🔐 Tutorial 5: Asymmetric Encryption

In this tutorial, we will explore and complete hands-on tasks related to **asymmetric key encryption**.

🔑 1) Number of Keys

Let's examine the number of keys required in a cryptosystem.

a) Symmetric Case

Assume there are **n** users in a fully decentralized (mesh) system.
👉 How many symmetric keys are needed to ensure confidentiality between all participants?

b) Asymmetric Case

If **public-key cryptography** is used for encryption instead of symmetric cryptography,
👉 how many keys are required when there are **n** users in the system?

2) SSH Keys

Currently, you log in to the VM using a username and password. In this task, we will create a new user on the VM, generate a new private and public key pair, and test logging into the machine using the keys.

a) Set up port forwarding in VirtualBox

You can add the port forwarding rule without shutting down the VM.

With the VM running:

- In the VM window, go to **Devices → Network → Network Settings...**
- Select **Adapter 1** (should be attached to **NAT**) → click **Advanced** → **Port Forwarding**.
- Add a new rule: Host Port `2222` → Guest Port `22`, Host IP `127.0.0.1`.
- Click **OK**.

b) Create a new user on your VirtualBox VM

- Use the command: `sudo adduser sec_user`
- Next, you will be prompted to enter a password. Use one that you will remember or note it down.
- Then, it will ask for some information like Full Name, Room Number, etc. You can simply press Enter to leave these blank.

- Now, switch to the newly created user by running: `su sec_user`
- If successful, you should be logged in as the new user (i.e., your command prompt will look like `sec_user@<hostname>:/home/ubuntu$`). You will be asked to enter the password.
- Create a hidden directory called `.ssh` in the `sec_user`'s home directory by running: `mkdir ~/.ssh`. Here, `~` is a shortcut to the user's home directory (i.e., `/home/sec_user/`).
- Verify that the hidden directory was created:
  - Navigate to the home directory using either `cd ~/` or `cd /home/sec_user`
  - Run `ls -ltr` — you will not see the `.ssh` directory because normal `ls` options do not show hidden files or directories.
  - Next, run `ls -lart` — this will show the hidden `.ssh` directory.

c) On your local machine (host), create a private and public key pair using **ssh-keygen**

For example, on a Mac, you can do the following in the command line:

```
mac_user@mac CSEC5616 % ssh-keygen -t rsa

Generating public/private rsa key pair.

Enter file in which to save the key
(/Users/mac_user/.ssh/id_rsa):

Enter passphrase (empty for no passphrase):

Enter same passphrase again:

Your identification has been saved in
/Users/mac_user/.ssh/id_rsa

Your public key has been saved in
/Users/mac_user/.ssh/id_rsa.pub
```

Note the following:

- You will be asked to create a passphrase. This adds extra security to the key. You can leave it empty by pressing Enter.
- The command creates two files: **id_rsa**, which is the **private key**, and **id_rsa.pub**, which is the **public key**.
- These files are saved in a directory similar to before (i.e., `/Users/mac_user/.ssh/`).

Windows follows a similar process as described here:
https://phoenixnap.com/kb/generate-ssh-key-windows-10

d) Copy your public key to the VM

Since you're connecting to `127.0.0.1` on a custom port (`2222`) instead of a normal hostname/IP on port 22, you need to tell `ssh-copy-id` (or your manual copy command) about the custom port:

```
ssh-copy-id -p 2222 sec_user@127.0.0.1
```

If `ssh-copy-id` is not available (e.g. on some Windows setups), you can copy the contents of `id_rsa.pub` manually and append it to `/home/sec_user/.ssh/authorized_keys` on the VM. If there is no file called `/home/sec_user/.ssh/authorized_keys` in the VM, you can create one and add the public key.

```
cat ~/.ssh/id_rsa.pub | ssh -p 2222 sec_user@127.0.0.1 "mkdir -p ~/.ssh && cat >> ~/.ssh/authorized_keys"
```

e) Test logging in using your key pair

```
ssh -p 2222 sec_user@127.0.0.1
```

If everything is set up correctly, you should be logged in **without** being prompted for `sec_user` 's password (unless you set a passphrase on your key, in which case you'll be asked for the passphrase instead).

5. Understanding SSH Authentication and Server Trust

When you connect to a remote server using SSH with public key authentication, your private key on the client is used to prove your identity by responding to a challenge from the server.

a) Explain why it is safe to share your public key with the server but not your private key.

b) Consider the scenario where an attacker sets up a fake server impersonating the legitimate server. Discuss what risks this poses to the client during the SSH connection.

c) Describe how the SSH client can protect itself from connecting to an impersonating (malicious) server.

d) What could happen if the client ignores warnings about changes in the server's identity (e.g., "REMOTE HOST IDENTIFICATION HAS CHANGED")?

Bonus: Suggest best practices users should follow to maintain secure SSH connections.

3) Textbook RSA

Here, we will perform a simple RSA calculation by hand and, if needed, using a bit of programming.

You are given the following RSA parameters:

- n = 3127
- e = 3
- d = 2011
- p = 53
- q = 59

a) Keys

- What is the **public key**?
- What is the **private key**?

📝 **Note: The public key is composed of (e, n) and the private key is composed of (d, n).**

b) Euler function and co-primality

Euler found a neat formula to compute the number of integers that that are co-prime to a given integer n:

$\phi(n) = n\prod_{i|n}(1-\frac{1}{i})$, where $i|n$ means the (distinct) prime divisors of n.

- What is the definition of co-prime?
- Write down all numbers to which 1 is co-prime.
- Verify the validity of this equation using 8 and 15
- Using the above equation, show that $\phi(n) = (p-1)\cdot(q-1)$ when p, q are prime and $n = pq$.

b) Euler's Totient Function and Co-Primality

Euler derived a neat formula to compute the number of integers that are co-prime to a given integer (n):

$\phi(n) = n\prod_{i|n}(1-\frac{1}{i})$, where $i|n$ means the (distinct) prime divisors of n.

- What is the definition of **co-prime**?
- Write down all numbers to which 1 is co-prime.
- Verify the validity of this equation using $n = 8$ and $n = 15$.
- Using the above equation, show that $\phi(n) = (p-1)\cdot(q-1)$ when p, q are prime and $n = pq$.

c) Safekeeping

Which information should be securely destroyed after generating the key?

d) Key length

- Does this key pair look secure for use over the next five years?

- What is the key length in bits?

e) Manual encryption

Encrypt the letter **A** using the given RSA parameters. **Note:** 'A' corresponds to 65_{10} in ASCII. Check your solution by decrypting it again.

4) Implementing RSA

Next, we implement a simple version of RSA using Python. This is a basic version and **is not secure** for practical use. In practice, you must use established cryptographic libraries.

- Most of the code is provided. You only need to complete a few parts.
- Your tutor will guide you through the key ideas behind the code.
- Complete the code as instructed.
- Try a simple encryption and decryption.
- Note that in some cases, this simple implementation may return the same value for both e and d. This is not recommended for real applications.

In [3]:
```python
import math
import os
import random

MAX_LENGTH = 20  # bits

## TODO: Function for calculating the GCD using the Euclidean Algorithm
def gcd(a, b):
    # TODO
    return a

# Function for EGCD
# Returns (g, x, y) such that a*x + b*y = gcd(a, b)
# Already completed
def egcd(a, b):
    if a == 0:
        return (b, 0, 1)
    else:
        g, x, y = egcd(b % a, a)
        return (g, y - (b // a) * x, x)

# Function to find the multiplicative inverse
# Already completed
def mulinv(b, n):
    g, x, _ = egcd(b, n)
    if g == 1:
        return x % n
    else:
        return -1

# TODO - Complete the function to compute d when e and phi are known
def compute_d(e, phi):
    # TODO
    return d
```

```
def create_key():
    # We are sticking to two small prime numbers for demonstration. In re
    p = 7
    q = 31
    phi = (p - 1) * (q - 1)
    e = random.randint(1, phi)
    g = gcd(e, phi)
    while g != 1:
        e = random.randint(1, phi)
        g = gcd(e, phi)
    d = compute_d(e, phi)
    return p, q, phi, e, d

def encrypt(pub_key, plaintext):
    e, n = pub_key
    x = []
    for i in plaintext:
        if i.isupper():
            m = ord(i) - 65
            c = (m ** e) % n
            x.append(c)
        elif i.islower():
            m = ord(i) - 97
            c = (m ** e) % n
            x.append(c)
        elif i.isspace():
            x.append(400)
    return x

def decrypt(priv_key, ciphertext):
    d, n = priv_key
    x = ''
    for i in ciphertext:
        if i == 400:
            x += ' '
        else:
            m = (i ** d) % n
            m += 65
            c = chr(m)
            x += c
    return x

## TODO - Do the following inside the main function:
## create the key
## print the RSA parameters
## encrypt and decrypt a simple text using the two functions.

def main():
    # TODO

main()
```

```
p = 7
q = 31
phi = 180
e = 169
d = 49
Ciphertext: [165, 39, 16, 195, 24, 190, 152, 17]
Decrypted text: SECURITY
```