Objective of this notebook : illustrate the use of RSA keys to sign and encrypt

This example is freely inspired from https://medium.com/@Raulgzm/rsa-with-cryptography-python-library-462b26ce4120 .

Maria and Raul want to communicate through an insecure channel. They choose to use RSA system to encryt and signe their messages.

They decide to use the python module Cryptography (see https://cryptography.io). If this module is not installe, please, run the next cell.

```
!pip3 install 'cryptography'
```

```
Collecting cryptography
  Downloading https://files.pythonhosted.org/packages/b2/26/7af637e6a7e8725
     |████████████████████████████████| 3.2MB 8.2MB/s
Requirement already satisfied: cffi>=1.12 in /usr/local/lib/python3.7/dist-
Requirement already satisfied: pycparser in /usr/local/lib/python3.7/dist-p
Installing collected packages: cryptography
Successfully installed cryptography-3.4.7
```

Module loading

First, they need to load the necessary python modules :

```
from cryptography.hazmat.backends import default_backend
from cryptography.hazmat.primitives.asymmetric import rsa
from cryptography.hazmat.primitives.asymmetric import padding
from cryptography.hazmat.primitives import hashes
from cryptography.hazmat.primitives import serialization
from cryptography.hazmat.primitives.serialization import load_pem_public_key
from cryptography.exceptions import InvalidSignature
```

Key generation

Maria wants to send a message signed and encrypted to Raul. Thus, they decide to generate their pair of RSA keys.

Maria generates her pair of keys.

```
maria_private_key = rsa.generate_private_key(
 public_exponent=65537,
```

✓ 0 s terminée à 23:29

To share her public key with other people, she serializes it, i.e. she transforms it as a butes array. Then, we assume that she sends her public key to Raul.

```
pem_maria_public_key = maria_public_key.public_bytes(
    encoding=serialization.Encoding.PEM,
    format=serialization.PublicFormat.SubjectPublicKeyInfo
)
```

Raul does the same operations.

```
raul_private_key = rsa.generate_private_key(
 public_exponent=65537,
 key_size=2048,
 backend=default_backend()
)
raul_public_key = raul_private_key.public_key()

pem_raul_public_key = raul_public_key.public_bytes(
    encoding=serialization.Encoding.PEM,
    format=serialization.PublicFormat.SubjectPublicKeyInfo
)
```

Key deserialization

Maria wants to send a message to Raul, so she must use his public key. Thus she first loads it from the serialized form.

```
loaded_raul_key = load_pem_public_key(pem_raul_public_key)
```

Encrypting

Now, she can encrypt her message with the Raul's public key.

```
message = 'the side must be like a piece of music'
message_bytes = message.encode()

ciphertext = raul_public_key.encrypt(
      message_bytes,
      padding.OAEP(
            mgf=padding.MGF1(algorithm=hashes.SHA256()),
```

derived key.

Signing

Then, Maria wants to sign her message. Thus, she uses her secret key.

```
signature = maria_private_key.sign(
    message_bytes,
    padding.PSS(
        mgf=padding.MGF1(hashes.SHA256()),
        salt_length=padding.PSS.MAX_LENGTH
    ),
    hashes.SHA256()
)
```

Then, she sents the encrypted message and the signature.

```
data = (ciphertext, signature)
```

Decrypting

Raul receives the data. First, he must decrypt the ciphertext with his secret key.

```
plain_text = raul_private_key.decrypt(
    ciphertext,
    padding.OAEP(
        mgf=padding.MGF1(algorithm=hashes.SHA256()),
        algorithm=hashes.SHA256(),
        label=None
    )
)

# to have a string version
plain_text_string = plain_text.decode()
```

Then, he verifies the signature with the public key of maria. He first loads the key from the

```
            ,,
            hashes.SHA256()
        )
    except InvalidSignature :
        print("Invalid signature")
```