RSA ALGORITHM

2024

Presented by:

M. Umer Riaz
Nabeel Ahmed
Subhan Amjad
Noor Fatima

RSA ALGORITHM

By

M. Umer Riaz

Nabeel Ahmed

Subhan Amjad

Noor Fatima

To

Dr. Nauman Shamim

May 19, 2024

Department of Computer and Information Sciences

PAKISTAN INSTITUTE OF ENGINEERING AND APPLIED SCIENCES

NILORE, ISLAMABAD 45650

INTRODUCTION

RSA encryption, named after its inventors, *Ron Rivest, Adi Shamir, and Leonard Adleman,* is a public-key cryptographic algorithm widely used for securing digital communication. It provides a robust method for ensuring data Confidentiality, Integrity, and Authentication (CIA) over insecure channels such as the internet. RSA encryption forms the foundation of many security protocols and applications, making it a crucial component of modern cryptography. Let's go through some basic and important concepts before we move on to encryption and decryption formulas.

FUNDAMENTAL PRINCIPLES

Euler's Totient Function (φ)

Euler's totient function, denoted as $\varphi(n)$, is a fundamental function in number theory. It is defined as the number of integers up to n that are co-prime to n. For a given integer n, $\varphi(n)$ counts the number of integers between 1 and n that share no common factors with n other than 1.

For RSA encryption, where n is the product of two distinct prime numbers p and q, the totient function is calculated as: $\varphi(n) = (p-1) \times (q-1)$ This is because, for a prime number p, there are $p-1$ numbers less than p that are relatively prime to p.

Fermat's Little Theorem

Fermat's Little Theorem states that if p is a prime number and a is an integer not divisible by p, then: $a^{(p-1)} \equiv 1 \bmod p$. Fermat's theorem is based on Euler's theorem because in Euler's theorem we say that: $gcd(a,n)=1$, that is, $a^{\varphi(n)} \equiv 1 \bmod n$ but in Fermat's theorem, we specify that n is a prime number, so Euler's theorem becomes $a^{(p-1)} \equiv 1 \bmod p$. This is crucial in RSA for determining the relationship between the public and private keys.

Prime Number Factorization and RSA security:

RSA encryption's security is directly linked with the prime number factorization. In RSA, the security of the algorithm relies on the mathematical impossibility of solving the problem of discovering the prime factors of a large composite number **n**, which is the product of the two large prime numbers **p** and **q.** Although multiplying two large prime numbers to get **n** is not difficult, it is computationally impracticable to reverse this procedure and recover **p** and **q** from the resultant **n.** Thus, the asymmetry of this difficulty is a reason for the secure use of RSA against cryptographic attacks.

Key Generation in RSA Using These Concepts

1. **Prime Number Selection:**

   Two distinct large prime numbers p and q are chosen. These primes must be kept secret as they form the basis of the security of RSA.

2. **Modulus Calculation:**

   The modulus n is computed as the product of p and q: $n=p\times q$. The modulus n is part of both the public and private keys.

3. **Euler's Totient Function Calculation:**

   The totient function $\varphi(n)$ is calculated: $\varphi(n)=(p-1)\times(q-1)$. This step uses the property of the totient function for the product of two primes.

4. **Public Key Exponent Selection:**

   A public exponent e is chosen such that $1<e<\varphi(n)$ and $gcd(e,\varphi(n))=1$. This means e must be a number that is relatively prime to $\varphi(n)$, ensuring it has an inverse modulo $\varphi(n)$.

5. **Private Key Exponent Calculation:**

   The private exponent d is computed as the modular multiplicative inverse of e modulo $\varphi(n)$. This means:

$$d \times e \equiv 1 \bmod \varphi(n)$$

   The above equation can be written as:

$$d = ((\Phi(n) \times i) + 1) / e$$

   We will iterate and increment the value of **i** by 1 each time until we find a value of **d** that is not in decimal.

ENCRYPTION PROCESS

To encrypt a plaintext message m, the sender uses the recipient's public key (n, e). The encryption process involves converting the plaintext message into an integer m within the range $0\leq m<n$ and computing the ciphertext c as $c = m^e \bmod n$.

DECRYPTION PROCESS

Upon receiving the ciphertext c, the recipient utilizes their private key (n, d) to decrypt the message. The original plaintext m is recovered by computing $m = c^d \bmod n$.

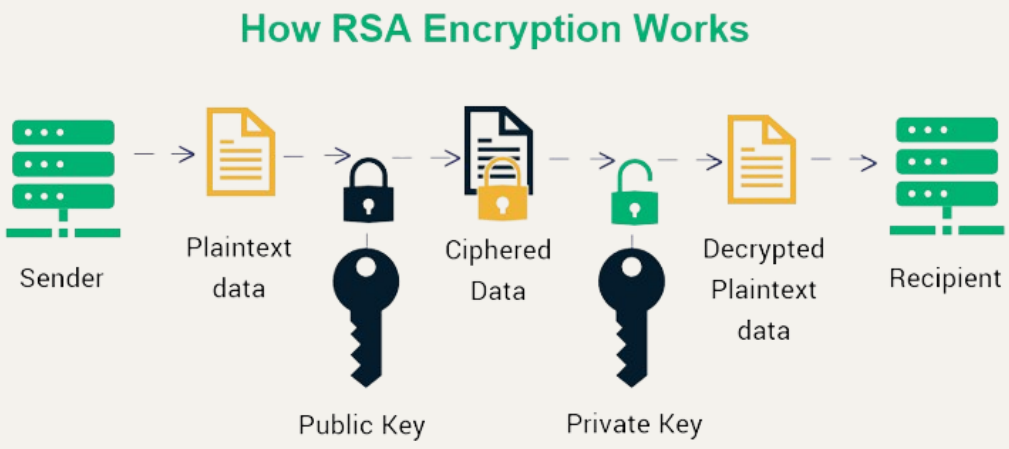

Figure 1: Working of RSA

Pseudocode for RSA Algorithm with Cost Analysis

Below is the pseudocode for the provided RSA algorithm implementation, along with the cost analysis for each function or section of the code.

Fermat Primality Test

```
function fermat_primality_test(number, k=5):
    if number <= 1:
        return False
    if number <= 3:
        return True
    for i from 1 to k:
        a = random_integer(2, number - 2)
        if pow(a, number - 1, number) != 1:
            return False
    return True
```

Cost Analysis

Each iteration of the loop: O (log n)

Total for k iterations: O (k (log n))

Generate Prime

```
function generate_prime(min_value, max_value):
    prime = random_integer(min_value, max_value)
    while not fermat_primality_test(prime):
        prime = random_integer(min_value, max_value)
    return prime
```

Cost Analysis

Average iterations to find a prime: O (log n)

Fermat test per iteration: O (k (log n))

Total: O (k (log n)^2)

Modular Inverse

```
function mod_inverse(e, Φ):
    for d from 3 to Φ - 1:
        if (d * e) % Φ == 1:
            return d
    raise ValueError("MOD_INVERSE DOES NOT EXIST!")
```

Cost Analysis

Brute-force search: O (Φ) = O (n)

Extended Euclidean algorithm (alternative): O(log n)

Generate Keys

```
function generate_keys():
    p = generate_prime(1000, 50000)
    q = generate_prime(1000, 50000)
    while p == q:
        q = generate_prime(1000, 50000)
    n = p * q
    Φ(n) = (p - 1) * (q - 1)
    e = random_integer(3, Φ(n) - 1)
    while gcd(e, Φ(n)) != 1:
        e = random_integer(3, Φ(n) - 1)
    d = mod_inverse(e, Φ(n))
    update_keys_display(False)
```

Cost Analysis

Prime generation: 2 x O (k (log n)^2) = O (k (log n)^2)

Finding e: O (log n)

Finding d: O (n) using brute force or O (log n) (with extended Euclidean algorithm)

Update Keys Display

```
function update_keys_display(show):
    tree.delete_all()
    rows = [("Prime Number P", p), ("Prime Number Q", q), ("Public Key", e), ("Private Key", d), ("N", n), ("Φ of N", Φ(n))]
    for index, (key, value) in enumerate(rows):
        display_value = value if show else "****"
        tag = 'odd' if index % 2 == 0 else 'even'
        tree.insert("", "end", values=(key, display_value), tags=(tag,))
    tree.tag_configure('odd', background='#ECF0F1', foreground='#17202A')
    tree.tag_configure('even', background='#D5DBDB', foreground='#17202A')
```

Cost Analysis

Constant-time operations per key-value pair: O (1)

Total for a fixed number of keys: O (1)

Encrypt Message

```
function encrypt_message():
    message = get_message()
    if message is empty:
        show_warning("Please enter a message to encrypt.")
        return
    message_encoded = [ord(ch) for ch in message]
    ciphertext = [pow(ch, e, n) for ch in message_encoded]
    ciphertext_str = " ".join(map(str, ciphertext))
    update_ciphertext_label(ciphertext_str)
```

Cost Analysis

Encoding message: O (L) where L is the length of the message

Encryption (modular exponentiation): O (L log n).

Decrypt Message

```
function decrypt_message():
    ciphertext_str = get_ciphertext()
    if ciphertext_str is empty:
        show_warning("No ciphertext available to decrypt.")
        return
    ciphertext = map_to_int(ciphertext_str.split())
    decoded_msg = [pow(ch, d, n) for ch in ciphertext]
    msg = "".join(chr(ch) for ch in decoded_msg)
    update_decrypted_text_label(msg)
```

Cost Analysis

Decoding message: O(L) where L is the length of the ciphertext

Decryption (modular exponentiation): O (L log n).

GUI Setup and Main Loop

```
create_main_window():
    create_labels()
    create_message_entry()
    create_buttons()
    create_ciphertext_frame()
    create_decrypted_text_frame()
    create_treeview()
    create_view_hide_buttons()
    seed_random_number_generator()
    generate_keys()
    run_main_loop()
```

Cost Analysis

GUI setup: O (1)

Random seed: O (1)

Initial key generation: O (k (log n)^2)

Summary of Costs

Key Generation: O (k (log n)^2)

Encryption: O (L log n).

Decryption: O (L log n).

GUI Setup and Main Loop: O (1) for setup, O (k (log n)^2) for initial key generation

Overall Cost

The overall cost of the RSA algorithm in the context of this implementation is dominated by the key generation phase, with a complexity of O (k (log n)^2). The encryption and decryption phases have a linear complexity with respect to the length of the message or ciphertext, both being O (L log n).

INDUSTRIAL APPLICATIONS

Data Transmission

RSA encryption is widely *used to secure data transmission over insecure channels* such as the internet. It ensures that sensitive information such as financial transactions, personal data, and confidential communications remain confidential and secure during transmission.

Digital Signatures

RSA encryption is employed for creating digital signatures, which provide authenticity and integrity verification for messages and documents. By signing a message with their private key, a sender can authenticate the message's origin and ensure its integrity. Recipients can then verify the signature using the sender's public key.

Secure Email

RSA encryption plays a crucial role in secure email communication. Protocols like PGP (Pretty Good Privacy) and S/MIME (Secure/Multipurpose Internet Mail Extensions) utilize RSA to encrypt email contents and verify digital signatures, ensuring confidentiality and authenticity.

SSL/TLS Protocols:

RSA encryption is integral to the SSL/TLS protocols used to secure web browsing. During the SSL/TLS handshake process, RSA is employed to establish a secure connection between the client and server, encrypting data transmitted over the network.

ADVANTAGES/PROS

Confidentiality

Only authorized parties with access to the private key can decrypt and access sensitive data.

Integrity

RSA encryption ensures that the transmitted data remains unchanged and unaltered during transmission.

Authentication

By securely distributing its public key, the financial institution can authenticate the origin of the encrypted data.

DEMONSTRATION

Example Key Generation

Select Primes:

$$p = 61,\ q = 53$$

Compute n:

$$n = 61 \times 53 = 3233$$

Compute $\varphi(n)$:

$$\varphi(n) = (61-1) \times (53-1) = 3120$$

Choose e:

$$e = 17$$

Compute d:

e x d = 1 mod $\varphi(n)$

$17 \times 2753 \equiv 1 \bmod 3120$

d = ((Φ(n) x i) + 1)/e

d = ((3120 x 1) + 1)/17 = 183.58

d = ((3120 x 2) + 1)/17 = 367.11

d = ((3120 x 12) + 1)/17 = 2202.41

d = ((3120 x 15) + 1)/17 = 2753

$d = 2753$

We will iterate and increment the value of **i** by 1 each time until we find a value of **d** that is not in decimal.

Example Encryption

Plaintext Message:

$$m = 65$$

Ciphertext Calculation:

$$c = 65^{17} \bmod 3233 = 2790$$

Example Decryption

Received Ciphertext:

$$c = 2790$$

Plaintext Recovery:

$$m = 2790^{2753} \bmod 3233 = 65$$

CONCLUSION

In conclusion, RSA encryption serves as a cornerstone in modern cryptography, providing robust security mechanisms for data transmission, digital signatures, and secure communications. Its mathematical foundation and practical applications make it an indispensable tool for ensuring confidentiality, integrity, and authenticity in various industrial settings. Through its principles, applications, and practical demonstration, RSA encryption continues to play a vital role in safeguarding digital information in the digital age.