot be modified in storage or transition between sender and intended receiver without any addition to information being detected.
- **Non-repudiation:** The creator/sender of information cannot deny his intention to send information at a later stage.
- **Authentication:** The identities of the sender and receiver are confirmed. As well destination/origin of the information is confirmed.
- **Interoperability:** Cryptography allows for secure communication between different systems and platforms.
- **Adaptability:** Cryptography continuously evolves to stay ahead of security threats and technological advancements [3].

1.4. TYPES OF ENCRYPTIONS

1. Symmetric Encryption

Symmetric encryption is an encryption method that uses a single key to encrypt and decrypt data [4]. Symmetric encryption is generally less secure than asymmetric encryption, it's often considered more efficient because it requires less processing power.



https://media.geeksforgeeks.org/wp-content/uploads/20240409123909/Private-Key-Encryption-(1).png

2. Hash Functions

A **hash function** converts strings of different length into fixed-length strings known as hash values or digests [5]. A hash value with a fixed length is calculated as per the plain text which makes it impossible for the contents of plain text to be recovered.

3. Asymmetric Encryption

Asymmetric encryption is an encryption method that uses two different keys—a public key and a private key—to encrypt and decrypt data [6]. Even if the public key is known by everyone the intended receiver can only decode it because he alone knows his private key.

https://media.geeksforgeeks.org/wp-content/uploads/20240409125853/Asymmetric-.webp

1.5. APPLICATIONS OF CRYPTOGRAPHY

Cryptography ensures messages are safe when people send them on internet, and it's used in many different areas. Here are some notable applications:

- **Secure Communication**: Cryptography is widely used to secure communication channels, such as emails, instant messaging, and voice-over-IP (VoIP) calls. Encrypting the transmitted data prevents eavesdropping and ensures the confidentiality and integrity of the information exchanged.

- **Data Protection**: It is vital in safeguarding sensitive data stored on computers, servers, and other digital devices. It encrypts files, directories, and entire storage volumes, thereby preventing unauthorized access and data breaches.

- **Online Transactions**: In e-commerce and online banking, cryptography is employed to secure transactions conducted over the Internet. It enables the encryption of financial data, including credit card details and banking credentials, protecting them from interception by cybercriminals.

- **Authentication and Digital Signatures**: Cryptography facilitates authentication and verification mechanisms, allowing users to prove their identity and verify the integrity of digital documents. Digital signatures, created using cryptographic techniques, provide non-repudiation and ensure that the signer cannot deny their involvement in the transaction [7].

1.6. ADVANTAGES OF CRYPTOGRAPHY

- **Access Control:** Cryptography can be used for access control to ensure that only parties with the proper permissions have access to a resource. Only those with the correct decryption key can access the resource thanks to encryption.
- **Secure Communication:** For secure online communication, cryptography is crucial. It offers secure mechanisms for transmitting private information like passwords, bank account numbers, and other sensitive data over the Internet.
- **Protection against attacks:** Cryptography aids in the defense against various types of assaults, including replay and man-in-the-middle attacks . It offers strategies for spotting and stopping these assaults.

CHAPTER 2

RSA ALGORITHM

2.1. RSA ALGORITHM: INTRODUCTION

The RSA algorithm (Rivest-Shamir-Adleman) is the basis of a cryptosystem, a suite of cryptographic algorithms that are used for specific security services or purposes, which enables public key encryption and is widely used to secure sensitive data, particularly when it is being sent over an insecure network such as the internet.

RSA was first publicly described in 1977 by Ron Rivest, Adi Shamir and Leonard Adleman of the Massachusetts Institute of Technology, though the 1973 creation of a public key algorithm by British mathematician Clifford Cocks was kept classified by the U.K.'s GCHQ until 1997 [8].

The growing domain of computer networks required a solution to secure digital communication. RSA was initially developed in 1977 as one such solution. The primary focus of RSA was to allow data to be securely transmitted over unsecured networks, specifically to enable private communications over the internet and other electronic system.

In traditional cryptographic systems, secure key distribution was a challenge. It required both parties to share a secret key before sending or receiving a message. With RSA, public-key Cryptography helps users to share their public key openly, while keeping their private key secret. This solved the problem of key distribution and allowed users to communicate securely without prior key sharing.

2.2. HOW DOES RSA WORKS?

The option to encrypt either with the private key or the public key provides a multitude of services to RSA users. If the public is used for encryption, the private key must be used to decrypt data. This is perfect for sending sensitive information across a network or internet connection, where the recipient of the data sends the data sender their public key [9]. The sender of the data sender then encrypts the sensitive information with the public key and sends it to the recipient. Since the public key encrypted the data, only the owner of the private key can decrypt the sensitive data. Thus, only the intended recipient of the data can decrypt it, even if

the data were taken in transit. The other method of asymmetric encryption with RSA is encrypting a message with the private key. In this example, the sender of the data encrypts the data with their private key and sends encrypted data and their public key along to the recipient of the data. The recipient of the data can then decrypt the data with the sender's public key, thus verifying the sender is who they say they are. With this method, the data could be stolen and read in transit, but the true purpose of this type of encryption is to prove the identity of the sender. If the data were stolen and modified in transit, the public key would not be able to decrypt the new message and so the recipient would know the data had been modified in transit.

The technical detail of RSA works on the idea that it is easy to generate a number by multiplying two sufficiently large prime numbers together but factorizing that number back into the original prime numbers, one of which is a product of two large prime numbers, one of which is a product of two large. Both use the same tow prime numbers to compute their value.

2.3. THE KEY IDEAS BEHIND RSA ALGORITHM

The idea of RSA is based on the fact that it is difficult to factorize a large integer. The public key (n, e), where n and **e** are publicly known, while the Private Key is (n, d). Since only the receiver knows the value of **d**, only they can decrypt the message. We know that $(d*e) \equiv 1 \bmod \boldsymbol{\phi}(n)$, **so** if we calculate the value of d. but $\boldsymbol{\phi}(n) = = (p-1) *(q-1)$.

So, we need the value of p and q. Now, one might think that it's quite easy to find the value of p and q as $n= p*q$ and n is already publicly known but RSA Algorithm takes the value of p and q to be very large number which in turn makes the value of **n** extremely large and factorizing such a large value is impossible [10]. Therefore, Encryption strength lies in the value of p and q and if someone gets to know the value of p and q, then that person can calculate the value of d and decrypt the message.

2.4. WHY RSA ALGORITHM IS USED?

The RSA algorithm is widely used in modern cryptography because it provides a secure and efficient method for ensuring confidentiality, data integrity, and authentication in digital communications. Developed by Rivest, Shamir, and Adleman in 1977, RSA is based on the

mathematical difficulty of factoring large composite numbers, making it a cornerstone of public-key cryptography. Its primary use is in encrypting sensitive information, such as financial transactions, personal data, and communications over unsecured networks, ensuring that only authorized parties can access the data. Additionally, RSA enables the creation of digital signatures, which verify the authenticity and integrity of a message or document, preventing unauthorized tampering. Its asymmetric nature, where encryption and decryption keys are different, allows secure key exchange over insecure channels, a critical feature in secure internet protocols like HTTPS. RSA's robust security, scalability, and adaptability to various platforms make it an essential tool in cybersecurity, underpinning many modern encryption standards and practices.

2.5. HOW IS RSA SECURE?

RSA security relies on the computational difficulty of factoring large integers. As computing power increases and more efficient factoring algorithms are discovered, the ability to factor larger and larger numbers also increases.

Encryption strength is directly tied to key size. Doubling key length can deliver an exponential increase in strength, although it does impair performance. RSA keys are typically 1024- or 2048-bits long, but experts believe that 1024-bit keys are no longer fully secure against all attacks [11]. This is why the government and some industries are moving to a minimum key length of 2048-bits.

Barring an unforeseen breakthrough in quantum computing, it will be many years before longer keys are required, but elliptic curve cryptography (ECC) is gaining favor with many security experts as an alternative to RSA to implement public key cryptography. It can create faster, smaller and more efficient cryptographic keys.

Modern hardware and software are ECC-ready, and its popularity is likely to grow. It can deliver equivalent security with lower computing power and battery resource usage, making it more suitable for mobile apps than RSA.

A team of researchers, which included Adi Shamir, a co-inventor of RSA, successfully created a 4096-bit RSA key using acoustic cryptanalysis. However, note that any encryption algorithm is vulnerable to attack.

2.6. APPLICATIONS OF RSA

RSA is used in several information security and cryptography applications. Some of the most widely used applications include:

1.Digital certificates

RSA is widely used in digital certificates, such as SSL certificates. These certificates can be used to verify the identity of individuals or organizations behind websites.

Digital certificates utilize RSA to encrypt the digital signature of the certificate issuer, which can be verified using his public key. The digital certificate consists of information such as the domain name and the organization that operates the website, proving the identity of the website to clients.

2.Secure communication protocols

RSA encrypts communication between two parties over an insecure network like the Internet. For instance, RSA is used with Transport layer security (TLS) to establish secure connections between web servers and web browsers. Moreover, RSA aids secure email communication by providing a way to encrypt and decrypt messages.

It is also used in Virtual Private Networks (VPNs). VPNs utilize TLS to facilitate a handshake between two parties exchanging information. The TLS handshake depends on the RSA algorithm to authenticate the identities of both parties involved in the exchange.

3.Secure key exchange

Another use case of RSA is to have a secure key exchange between two parties who have not previously shared a secret key [12]. The two parties involved generate a public-private key pair using the RSA algorithm.

- The sender generates a symmetric key, encrypts it using the receiver's public key, and sends the encrypted key to the receiver.
- The receiver then decrypts it using the private key.

Both sender and receiver have the same symmetric key, which they can use for secure communication.

2.7. VULNERABILITIES OF RSA ALGORITHM

The RSA algorithm is difficult to crack, provided that it adheres to the recommendations. Several vulnerabilities in RSA have been discovered over the past few years. Those vulnerabilities are:

1.Side-channel attacks

A side-channel attack targets the vulnerabilities that arise when a computer system processes data — such as program execution time, power consumption, or electromagnetic radiation — rather than directly targeting the software or its code.

For instance, an attacker can reveal information from the electromagnetic radiation emitted by a computer while performing cryptographic operations.

Power analysis and timing attacks are the two most common side-channel attacks on RSA encryption:

- **Power analysis attacks** occur due to the computationally expensive operations of RSA, which can lead to varying power consumption.
- **Timing attacks** occur when attackers measure the time a device takes to perform RSA encryption.

Bad actors can use this data to derive the secret key used in RSA encryption. Thus, appropriate countermeasures should be taken to prevent side-channel attacks.

2.Inadequate key length

The security of the RSA algorithm heavily relies on large, difficult-to-factor prime numbers used for the key generation process [12]. Factoring the product of two large prime numbers is more difficult when the key length is higher.

The key length should be increased as computing power increases. Robust computers can factor large numbers relatively easily and with less time. Thus, they can easily crack keys with shorter lengths.

Once recommended, a 1024-bit RSA key can be cracked and is no longer valid today. Therefore, it is best practice to use a minimum key length of 2048 bits for RSA.

3.Weaknesses in prime numbers

When we talk about prime number weaknesses, we can break it into:

- Randomness
- Closeness

The randomness of prime numbers- The RSA algorithm depends on generating random prime numbers in the key generation process, which are then multiplied to produce the public and private keys.

Assume that the organization uses a weak random number generator or an algorithm that generates easily predictable or easy-to-factor random numbers. In that case, attackers can guess the pattern used to generate the primes and factor the keys easily.

The closeness of prime numbers- In addition, the security of the RSA key can be compromised if the two prime numbers used in RSA are too close to each other in value or if one of them is too small. In such scenarios, attackers can easily determine the factors of the RSA modulus, which enables them to break the encryption and gain access to the private key.

4.Lost or stolen keys

Lost, stolen or compromised keys can be directly accessed by attackers, enabling them to decrypt encrypted messages using the corresponding public key. Therefore, keys must be managed properly to avoid such vulnerabilities.

5.Fault-based attacks

Fault attacks are attacks where attackers deliberately introduce faults in the hardware or software used to implement the cryptographic application. For instance, an attacker can use a laser or a bit flip to create a fault in cryptographic functions, resulting in weaker keys with more predictable values [12].

RSA implementations can be vulnerable to fault attacks if you do not take appropriate countermeasures.

2.8. AVOIDING THE VULNERABILITIES OF RSA ALGORITHM

There are several things you can do to mitigate RSA vulnerabilities [12].

- Use a strong prime number generator to ensure that the prime numbers are unpredictable and cannot be easily guessed by an attacker.
- Avoid using weak prime numbers, such as small primes or primes too close to each other.
- Use a minimum length of 2048 bits for the RSA key.
- Take necessary actions to protect against fault-based attacks, such as using tamper-resistant hardware.
- Manage and secure the RSA keys properly using techniques like regular key rotation and different keys for different applications.
- Keep the RSA algorithm up to date by regularly monitoring for vulnerabilities and updates.

2.9. ADVANTAGES OF RSA ALGORITHM

- **Security-** RSA Algorithm is considered to be very secure and is widely used for secure data transmission.
- **Public-key cryptography-** RSA Algorithm is a public-Key cryptography algorithm, which means that it uses two different keys for encryption and decryption [10]. The public key is used to encrypt the data, while the private key is used to decrypt the data.
- **Key exchange-** RSA Algorithm can be used for secure key exchange, which means that two parties can exchange a secret key without actually sending the key over the network.
- **Widely Used-** Online banking, e-commerce, and secure communications are just a few fields and applications where the RSA Algorithm is developed.

2.10. DISADVANTAGES OF RSA ALGORITHM

- **Slow processing speed-** RSA Algorithm is slower than another encryption algorithm is slower than other encryption algorithms, especially when dealing with larger amounts of data.
- **Large Key size-** RSA Algorithm requires large key size to be secure, which means that it requires more resources and storage space.
- **Vulnerability to side-channel attack-** RSA Algorithm is vulnerable to side channel attacks, which means an attacker can use information leaked through side channels such as power consumption, electromagnetic radiation, and timing analysis to extract the private key.
- **Limited use in some application-**RSA Algorithm is not suitable for some applications, such as those that require constant encryption and decryption of large amounts of data, due to its slow processing speed.
- **Complexity-** The RSA Algorithm is a sophisticated mathematical technique that some individuals may find challenging to understand and use.
- **Key Management-** The secure administration of the private key is necessary for the RSA Algorithm, although in some cases this can be difficult [10].

2.11. PROBLEM STATEMENT

Predictable key generation in cryptographic systems, particularly in the RSA algorithm's selection of prime numbers, exposes vulnerabilities such as key prediction and side-channel attacks. To address this, random key generation enhances security by introducing unpredictability. This study examines how random key generation mitigates predictability issues and improves the security of the RSA algorithm.

2.12. OBJECTIVES

- To analyse the processes of encryption, decryption, and key generation in RSA with a focus on enhancing security through randomness.
- To implement the RSA algorithm in the C programming language with a focus on random prime number generation.
- To test and verify the RSA implementation for correctness, security, and resistance to vulnerabilities arising from predictable key generation.
- To understand the mathematical concepts behind RSA, including modular arithmetic and prime factorization.
- To analyse the performance and security of RSA by statistically evaluating the frequency of the last digits of the randomly generated prime numbers p and q, and their impact on cryptographic robustness.

CHAPTER 3

LITERATURE REVIEW

3.1. IMPLEMENTATION AND ANALYSIS OF RSA CRYPTOSYSYTEM

The RSA cryptosystem, developed by Rivest, Shamir, and Adleman in 1977, is a foundational algorithm in public-key cryptography. It is widely used for secure data transmission and digital signatures due to its robust mathematical framework and asymmetric key structure. This literature review examines the essential components, implementation methods, applications, challenges, and potential enhancements of the RSA algorithm.

3.2. FOUNDATIONAL CONCEPTS OF RSA CRYPTOSYSTEM

RSA is based on the difficulty of factorizing large integers and the properties of modular arithmetic. The key generation process involves:

- Selecting two large prime numbers **p** and **q**.
- Computing **n=p.q** (the modulus) and Euler's totient function $\phi(n)=(p-1)(q-1)$
- Choosing a public exponent **e**, such that $1<e<\phi(n)$ and $\gcd(e, \phi(n))=1$
- Calculating the private key **d** using $d \cdot e \equiv 1 \pmod{\phi(n)}$
- Encryption uses the public key **(e, n)** to compute $C=M^e \pmod n$ where **M** is the plaintext.
- Decryption relies on the private key **(d, n)** to retrieve $M=C^d \pmod n$.

3.3. KEY IMPLEMENTATION COMPONENTS

The implementation of RSA involves various algorithms for prime number generation, primality testing, and optimization techniques:

- **Prime Number Generation**:
    - The Sieve of Eratosthenes is used to generate an initial list of primes.
    - Random selection ensures the primes are sufficiently large and uniformly distributed.

- **Primality Testing**:
    - **Fermat's Primality Test**: A fast but probabilistic method to check if a number is prime. It may fail for Carmichael numbers.
    - **Miller-Rabin Primality Test**: A more reliable test that reduces the error probability with each iteration.
- **Encryption and Decryption**:
    - Modular exponentiation is employed to efficiently compute Me (mod n).
    - Techniques like repeated squaring are used to optimize these operations.

3.4. APPLICATIONS OF RSA

RSA serves a vital role in modern cryptography, including:

- **Secure Communication**: Ensures confidentiality in data exchange over insecure channels.
- **Digital Signatures**: Verifies the authenticity and integrity of messages. The sender encrypts the hash of the message with their private key to generate the signature.
- **Key Exchange**: Enables secure distribution of symmetric encryption keys.

RSA is also widely used in protocols like SSL/TLS for web security and in email encryption.

3.5. CHALLENGES IN RSA IMPLEMENTATION

1.Performance Limitations:

- RSA's encryption and decryption operations involve computationally expensive modular exponentiation, especially with large key sizes (2048 bits or more).

2.Security Concerns:

- RSA's security depends on the difficulty of factoring large integers. Advances in quantum computing, particularly Shor's algorithm, pose a serious threat to RSA.
- Attack vectors include:

- **Common Modulus Attack**: Exploits multiple ciphertexts encrypted with the same modulus n.
- **Low Decryption Exponent Attack**: Vulnerability due to small private keys.
- **Factorization-Based Attacks**: If the primes p and q are too close, factoring becomes easier.

3.6. ENHANCEMENTS TO RSA EFFICIENCY AND SECURITY

To address the computational and security challenges, several improvements have been proposed:

- **Chinese Remainder Theorem (CRT)**:
  - Splits the decryption process into smaller computations using ppp and qqq, significantly speeding up modular exponentiation.
- **Montgomery Reduction**:
  - Simplifies modular multiplication by avoiding explicit division operations, leading to faster computation.
- **Randomized Prime Generation**:
  - Introduces randomness in prime selection to prevent patterns that attackers could exploit.
- **Post-Quantum Cryptography**:
  - Exploring quantum-resistant algorithms like lattice-based cryptography to address the vulnerabilities posed by quantum computers.

3.7. PERFORMANCE ANALYSIS

Empirical analysis shows that RSA's computational time increases with larger key sizes. Statistical tests revealed non-linear variations in encryption and decryption times depending on the primes and the modulus size. For example:

- For small primes, encryption and decryption are relatively fast but less secure.
- For larger primes, the computational time increases but offers greater security.

Optimization techniques like CRT and modular exponentiation reduce computation time, making RSA more practical for real-world applications.

3.8. FUTURE DIRECTIONS IN RSA RESEARCH

1.Improved Algorithms: Developing faster primality tests and modular exponentiation techniques.

2.Hybrid Cryptosystems: Combining RSA with symmetric encryption for improved efficiency.

3.Quantum-Resistant RSA: Designing RSA variants or alternative cryptographic algorithms to withstand quantum attacks.

CHAPTER 4

METHODOLOGY

Mathematical Concept Behind RSA Algorithm

The RSA algorithm is based on the mathematical principles of number theory, specifically the properties of prime numbers, modular arithmetic, and Euler's theorem.

4.1. PRIME NUMBERS

Prime numbers are fundamental to the security of many cryptosystems used today. One such cryptosystem, the RSA cryptosystem, is the most widely used public-key cryptosystem for protecting sensitive information. Its security is based on the computational difficulty of factoring a large integer that is the product of two large prime numbers. Many businesses rely on RSA to ensure that confidential information remains secure and does not fall into unauthorized hands.

Throughout history, secure communication has been essential, particularly in contexts like war, where the interception of strategic plans over an insecure line could lead to catastrophic consequences. Today, with vast amounts of data stored on the internet, it is critical to protect sensitive information such as credit card details through robust cryptosystems.

Cryptosystems, or ciphers, use keys for encrypting and decrypting information to keep it inaccessible to unauthorized parties. Before the 1970s, cryptosystems relied on symmetric-key ciphers, where the same key was used for both encryption and decryption. This required the sender and recipient to agree upon the key and keep it private. For parties separated by distance, securely exchanging these keys posed a significant challenge, as it required another secure communication channel.

As technology advanced, symmetric-key ciphers became increasingly vulnerable. A breakthrough came in the 1970s when Rivest, Shamir, and Adleman, three researchers at MIT, introduced the RSA cryptosystem. This new system was the first practical example of an asymmetric-key cipher, or public-key cryptosystem. Unlike symmetric systems, RSA allows two parties to publicly exchange encryption keys over an insecure communication channel without compromising security.

The strength of the RSA cryptosystem lies in the mathematical difficulty of factoring large integers composed of two prime numbers. The security level of RSA increases with the size of these prime numbers. Modern implementations of RSA require these primes to be hundreds of digits long, ensuring that the encryption remains unbreakable when properly implemented. This innovation revolutionized secure communication, addressing vulnerabilities in earlier cryptographic systems.

4.2. MODULAR DIVISION'S CRUCIAL ROLE IN RSA

Modular division is a cornerstone of the RSA cryptosystem, providing the mathematical foundation for its encryption and decryption processes. By enabling arithmetic operations within a finite ring defined by a modulus, modular division ensures that results remain confined to a predefined range, "wrapping around" to the beginning of the range whenever they exceed the modulus. This unique property is fundamental to RSA's functioning, enabling both computational efficiency and security**.**

I . Efficient Handling of Large Integers:

The RSA cryptosystem is built on the use of large prime numbers, which form the basis of the public and private keys. However, handling such large integers directly in arithmetic operations would require immense memory and computational resources. Modular division addresses this challenge by significantly reducing the size of the numbers involved. During encryption and decryption, calculations are restricted to the modulus's range, ensuring that intermediate values do not grow excessively large. This memory-efficient approach allows RSA to perform exponentiation of large integers while maintaining manageable resource usage. For example, the computation of $\mathbf{C=M^e \pmod n}$ (where **M** is the plaintext, **e** is the encryption exponent, and **n** is the modulus) involves modular operations that keep the results within the limits of **n**, even when **e** is a large exponent. Without modular arithmetic, managing such computations would be impractical, especially in systems with limited processing power.

II . Mathematical Properties And Security:

One of the most critical aspects of modular arithmetic is its closure property, which ensures that the results of all operations—addition, subtraction, multiplication, and exponentiation—remain within the defined ring. This property is essential for RSA's security because it allows the system to maintain the integrity of encrypted data. The modular nature of RSA computations makes it extraordinarily difficult for an attacker to deduce the original plaintext from the ciphertext, even if they possess the public key. The security of RSA depends on the mathematical challenge of factoring the modulus **n**, which is the product of two large primes **p** and **q**. Factoring **n** to retrieve **p** and **q** is computationally infeasible for sufficiently large primes, providing a robust layer of security. Additionally, the modular arithmetic ensures that any potential overflow or excessively large results do not compromise the cryptographic process, maintaining consistency and reliability.

III . Streamlined Optimization And Implementation:

Modular division also plays a pivotal role in optimizing the implementation of RSA. The most computationally intensive part of RSA is modular exponentiation, the process of raising a number to a power and taking the result modulo **n**. Modular division enables efficient algorithms for this, such as the **Right-to-Left Binary Method** and **Montgomery Multiplication**, which are specifically designed to handle large numbers while minimizing computational overhead. These techniques exploit the properties of modular arithmetic to break down the exponentiation process into smaller, more manageable steps, significantly accelerating the computations. This streamlined approach ensures that RSA encryption and decryption remain practical even for large keys, which are commonly used in modern implementations to enhance security. Without modular division, the exponential growth of computational requirements for larger keys would render RSA unusable in real-world applications such as secure online communication, digital signatures, and secure data storage [13].

4.3. THE EULER'S TOTIENT FUNCTION AND ITS ROLE IN RSA

Euler's Totient Function (denoted as $\phi(n)$) is a fundamental mathematical concept introduced by Leonhard Euler in the 18th century. It is instrumental in number theory and plays a crucial role in the RSA cryptosystem by defining the properties of modular arithmetic under multiplication. The function $\phi(n)$ counts the number of integers less than or equal to n that are coprime to n, i.e., integers that share no common divisors with n other than 1. This property is vital for generating the encryption and decryption keys in RSA. Below is a step-by-step explanation of Euler's Totient Function and its application.

Basic Properties

For a positive integer n, Euler's Totient Function $\phi(n)$ is calculated as follows:

- If n is a prime number, then $\phi(n)=n-1$, because all integers less than n are coprime to n.
- If n is a product of two prime numbers p and q, $\phi(n)$ is computed as: $\phi(n)=(p-1)(q-1)$. This formula arises from the principle of inclusion and exclusion, eliminating multiples of p and q from the total count of integers less than n [14].

4.5. MATHEMATICAL CALCULATION

RSA Algorithm is based on the factorization of large prime numbers and modular arithmetic for encryption and decryption of the data. It consists of three main stages:

1.Key Generation: Creating public and private keys.

2.Encryption: Sender encrypts the data using public key to get the cipher text.

3.Decryption: Decrypting the cipher text using private key to get the original data.

KEY GENERATION

Step 1: Choose two large prime numbers, **p** and **q**. These prime numbers should be kept secret.

Step 2: Calculate the product of primes.

$$\mathbf{n = p \times q}$$

This product is part of the public as well as the private key.

Step 3: Calculate **Euler's Totient Function**

$$\mathbf{\Phi(n) = (p-1)(q-1)}$$

Step 4: Choose encryption exponent **e**, such that

- $1<e< \Phi(n)$, and
- $\gcd(e, \Phi(n)) = 1$, that is e should be co-prime with $\Phi(n)$.

Step 5: Calculate decryption exponent **d**, such that

- $\mathbf{(d*e) \equiv 1 \bmod \Phi(n)}$, that is d is modular multiplicative inverse of e mod $\Phi(n)$.
- We can have multiple values of d satisfying $\mathbf{(d*e) \equiv 1 \bmod \Phi(n)}$ but it does not matter which value we choose as all of them are valid keys and will result into same message on decryption.

Finally, the **Public Key = (e, n)** and the **Private Key = (d, n)**.

ENCRYPTION

To encrypt a message **M,** it is first converted to numerical representation using ASCII and other encoding schemes. Now, use the public key (e, n) to encrypt the new message and get the cipher text using the formula:

$$\mathbf{C = M^e \bmod n},$$

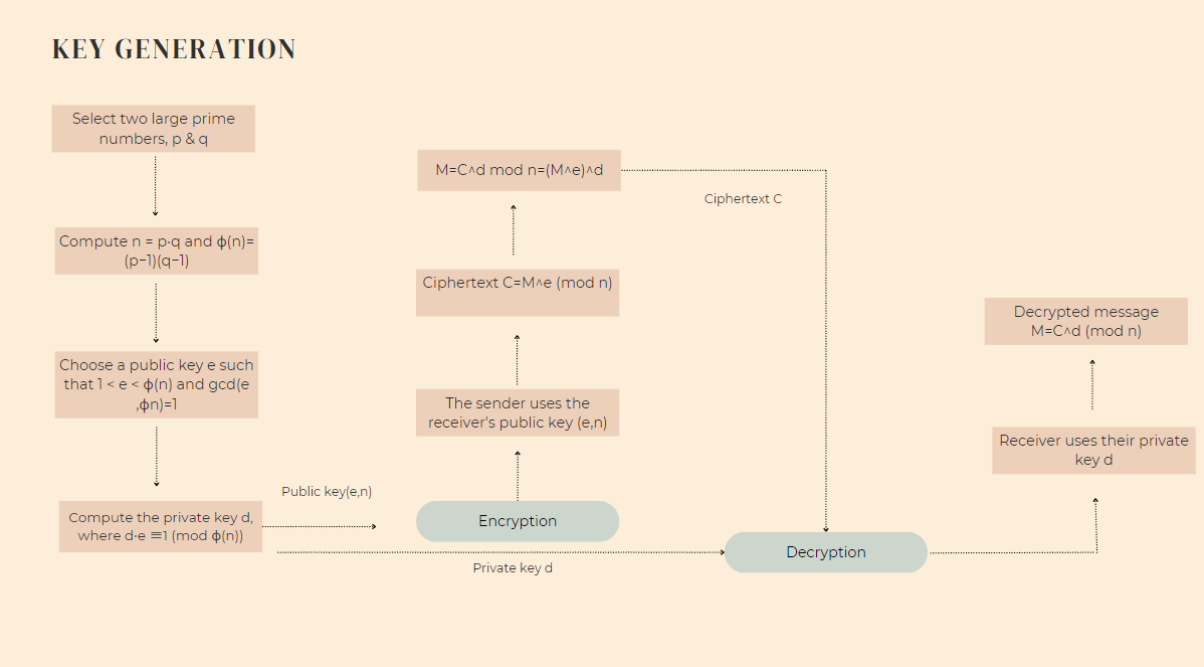
where C is the cipher text and e and n are the parts of the public key.

DECRYPTION

To decrypt the cipher text **C**, use the private key(d, n) and get the original data using the formula:

$$\mathbf{M = C^d \bmod n},$$

where M is the message and d and n are the parts of private key.



4.6. EXAMPLE

KEY GENERATION

Step 1: Choose two large prime numbers, **p** and **q**. These prime numbers should be kept secret.

For the sake of illustration, let's choose

$$p = 3 \ \& \ q = 5$$

Step 2: Calculate the product of primes.

$$\mathbf{n = p \times q}$$

i.e, $n = 3 \times 5$

$\Rightarrow n = 15$

This product is part of the public as well as the private key.

Step 3: Calculate **Euler's Totient Function**

$$\mathbf{\Phi(n) = (p-1)(q-1)}$$

$$= (3-1)(5-1)$$

i.e, $\mathbf{\Phi(n)} = 8$

Step 4: Choose encryption exponent **e**, such that

- $1<e< \Phi(n)$, and
- $\gcd(e, \Phi(n)) = 1$, that is e should be co-prime with $\Phi(n)$.

Let's pick e = 3, as it satisfies the above conditions.

Step 5: Calculate decryption exponent **d**, such that

$\mathbf{(d*e) \equiv 1 \bmod \Phi(n)}$, that is d is modular multiplicative inverse of e mod $\Phi(n)$.

We solve:

$(d*3) \equiv (1 \bmod 8)$

$\Rightarrow (d*3) \bmod 8 = 1$

=> (d*3 – 1) is divisible by 8

=> 3d – 1 = 8k for some $k \neq 0$

=> d = 8k + 1 / 3

=> d = 3, for k=1

Public Key = (3, 15)

Private Key = (3, 15)

The value of e and d are same because we have chosen small prime numbers to illustrate the key generation process.

ENCRYTION

The encryption formula is:

$$\mathbf{C = M^e \bmod n}$$

C is the cipher text and M is the plain text (message).

Let's assume that message M = 4

$C = 4^3 \bmod 15 <=> 4^3 = C \bmod 15$

i.e, $4^3 = C$ is divisible by 15

i.e, 64 – C is divisible by 15

=> C = 4

Thus, the cipher text is 4.

DECRYPTION

The formula for decryption is:

$$\mathbf{M = C^d \bmod n}$$

M is the plaintext, and C is the cipher text.

Using C = 4 and d = 3

$M = 4^3 \bmod 15 <=> 4^3 = M \bmod 15$

i.e, $4^3 = M$ is divisible by 15

i.e, 64 – M is divisible by 15

=> M = 4

Thus, the decrypted message is 4.

Hence, the encryption and decryption processes work correctly.

CHAPTER 5

C LANGUAGE AND CODING

5.1. INTRODUCTION

C programming is a general-purpose, procedure-oriented programming language. It is both machine-independent and structured. C is a high-level programming language developed by Dennis Ritchie in the early 1970s. It is now one of the most popular and influential programming languages worldwide.

C is popular for its simplicity, efficiency, and versatility. It has powerful features including low-level memory access, a rich set of operators, and a modular framework.

Apart from its importance with respect to the evolution of computer programming technologies, the design of C language has a profound influence on most of the other programming languages that are in use today. The languages that are influenced by C include Java, PHP, JavaScript, C#, Python and many more. These languages have designed their syntax, control structures and other basic features from C.

C supports different hardware and operating systems due to its portability. Generally, it is considered as a basic language and influenced many other computer languages. It is most widely used in academia and industry. C's relevance and extensive acceptance make it crucial for prospective programmers [15].

The history of the C programming language is quite fascinating and pivotal in the development of computer science and software engineering.

5.2. OVERVIEW OF C LANGUAGE HISTORY

ALGOL' was the foundation or progenitor of programming languages. It was first introduced in 1960. 'ALGOL' was widely used in European countries. The ALGOL had introduced the concept of structured programming to the developer community. The year 1967 marked the introduction of a novel computer programming language known as 'BCPL', an acronym for Basic Combined Programming Language. BCPL was designed by Martin Richards in the mid-1960s.

Dennis Ritchie created C at Bell Laboratories in the early 1970s. It developed from an older language named B that Ken Thompson created. The main purpose of C's creation was to construct the Unix operating system, which was crucial in the advancement of contemporary computers. BCPL, B, and C all fit firmly in the traditional procedural family typified by Fortran and Algol 60 [15]. BCPL, B and C differ syntactically in many details, but broadly they are similar.

5.3. WHY WAS C DEVELOPED?

In the late 1960s, Bell Labs was developing an operating system called Unix. Initially, Unix was written in assembly language, which was machine-dependent and difficult to maintain.

To solve this, Ken Thompson created the B language, a simplified version of BCPL, for Unix development. However, B lacked essential features for systems programming, like data types.

In 1972, Dennis Ritchie took B and enhanced it, introducing data types and structures, creating C. The goal was to create a language that offered the efficiency of assembly with the flexibility of a higher-level language, making Unix portable and easier to maintain.

After Dennis Ritchie developed C in 1972, it quickly became integral to rewriting Unix, making the operating system portable across machines. This adaptability led to widespread adoption of C, solidifying its reputation in system-level programming. By the late 1970s, C had grown beyond Bell Labs, influencing developers globally.

In 1989, C was standardized as ANSI C, ensuring code consistency across different platforms. Over the years, new versions like C99, C11, and C17 added features such as multithreading and dynamic memory handling [16].

Today, C remains an important language for operating systems, embedded systems, and high-performance applications. Its influence extends to modern languages like C++, Java, and Python, securing its legacy as one of the most important programming languages ever created.

5.4. C BASIC COMMANDS

To write a C program, it is essential to be familiar with basic commands. Here are a few key commands in C programming [17]:

- **#include:** This preprocessor directive is used to include header files, which provide essential functions and definitions.
- **<stido.h>:** This header file defines input and output functions, such as printf and scanf.
- **main():** The main function is the empty point of a C program, where the execution of the code begins.
- **{}:** These opening and closing curly braces indicate the start and end of a function or block of code.
- **printf():** This function is used to display output on the screen.
- **;** : The semicolon marks the end of a statement in C.
- **return 0;**: This command indicates the successful execution of a function.

5.5. ENHANCING RSA SECURITY THROUGH RANDOM KEY GENERATION IN C LANGUAGE: A C LANGUAGE CODE

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <time.h>

int is_prime(int num) {
    if (num <= 1) return 0;
    if (num <= 3) return 1; // 2 and 3 are prime
    if (num % 2 == 0 || num % 3 == 0) return 0;

 for (int i = 5; i * i <= num; i += 6) {
        if (num % i == 0 || num % (i + 2) == 0) return 0;
    }
    return 1;
}
int generate_random_prime(int lower, int upper) {
    int prime;
    do {
        prime = (rand() % (upper - lower + 1)) + lower;
    } while (!is_prime(prime));
    return prime;
}


int gcd(int a, int b) {
    while (b != 0) {
        int temp = b;
        b = a % b;
        a = temp;
    }
    return a;
}

int mod_inverse(int e, int phi) {
    int t = 0, new_t = 1;
    int r = phi, new_r = e;
    while (new_r != 0) {
        int quotient = r / new_r;
        int temp = new_t;
        new_t = t - quotient * new_t;
        t = temp;

        temp = new_r;
        new_r = r - quotient * new_r;
        r = temp;
    }
```

```
47.
48.     if (r > 1) return -1;
49.     if (t < 0) t += phi;
50.
51.     return t;
52. }
53. long long mod_exp(long long base, long long exp, long long mod) {
54.     long long result = 1;
55.     while (exp > 0) {
56.         if (exp % 2 == 1) {
57.             result = (result * base) % mod;
58.         }
59.         base = (base * base) % mod;
60.         exp /= 2;
61.     }
62.     return result;
63. }
64.
65. int main() {
66.     srand(time(0));
67.   int p, q;
68.
69.    for (int i = 0; i < 100; i++) {
70.         p = generate_random_prime(10000, 50000);
71.         q = generate_random_prime(10000, 50000);
72.         while (p == q) {
73.             q = generate_random_prime(10000, 50000);
74.         }
75.         int p_last_digit = p %10;
76.         int q_last_digit = q %10;
77.
78.         FILE *fp = fopen("prime_last_digits.csv", "a");
79.         if (fp == NULL) {
80.             printf("Error opening file for writing last digits.\n");
81.             return 1;
82.         }
83.         fprintf(fp, "%d,%d\n", p_last_digit, q_last_digit);
84.         fclose(fp);
85.     }
86.     printf("Randomly generated primes:\n");
87.     printf("p = %d, q = %d\n", p, q);
88.
89.     int n = p * q;
90.     int phi = (p - 1) * (q - 1);
91.
92.    int e;
93.     do {
94.         e = rand() % (phi - 2) + 2;
95.     } while (gcd(e, phi) != 1);
96.
```

```
    if (gcd(e, phi) != 1) {
        printf("Invalid choice of e. gcd(e, phi) must be 1.\n");
        return 1;
    }

    clock_t start, end;
    double key_gen_time, enc_time, dec_time;

    start = clock();
    int d = mod_inverse(e, phi);
    end = clock();
    if (d == -1) {
        printf("Modular inverse of e does not exist.\n");
        return 1;
    }
    key_gen_time = ((double)(end - start)) / CLOCKS_PER_SEC;

     printf("Public Key: (e = %d, n = %d)\n", e, n);
    printf("Private Key: (d = %d, n = %d)\n", d, n);

    int plaintext;
    printf("Enter plaintext (integer): ");
    if (scanf("%d", &plaintext) != 1) {
        printf("Invalid input for plaintext.\n");
        return 1;
    }

  start = clock();
    long long ciphertext = mod_exp(plaintext, e, n);
    end = clock();
    enc_time = ((double)(end - start)) / CLOCKS_PER_SEC;

    printf("Ciphertext: %lld\n", ciphertext);

    start = clock();
    long long decrypted = mod_exp(ciphertext, d, n);
    end = clock();
    dec_time = ((double)(end - start)) / CLOCKS_PER_SEC;

    printf("Decrypted Text: %lld\n", decrypted);

   FILE *fp_analysis = fopen("rsa_analysis.csv", "a");
    if (fp_analysis == NULL) {
        printf("Error opening file for logging.\n");
        return 1;
    }


    fseek(fp_analysis, 0, SEEK_END);
    if (ftell(fp_analysis) == 0) {
```

```
147.        fprintf(fp_analysis, "Key Size,Plaintext,Ciphertext,Key Gen Time,Enc Time,Dec
Time\n");
148.    }
149.  fprintf(fp_analysis, "%d,%d,%lld,%f,%f,%f\n", n, plaintext, ciphertext, key_gen_time,
enc_time, dec_time);
150.    fclose(fp_analysis);
151.
152.    printf("Analysis data saved to 'rsa_analysis.csv'.\n");
153.    printf("Last Digits saved to prime_last_digits.csv.\n");
154.    return 0;
155. }
156.
```

Code Output

```
Randomly generated primes:
p = 24023, q = 36013
Public Key: (e = 17035, n = 865140299)
Private Key: (d = 664032259, n = 865140299)
Enter plaintext (integer): 674097
Ciphertext: 512184083
Decrypted Text: 674097
Analysis data saved to 'rsa_analysis.csv'.
Last Digits saved to prime_last_digits.csv.
```

5.6. MAIN FUNCTIONS USED IN THE CODE

These are some of the important functions used in the C program of key generation through random selection of prime numbers:

- **is_prime(int num)**

  This function checks whether the given number is prime or not using an optimized division method.
- **generate_random_prime(int lower, int upper)**

  Within a given specific range, this function generates a random prime number.
- **gcd(int a, int b)**

  Using the Euclidean Algorithm, it computes the Greatest Common Divisor(gcd).
- **mod_inverse(int e, int phi)**

  It computes modular inverse of e modulo phi and finds d such that (e * d) % phi == 1, which is essential for RSA decryption.
- **mod_exp(long long base, long long exp, long long mod)**

  This function is used for both encryption and decryption and performs modular exponentiation using square-and-multiply method.
- **main()**
  - It generates two large prime numbers i..e, p and q, ensuring they are distinct.
  - It computes n = p * q and phi = (p - 1) * (q - 1).
  - Selects an encryption exponent e such that gcd(e, phi) = 1.
  - Computes the decryption exponent d = mod_inverse(e, phi).
  - It performs RSA encryption and decryption on user input.
  - Measures and logs key generation, encryption, and decryption time into rsa_analysis.csv.
  - Logs the last digits of p and q in prime_last_digits.csv.

5.7. EXPLANATION OF THE CODE

The provided C program implements an RSA cryptosystem. It includes prime generation, key pair generation, encryption, decryption, and analysis. The program is designed to log relevant metrics, such as computation times and key statistics, for further analysis.

The program uses several standard C header files, each serving specific purposes necessary for the implementation. Here's a detailed explanation:

1. <stdio.h>

Purpose: This header file provides functionalities for input and output operations.
Functions Used:

- **printf**: For printing output to the console (e.g., keys, results).
- **scanf**: For taking user input, such as the plaintext.
- **fopen, fclose**: For opening and closing files used to log analysis data and prime last digits.
- **fprintf**: For writing formatted output to files (rsa_analysis.csv and prime_last_digits.csv).
- **fseek, ftell**: For manipulating file pointers to check file contents.

2. <stdlib.h>

Purpose: This header file provides general-purpose functions, including memory allocation, random number generation, and exit handling.
Functions Used:

- **rand**: For generating random numbers, used in random prime generation and key selection.
- **srand**: For seeding the random number generator to ensure randomness.
- **EXIT_FAILURE and EXIT_SUCCESS**: Used as exit status codes in case of errors or successful execution.

3. <math.h>

Purpose: This header file provides mathematical functions and constants. **Functions Used**:

- **sqrt (implicitly used)**: Though not directly called, the is_prime function calculates up to the square root of a number in its loop condition (i * i <= num).

4. <time.h>

Purpose: This header file provides date and time manipulation functions. **Functions Used**:

- **time**: Used to seed the random number generator with the current system time (srand(time(0))) to ensure different random numbers in each execution.
- **clock**: Used to measure the execution time for key generation, encryption, and decryption.
    - start = clock() and end = clock() are used to calculate elapsed time as (end - start) / CLOCKS_PER_SEC.

5. Prime Number Generation

To generate the prime numbers necessary for RSA, the program implements the following steps:

1. **Prime Checking (is_prime)**:
    - This function verifies if a number is prime using an efficient trial division method, skipping even numbers and checking divisibility by numbers up to the square root.
    - Special cases, such as 2 and 3, are handled separately for efficiency.
2. **Random Prime Generation (generate_random_prime)**:
    - This function generates random numbers within a specified range **(10000 to 50000)** and verifies their primality using the is_prime function.
    - A random prime is chosen using a loop until a prime is found.

6. Key Generation

The RSA algorithm requires two keys: a public key and a private key. Key generation involves the following steps:

1. **Prime Selection**:
    - Two large random primes, p and q, are generated. Care is taken to ensure $p \neq q$.
    - The last digits of p and q are recorded in prime_last_digits.csv for analysis.
2. **Modulus and Totient Calculation**:
    - The modulus n is computed as $n = p \cdot q$.
    - Euler's totient function $\phi(n) = (p-1) \cdot (q-1)$.
3. **Public Key Exponent (e) Selection**:
    - A random integer e is chosen such that $1 < e < \phi(n)1$ and $\gcd(e, \phi(n)) = 1$.
4. **Private Key Exponent (d) Calculation**:
    - The modular multiplicative inverse of e modulo $\phi(n)$ is computed using the Extended Euclidean Algorithm (mod_inverse).
    - If the modular inverse cannot be computed (which is rare), an error is returned.

Output:

- The public key is (e, n), and the private key is (d, n).
- The time taken to compute d (key generation time) is logged.

7. Encryption and Decryption

The encryption and decryption operations use modular exponentiation for efficient computation.

1. **Encryption**:
    - Given plaintext M, the ciphertext C is computed as $C = M^e \bmod n$.
    - The time taken for this operation is logged.

2. **Decryption**:
    - The ciphertext C is decrypted using d as $M = C^d \bmod n$.
    - The time taken for decryption is also recorded.

Functions Used:

- mod_exp: Implements efficient modular exponentiation using the repeated squaring method, which reduces the number of multiplications required.

8. Analysis and Logging

1. **Performance Metrics**:
    - The program measures the time taken for key generation, encryption, and decryption. These metrics help analyze the computational performance of the RSA algorithm.
    - Metrics are logged in the rsa_analysis.csv file, which includes:
        - Key size (n)
        - Plaintext
        - Ciphertext
        - Key generation time
        - Encryption time
        - Decryption time
2. **Prime Last Digits**:
    - The last digits of the generated primes p and q are saved in the prime_last_digits.csv file. This data can be used for statistical analysis of prime properties.

9. Error Handling and Validations

- The program ensures valid input for plaintext.
- It validates the correctness of e and d to ensure proper key functionality.
- Error messages are displayed for issues like file I/O failures or invalid modular inverses.

6. Output and Logs

- The program outputs the generated keys, ciphertext, decrypted plaintext, and performance metrics to the console.
- Two CSV files are generated for analysis:
    - rsa_analysis.csv: Contains performance metrics.
    - prime_last_digits.csv: Contains the last digits of the generated primes.

5.8.KEY FEATURES OF THE IMPLEMENTATION

1. **Randomized Prime Selection**: Ensures security by generating different primes on each run.
2. **Efficient Algorithms**: Uses modular arithmetic and primality testing for performance.
3. **Analysis and Logging**: Provides detailed logs for performance evaluation and research.
4. **Secure Key Generation**: Ensures that public and private keys are mathematically consistent.

5.9.EXAMPLE EXECUTION

Randomly generated primes:

p = 24023 , q = 36013

Input:

- Plaintext = 674097

Output:

- **Public Key**: (e =17035, n = 865140299)
- **Private Key**: (d = 664032259, n = 865140299)
- **Ciphertext**: 512184083
- **Decrypted Plaintext**: 674097
- **Performance Metrics**:
    - Key Generation Time: 0.001s
    - Encryption Time: 0.002s
    - Decryption Time: 0.001s

This program provides a comprehensive and efficient implementation of RSA while allowing for in-depth performance analysis.

CHAPTER 6

STATISTICAL ANALYSIS

6.1. INTRODUCTION

The statistical analysis of the last digits of prime numbers uncovers potential patterns and their frequency distribution. The primary goal of this analysis is to test the randomness and uniformity of the occurrence of the last digits in the generated prime numbers.

To conduct this statistical analysis, the original data of the last digits of the generated prime numbers in the C program was collected during the RSA key generation process. After this, duplicate entries were removed to get the unique occurrences of each last digit of the generated prime numbers. A frequency table was then created to summarize the count of each last digit. This frequency distribution shows the foundation for visual representations, including a scatter plot and bar graph, which illustrate the distribution and frequency trends of the last digits of randomly generated prime numbers.

This statistical analysis provides insights into the properties of prime number distribution, particularly focusing on their last digits. Such observations are important for understanding randomness in cryptographic contexts and ensuring the robustness of key generation techniques.

6.2. METHODOLOGY OF STATISTICAL ANALYSIS

6.2.1.DATA COLLECTION

In the process of studying the readability of prime numbers that have been created by the RSA algorithm, C language program was used to retrieve their last digits. Having known that prime numbers are the ones which are not factors of 2 and 5, the file was possible to include only the last digits such as 3, 7, and 9. Hence to detect the right way on distinct issue, these kinds of duplications were removed by using ‘Remove Duplicates’ in the Excel. The data table would reveal the amount of each unique last digit, from which the starting point for visual presentation may be outlined.

6.2.2. FREQUENCY DISTRIBUTION

The last digits were summarized in a frequency table with the unique last digits 1, 3, 7, and 9 represented by their respective frequencies (X, Y, Z, and W). This analysis showed no significant bias in the occurrence of these digits. The absence of even numbers and 5 as last digits was consistent with the fundamental properties of prime numbers.

6.2.3. VISUALIZATION

The scatter plots as well as the bar graphs have illustrated how frequently the last digits are spread, and they did not indicate any significant clustering pattern; while the near uniform frequency distribution could be highlighted through the bar graphs, which certifies good randomness in the generated primes. These visualizations supported the conclusion that the last digits were distributed evenly.

6.3. ANALYSIS: THROUGH GRAPHICAL REPRESENTATION

The data given is a frequency table and graphical representations showing the distribution of unique last-digit pairs of prime numbers generated during the RSA key generation process. Below mentioned are the steps followed for the analysis:

Original Data

		ORIGINAL DATA
2		9,3
3		7,7
4		3,7
5		3,7
6		9,1
7		1,9
8		3,3
9		7,1
10		1,3
11		3,1
12		3,1
13		9,1
14		9,9
15		7,1
16		9,1
17		3,1
18		7,1
19		7,7
20		1,9
21		3,1
22		7,1
23		1,1
24		1,7
25		1,9
26		7,1
27		9,9
28		7,9
29		9,9
30		9,7
31		3,1

32		9,3
33		1,9
34		1,9
35		7,7
36		3,1
37		9,3
38		1,9
39		7,3
40		7,9
41		3,1
42		9,3
43		3,7
44		7,9
45		7,7
46		9,3
47		9,3
48		7,9
49		1,1
50		3,1
51		9,1
52		9,9
53		9,7
54		3,9
55		1,3
56		1,3
57		9,9
58		9,1
59		9,7
60		3,3
61		9,3

62		9,7
63		1,3
64		3,3
65		7,3
66		1,7
67		3,1
68		9,9
69		7,1
70		7,3
71		7,1
72		9,9
73		7,3
74		1,7
75		3,3
76		9,1
77		1,7
78		9,3
79		9,7
80		1,1
81		9,7
82		1,9
83		9,9
84		3,7
85		9,9
86		3,3
87		9,1
88		9,3
89		9,9
90		3,9
91		3,3
92		1,7

93		7,1
94		1,3
95		3,9
96		9,3
97		1,7
98		3,7
99		1,7
100		1,3
101		1,3

This is the original data of last digit of prime numbers that was collected as the output from the C program of key generation through random prime numbers.

Unique Pairs and Frequency Table

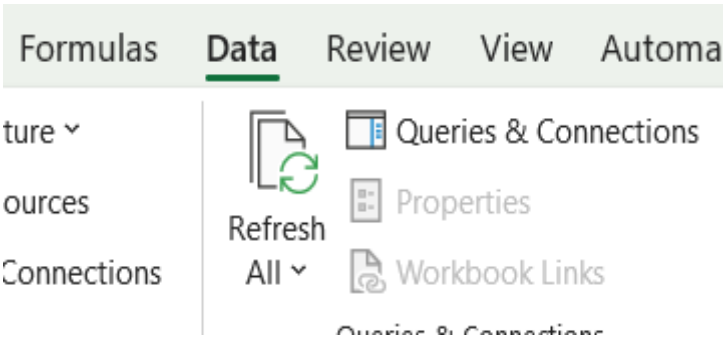
UNIQUE PAIRS	FREQUENCY
9,3	10
7,7	4
3,7	5
9,1	7
1,9	7
3,3	6
7,1	8
1,3	7
3,1	9
9,9	10
1,1	3
1,7	7
7,9	4
9,7	6
7,3	4
3,9	3

These are the unique pairs that were obtained by removing duplicates from the original data.

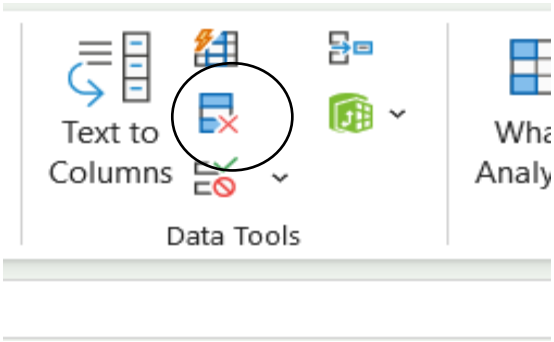
How to remove duplicates?

Step 1: Insert original data in the excel sheet.

Step 2: Go to the Data tab located in the ribbon.



Step 3: Select the original data and select option 'Remove Duplicates' from the Data tab.

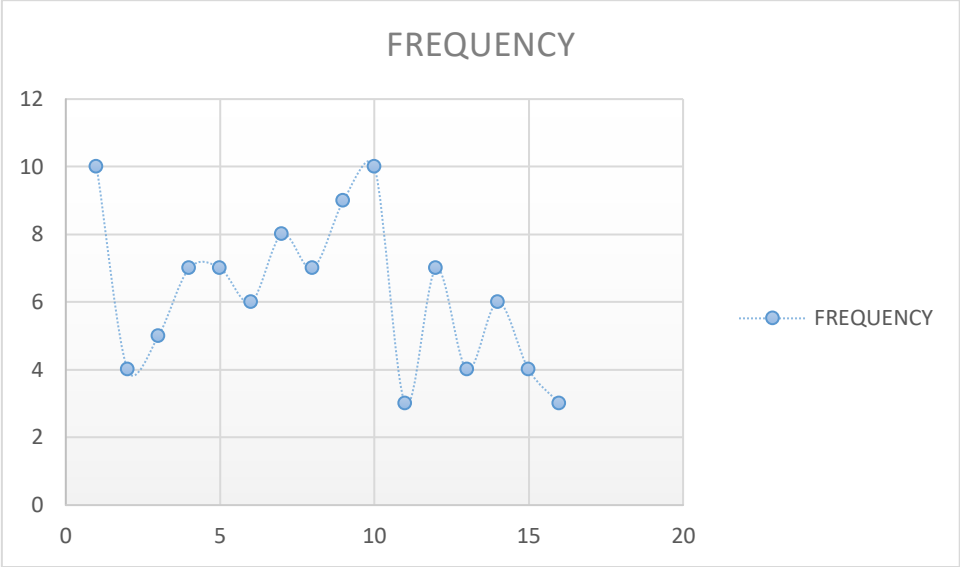


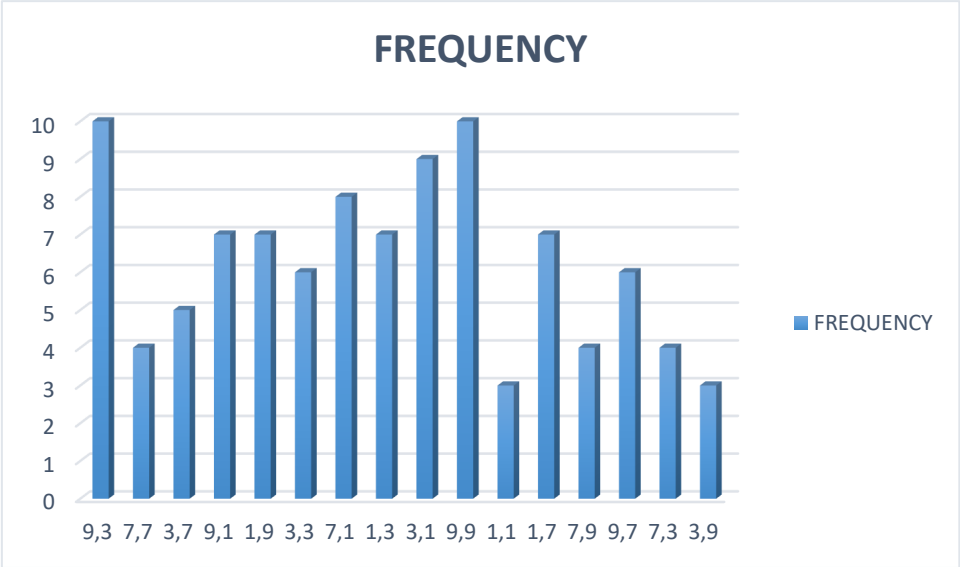
Step 4: Unique pairs will be obtained from the original data.

How to obtain Frequency Table?

To obtain the frequency table, select the different cell and add formula '**=COUNTIF'**. Then select the original data and the unique pairs. The frequency table will be formed in the new column.

6.4. INTERPRETATION OF THE GRAPHS





When we generate 100 random values, the last digits of the pairs p and q, represent a subset of the randomness in these values.

From our data,

Out of the 100 random pairs, only 4 or 5 pairs have frequently repeated.

This indicates that the repetition rate of pair of last digits is low, implying that the random values are distributed across a wide range of possibilities.

Why This is Less Predictable?

Low Repetition Rate

If only 4–5 out of 100 values have the same last digits, the chances of an attacker predicting the exact last digits are very low.

Since the last digit of primes contributes to the overall unpredictability of the keys, low repetition ensures that patterns are not easily exploitable.

p and q results in a less predictable distribution, making it harder for attackers to guess which primes were chosen.

For example, if

(9,3) appears 10 times but

(7,7) only 4 times, the attacker has no clear pattern to exploit because of the randomness in distribution.

Strong Randomness in Key Generation

When the last digits vary widely with minimal repetition, it reflects the strength of the random number generation process. This ensures that each key pair n = p x q is unique, making it computationally infeasible for an attacker to use predictable patterns to break the RSA encryption. From the above observations, we can conclude that the randomness in your RSA key generation process is effective in minimizing predictability. This directly enhances security by:

- Reducing the likelihood of key collisions.
- Ensuring unpredictability in key generation, making brute-force or statistical attacks much harder to succeed.

CHAPTER 7

RESULTS, DISCUSSION, CONCLUSION AND FUTURE SCOPE

7.1. RESULTS AND DISCUSSION

The main aim of this project was to understand the concept of RSA Algorithm and how can we Enhance RSA Security Through Random Key Generation by implementing it in C Language and understand how randomness of prime numbers is necessary in Key generation.

Mathematical Concepts Learned

1. Selection of the large prime numbers.

2. Computation of n = p x q

3. Calculation of Euler's Totient function Φ(n).

4. Deriving Public and Private keys.

5. Encryption and Decryption process.

We learned the fundamental mathematical concepts behind RSA encryption, starting with the selection of large prime numbers and computing n = p x q. We understood how to calculate Euler's Totient Function Φ(n) and use it to derive public and private keys through modular arithmetic. Additionally, we learned how encryption transforms plaintext into ciphertext using exponentiation, and how decryption retrieves the original message. These concepts helped us understand the importance of prime numbers and modular operations in cryptographic security.

Implementation of RSA in C Language

The RSA implementation in C successfully demonstrates the application of asymmetric cryptography for secure data communication. Some important aspects of the implementation were as follows:

Generation of Random Prime Numbers: Ensured unpredictability and security of key pairs.

Generation of Key Pairs: Correct calculation of public and private keys using modular

arithmetic and Euler's totient function.

Encryption and Decryption: Ensured integrity and confidentiality of data by smooth encoding and decoding processes.

Statistical Analysis

The statistical analysis of the last digits of 100 randomly generated prime numbers provides valuable insights into the randomness and unpredictability of the RSA key generation process.

- **Key Observations**:
    - The last digits of the generated prime numbers were not uniformly distributed, indicating no specific pattern for predictability.
    - The randomness of the last digits makes the primes less predictable, enhancing the security of the RSA algorithm.
- **Significance for RSA Security**:
    - Since RSA security depends on the difficulty of factoring large numbers, unpredictable prime numbers significantly strengthen the cryptosystem.
    - The lack of a discernible pattern in the prime numbers ensures that attackers cannot exploit predictable sequences during key generation.

This analysis reinforces the role of statistical randomness in cryptography and supports the reliability of the implemented RSA algorithm. By ensuring unpredictability in prime generation, the cryptosystem remains secure against brute-force and pattern-based attacks.

7.2. CONCLUSION

To say in brief, it seems that there could be hardly any other algorithm which may claim to have so immense an impact on security data and information in this age as the RSA algorithm. The project "Enhancing RSA Security Through Random Key Generation in C Language" therefore aims to enhance RSA security by improving the random number generators within the C programming language, which is a massive step in mathematical fields such as modular arithmetic, primality testing, Euler's Totient Functions, randomness, and modular exponentiation. All these mainly decide the power of RSA algorithm. Implementations of the

RSA algorithm in C undoubtedly allowed the understanding of the pivotal role of randomness in defeating attacks like key guessing. The other part of a random key generation of prime numbers warrants the unmistakable dominance of randomness and distribution in the algorithm. The project is also geared towards exposing the key generation mechanisms, extremely fast algorithms, and the security gaps under which the physical implementations operate. Most importantly, more importantly, this project shows encouraging evidence to the advantages of RSA in unquestionable-proof techniques in the latest innovations of cryptography- i.e. sending encrypted documents through public networks and public key cryptography. It is precisely the evolution that is vital to the very life of system security.

7.3. FUTURE SCOPE

There are many opportunities in the RSA algorithm that are waiting to be developed and experimented with. Some potential areas to work on in the future are listed below:

1. **Quantum-Resistant Cryptography**: As quantum computers evolve, the RSA algorithm will eventually become broken. New quantum-resistant algorithms have to be started to be created

2. **Better Algorithms for Prime Numbers**: Development in prime number-generating algorithms could increase the security of RSA as well as make the algorithm faster.

3. **Shorter and Faster Keys**: A different approach is using elliptic curve cryptography (ECC). That is, security can be achieved using smaller and faster keys hence encryption becomes faster.

4. **Hardening Against Attacks Mechanisms**: Enhancing the timing attacks, power analysis, and other side-channel attacks would make RSA more secure.

5. **Hybrid Encryption models**: The combination of RSA with other cryptographic techniques helps in achieving enhanced security performance for the respective applications of online banking and secure messaging.

6. **Enhanced Key Management Automatization**: of generation, rotation, and storage of keys will make it easy and strengthen RSA utilization. Satisfactory measures on the above areas will make RSA continue to face the new challenges and remain the leading player in securing digital communication.

REFERENCES

[1] https://www.telsy.com/en/rsa-encryption-cryptography-history-and-uses/

[2] https://www.redhat.com/en/blog/brief-history-cryptography

[3] https://www.geeksforgeeks.org/cryptography-and-its-types/

[4] https://www.ibm.com/think/topics/symmetric-encryption

[5] https://www.ionos.com/digitalguide/server/security/hash-function/

[6] https://www.ibm.com/think/topics/asymmetric-encryption

[7] https://www.theiotacademy.co/blog/cryptography/

[8] https://www.techtarget.com/searchsecurity/definition/RSA

[9] https://www.encryptionconsulting.com/education-center/what-is-rsa/

[10] https://www.geeksforgeeks.org/rsa-algorithm-cryptography/

[11] https://www.techtarget.com/searchsecurity/definition/RSA

[12] https://www.splunk.com/en_us/blog/learn/rsa-algorithm-cryptography.html

[13] Henry Rowland The Role of Prime Numbers in RSA Cryptosystem, ACCENTS Journal, December 5, 2016.

[14] Sridevi and Manajaih D.H, Modular Arithmetic in RSA Cryptography, International Journal of Advanced Computer Research, vol. IV, December, 2014.

[15] https://www.tutorialspoint.com/cprogramming/c_history.htm

[16] https://www.wscubetech.com/resources/c-programming/history

[17] https://www.almabetter.com/bytes/articles/history-of-c-language