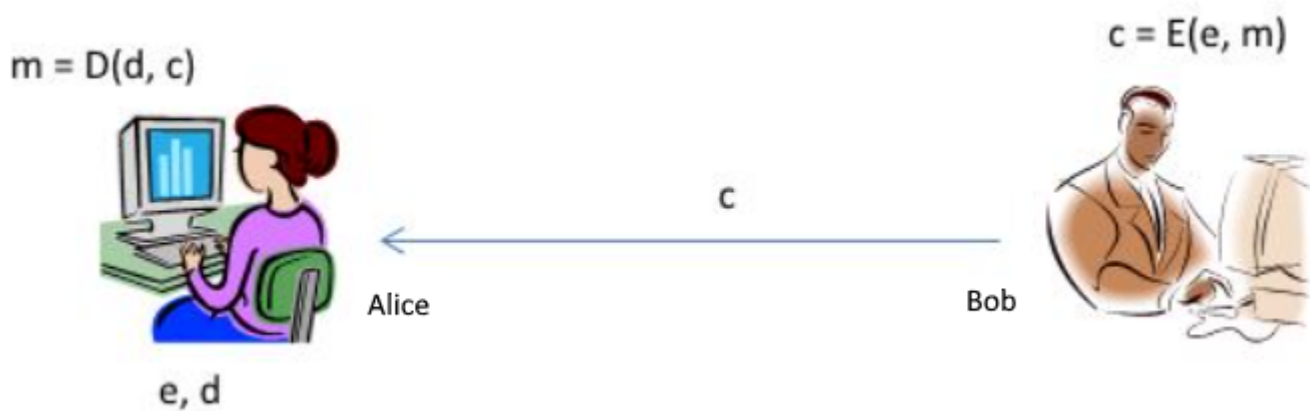


## The Basic Model of Public Key Systems

### Basic communication model



1. Alice generates a pair of keys: **e** (public key) and **d** (private key). In this context **e** means (encryption key) and **d** (decryption key).
2. She keeps **d** secret, but makes **e** public.
3. If Bob wants to send a message to Alice, he uses Alice's public key **e**.
4. Based on the equation  $( c = E(e, m) )$ , only Alice can decrypt  $( c )$  using her private key, with  $( m = D(d, c) )$ , where  $( m )$  is the message.
5. If anyone else wants to send a message to Alice, they can also use her public key **e**.

The system is secure from a decryption perspective because only Alice can decrypt the message, but Alice can never be sure if Bob sent the message, as the public key **e** can be used by anyone.

## RSA

**Rivest, Shamir, Adleman (1977)** developed the RSA algorithm, which is based on exponentiation and modular arithmetic (remainder division). The RSA encryption relies on the fact that if the following equation holds:

$$( T^d \pmod N = T )$$

Then the equation can be split into two parts, where the first equation represents encryption and the second represents decryption:

- **Encryption:**  $( C = T^e \pmod N )$
- **Decryption:**  $( T = C^d \pmod N )$

However, this relationship does not work for just any arbitrary choice of  $( e )$ ,  $( d )$ , and  $( N )$ . The algorithm relies on specific conditions for these values. Let's explore what conditions are required for the RSA algorithm to function correctly.

## How RSA Works

The RSA algorithm uses both **public** and **private keys**. For it to work correctly, the following steps are followed:

### 1.) Key Generation:

- Choose two large prime numbers:  $(p)$  and  $(q)$ .
- Calculate  $(N = p \times q)$ .
- Compute Euler's totient function  $(\phi(N) = (p-1) \times (q-1))$ . ( [totient function](#) )
- Choose a public exponent  $(e)$  such that  $(e)$  is relatively prime to  $(\phi(N))$  (i.e.,  $(1 < e < \phi(N))$  and  $(\gcd(e, \phi(N)) = 1)$ ).
- Compute the private key  $(d)$ , which is the modular multiplicative inverse of  $(e)$ , meaning  $(d \times e \equiv 1 \pmod{\phi(N)})$ .

### 2.) Encryption:

The message  $(T)$  is encrypted using the public key:  $(C = T^e \pmod N)$ .

### 3. Decryption:

The ciphertext  $(C)$  is decrypted using the private key:  $(T = C^d \pmod N)$ .

## Conditions for the Keys

The equations work correctly only when specific conditions are met:

- $(N)$  must be large enough to prevent attackers from easily factoring  $(N)$  into  $(p)$  and  $(q)$ , which would compromise the encryption.
- The public exponent  $(e)$  and the private exponent  $(d)$  must have a specific relationship with  $(N)$  and Euler's totient  $(\phi(N))$ . The relationship between  $(e)$  and  $(d)$  is that  $(e \times d \equiv 1 \pmod{\phi(N)})$ , ensuring that decryption reverses the encryption process.

## Summary

The RSA algorithm depends on the correct choice of prime numbers  $(p)$  and  $(q)$ , the calculation of  $(N)$ , and the mathematical relationship between the exponents  $(e)$  (public key) and  $(d)$  (private key). Without these specific conditions, the encryption and decryption process will not work properly.

## Example

### Step 1: Key Generation

1.) **Choose two large prime numbers**  $(p)$  and  $(q)$ :

- $(p = 61)$
- $(q = 53)$

2.) **Compute  $(N)$**  (the modulus):

$$N = p \times q = 61 \times 53 = 3233$$

So,  $(N = 3233)$ .

3.) **Compute Euler's totient function  $(\phi(N))$** :

$$\phi(N) = (p - 1) \times (q - 1) = (61 - 1) \times (53 - 1) = 60 \times 52 = 3120$$

So,  $(\phi(N) = 3120)$ .

4. **Choose a public exponent  $(e)$** , such that  $(1 < e < \phi(N))$  and  $(\gcd(e, \phi(N)) = 1)$ :

- Let's choose  $(e = 17)$ , which is relatively prime to 3120.

5. **Calculate the private exponent  $(d)$** , which is the modular multiplicative inverse of  $(e \pmod{\phi(N)})$ :

$$d \times e \equiv 1 \pmod{\phi(N)}$$

Using the extended Euclidean algorithm, we find that  $(d = 2753)$ .

## Step 2: Encryption

Now that the keys are set, let's encrypt a message. Suppose our message is a number  $(m = 65)$ .

1.) **Public key** is  $(e, N) = (17, 3233)$ .

2.) **Encryption formula:**

$$c = m^e \pmod{N}$$

\*  $(m = 65)$ ,  $(e = 17)$ ,  $(N = 3233)$ .  
 \* Compute  $(65^{17} \pmod{3233})$ :

Using modular exponentiation:

$$\backslash$$

$$65^{17} \pmod{3233} = 2790$$

So, the **ciphertext**  $(c = 2790)$ .

### Step 3: Decryption

Now, Alice receives the ciphertext  $(c = 2790)$  and uses her private key  $(d = 2753)$  to decrypt the message.

1.) **Private key** is  $(d, N) = (2753, 3233)$ . 2.) **Decryption formula:**

$$m = c^d \pmod{N}$$

\*  $(c = 2790)$ ,  $(d = 2753)$ ,  $(N = 3233)$ .  
\* Compute  $(2790^{2753} \pmod{3233})$ :

Again, using modular exponentiation:

$$2790^{2753} \pmod{3233} = 65$$

Thus, Alice successfully decrypts the message and retrieves the original plaintext **65**.

### Summary of the RSA Example:

- Public key:  $(e = 17, N = 3233)$
- Private key:  $(d = 2753, N = 3233)$
- Message to encrypt:  $(m = 65)$
- Ciphertext:  $(c = 2790)$
- Decrypted message:  $(m = 65)$

This is a simple RSA example. In real-world applications, much larger prime numbers  $(p)$  and  $(q)$  are used to ensure security.

From: <https://edu.iit.uni-miskolc.hu/> - Institute of Information Science - University of Miskolc

Permanent link: [https://edu.iit.uni-miskolc.hu/tanszek:oktatas:techcomm:rsa\\_encryption?rev=1728307735](https://edu.iit.uni-miskolc.hu/tanszek:oktatas:techcomm:rsa_encryption?rev=1728307735)

Last update: **2024/10/07 13:28**

