https://linkedin.com/in/mousa123

# RSA Encryption: A Practical, straight to the point Guide

Made with dedication, by Mousa Emarah.

### 1. The Core Concept

RSA is a **public-key cryptosystem** that enables **secure communication** over insecure channels.

- **Public Key**: (n, e) — Used to **encrypt** data (can be shared freely)
- **Private Key**: d — Used to **decrypt** data (kept secret by the receiver)

The beauty of RSA lies in **asymmetry**: what one key encrypts, only the other can decrypt.

---

### 2. How RSA Works

#### 🔑 Step 1: Key Generation (The Setup)

To use RSA, we need to generate two keys — one public, one private. Here's how it's done:

**1. Choose Two Large Prime Numbers**

Select two large primes, p and q.
For strong security, each should be at least **1024 bits** long.

**2. Compute the Modulus**

$n = p \times q$
This n is used in both encryption and decryption.

**3. Calculate Euler's Totient Function**

$\phi(n) = (p - 1) \times (q - 1)$
This number is essential for generating the private key.

**4. Select a Public Exponent**

Choose a small odd integer e such that:
$\gcd(e, \phi(n)) = 1$ (i.e., e is coprime with $\phi(n)$)
**Common choice**: e = 65537 (fast and secure)

**5. Compute the Private Exponent**

Calculate d, the **modular inverse** of e modulo $\phi(n)$:

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

This is typically done using the **Extended Euclidean Algorithm**.

https://linkedin.com/in/mousa123

✅ At this point, your keys are ready:

- **Public Key**: (n, e) — You can share this with anyone
- **Private Key**: d — Keep this secure and confidential

---

### 🔒 Step 2: Encryption (Locking the Message)

To encrypt a message M:

$C = M^e \bmod n$

- M must be an integer **less than n**
- Large messages should be **split into blocks**

### 🔧 Efficient Computation: Square-and-Multiply

RSA involves large exponentiations; another problem is the buffer overflow. To avoid performance issues, we use 2 important techniques:

-Square and Multiply “Modular Reduction”

-CRT

---

**First:** Square and Multiply:

1. Convert e to binary
2. Loop through each bit starting from the second most significant bit (10100)
   - **Y = x , enter iteration (num of iterations is t-1)**
   - **square current y where y = y^2 mod n**
   - **Multiply by the base** if the bit is 1, if zero no action.
   - Apply mod n at every step to keep numbers manageable

This method dramatically improves performance and prevents Buffer overflows.

**Example:**
Encrypt M = 4 using e = 7, n = 21:
$C = 4^7 \bmod 21 = 4$

https://linkedin.com/in/mousa123

### 🔒 Step 3: Decryption (Unlocking the Message)

To decrypt a ciphertext C:

M = $C^d$ mod n

Because d is large, decryption can be slow. We use the **Chinese Remainder Theorem (CRT)** to optimize.

⚡ **CRT Optimization,** Originally we have:

X^d mod n

But CRT lets us transform this into a different domain by dividing the problem into two parts:

Xp^dp mod p

Xq^dq mod q

This effectively means we compute the result in mod p and mod q separately, which takes significantly less time because the numbers involved are much smaller. Don't forget: always ensure X < n so that taking X mod n remains smooth and correct.

Remember:
n = p × q

This CRT trick makes decryption around 4x faster than square-and-multiply because the operands are smaller, reducing the computational cost.

📌 Note: If you're working with a modulus n that's 1024 bits or more and your message X is bigger, please segment the message before encrypting.

📌 Extra Tip: As explained by the professor, choosing e = 2^n + 1 is better in practice because it reduces the number of multiplications during encryption.

**Example:**
Given: C = 4, d = 7, p = 3, q = 7

- mp = 4^1 mod 3 = 1
- mq = 4^1 mod 7 = 4
- h = (4 - 1) × 5 mod 7 = 1 (since $3^{-1}$ mod 7 = 5)
- M = 1 + 3 × 1 = 4

Original message recovered ✅

https://linkedin.com/in/mousa123

### 3. Why RSA Is Secure

RSA's security comes from the **difficulty of factoring** large numbers:

- ✅ It's **computationally hard** to factor n = p × q if p and q are large enough
- ✅ No known efficient attack exists if **key length ≥ 2048 bits**
- ✅ While quantum computers threaten RSA, most symmetric ciphers can still survive with longer keys

**Conclusion:**

For now, RSA remains one of the most trusted encryption algorithms in real-world systems. it's behind **SSL/TLS**, **email encryption**, and **digital signatures**. By understanding how it works at the core, you can use it more confidently and recognize its strengths and limits.

| **Challenge** | **Solution** | **Benefit** |
|---|---|---|
| Large exponents | Square-and-Multiply | Efficient encryption |
| Slow decryption | Chinese Remainder Theorem (CRT) | 4x faster |
| Large messages | Split into chunks < n | Full message coverage |