

# Implementation of RSA Algorithm

---

**Network Security Assignment - CS1702**

**Submitted By**  
***Sairam S (CS22B1048)***

**To**

**Dr. Narendran Rajagopalan**  
***Associate Professor***  
***Department of Computer Science and Engineering***  
***National Institute of Technology Puducherry***  
***Karaikal – 609609***



**DEPARTMENT OF**  
**COMPUTER SCIENCE AND ENGINEERING**  
**NATIONAL INSTITUTE OF TECHNOLOGY PUDUCHERRY**  
**KARAikal – 609 609**  
**April 2025**

## Introduction to RSA:

**RSA (Rivest–Shamir–Adleman)** is among the earliest public-key cryptographic systems and remains widely used for secure communication. In this type of system, the encryption key is public, while the decryption key is private and confidential. RSA relies on the computational difficulty of factoring large composite numbers — a challenge known as the "factoring problem" — to maintain its security. The name "RSA" comes from its inventors: Ron Rivest, Adi Shamir, and Leonard Adleman, who introduced the algorithm in 1977. Interestingly, a similar method was developed earlier in 1973 by Clifford Cocks, a British mathematician at GCHQ, but it remained classified until 1997.

An RSA user generates and shares a public key derived from two large secret prime numbers and an auxiliary value. While anyone can encrypt messages using this public key, only someone with knowledge of the original prime numbers can decrypt them. The difficulty of breaking RSA encryption is known as the "RSA problem," though it's still uncertain whether this problem is exactly equivalent to the factoring problem. So far, no effective attacks have been made public against RSA when strong, properly-sized keys are used.

Due to its slower performance, RSA is not typically used for direct encryption of large amounts of data. Instead, it is often employed to securely exchange keys for symmetric encryption systems, which are much faster for bulk data processing.

## RSA Algorithm:

- Pick two large primes  $p, q$ .
- Compute  $n = pq$  and  $\phi(n) = \text{lcm}(p-1, q-1)$
- Choose a public key  $e$  such that  $1 < e < \phi(n)$  and  $\text{gcd}(e, \phi(n)) = 1$
- Calculate  $d$  such that  $de \equiv 1 \pmod{\phi(n)}$
- Let the message key be :  $m$
- Encrypt:  $c \equiv m^{**}e \pmod{n}$

- Decrypt:  $m \equiv c^{**d} \pmod{n}$

## RSA Code and Explanation:

INPUT:

```
1 import random
2
```

--Used to generate random prime numbers for key generation.

```
3 def gcd(a, b):
4     while b != 0:
5         a, b = b, a % b
6     return a
7
```

--Implements Euclidean Algorithm to find the greatest common divisor of a and b.

-- Used to ensure e is coprime to  $\phi(n)$ .

```
8 def mod_inverse(e, phi):
9     def extended_gcd(a, b):
10         if a == 0:
11             return (b, 0, 1)
12         else:
13             g, y, x = extended_gcd(b % a, a)
14             return (g, x - (b // a) * y, y)
```

--Inner function extended\_gcd: Recursively applies Extended Euclidean Algorithm.

--Finds integers x and y such that:  $a*x + b*y = \gcd(a,b)$ .

```
15     g, x, y = extended_gcd(e, phi)
16     if g != 1:
17         raise Exception("Modular inverse does not exist")
18     return x % phi
19
```

--If  $\gcd(e, \phi) = 1$ , returns  $x \pmod{\phi(n)}$  as the modular inverse d.

--Else, raises an error.

```
20 def is_prime(n):
21     if n <= 1:
22         return False
23     for i in range(2, int(n**0.5) + 1):
24         if n % i == 0:
25             return False
26     return True
27
```

--Checks if a number is prime using trial division.

--Needed to select good values for p and q.

```

28 def generate_keys():
29     p = q = 0
30     while not is_prime(p):
31         p = random.randint(100, 300)

```

--Keeps generating random numbers until it finds a prime p.

```

32     while not is_prime(q) or q == p:
33         q = random.randint(100, 300)
34

```

--Same for q, ensuring  $p \neq q$ .

```

35     n = p * q
36     phi = (p - 1) * (q - 1)
37

```

--Computes modulus n and Euler's totient  $\phi(n)$ .

```

38     e = random.randrange(2, phi)
39     while gcd(e, phi) != 1:
40         e = random.randrange(2, phi)
41

```

--Selects e such that  $\gcd(e, \phi(n)) = 1$  (ensures e has a modular inverse).

```

42     d = mod_inverse(e, phi)
43

```

--Computes  $d = e^{-1} \bmod \phi(n)$

```

44     return ((e, n), (d, n))
45

```

--Returns:

- **Public key:** (e,n)
- **Private key:** (d,n)

```

46 def encrypt(message, public_key):
47     e, n = public_key
48     return [pow(ord(char), e, n) for char in message]
49

```

--For each character:

- `ord(char)` : Convert to ASCII.
- `pow(base, exp, mod)`: Efficiently compute  $\text{char}^e \bmod n$ .

--Returns list of encrypted integers.

```

50 def decrypt(ciphertext, private_key):
51     d, n = private_key
52     return ''.join([chr(pow(char, d, n)) for char in ciphertext])
53

```

--For each encrypted number:

- Computes  $\text{char}^d \bmod n$ .
- Converts back to character with `chr()`.

--Returns the original message.

```
54 if __name__ == "__main__":
```

--Ensures the code only runs when the script is executed directly.

```
55     print("RSA Encryption/Decryption System")
56
57     message = input("Enter a message to encrypt: ")
58
```

--Gets input message from user.

```
59     public_key, private_key = generate_keys()
60     print("\nPublic Key:", public_key)
61     print("Private Key:", private_key)
62
```

--Generates and displays keys.

```
63     encrypted_msg = encrypt(message, public_key)
64     print("\nEncrypted Message:", encrypted_msg)
65
```

--Encrypts and prints the ciphertext.

```
66     decrypted_msg = decrypt(encrypted_msg, private_key)
67     print("Decrypted Message:", decrypted_msg)
68
```

--Decrypts and displays the original message.

OUTPUT:

```
(base) sairam@sairams-MacBook-Air ~ % python -u "/Users/sairam/Downloads/Assignments/NS_assign-1.py"
RSA Encryption/Decryption System
Enter a message to encrypt: O Captain! My Captain!

Public Key: (25321, 29503)
Private Key: (9601, 29503)

Encrypted Message: [14953, 2441, 7126, 14252, 5553, 21134, 14252, 19339, 23821, 18836, 2441, 29036, 225, 2441, 7126, 14252, 5553, 21134, 14252, 19339, 23821, 18836]
Decrypted Message: O Captain! My Captain!
```