Data Engineering 2 – Home Assignment 1

Authors: 2402779 & CEU ID #2

Generating keys

First, we need to generate our ceu_key keypair. We can do this with the following shell script:

```
ssh-keygen -t rsa -f "$(pwd)/ceu_key" -N ''
```

This command generates the key pairs in the current working directory.

Inspecting the keys and passing the public key to the website visitor

Then, with a Python script, we can write out the keys to the screen:

```
from pathlib import Path
from Crypto.PublicKey import RSA
#define our key files
pr_key_file = "ceu_key"  #private key
pub_key_file = "ceu_key.pub"  #public key
#checking if the keys really exist
assert Path(pr_key_file).exists(), f"Private key file {pr_key_file} does not
exist!"
assert Path(pub_key_file).exists(), f"Public key file {pub_key_file} does not
exist!"
#loading the private key from the file
with open(pr_key_file, "r", encoding="utf8") as key_file:
    private_key = RSA.import_key(key_file.read())
#printing out the keys
public_key = private_key.publickey()
print(f"Public key:\n{public_key.export_key().decode('utf-8')}")
print(f"Private key:\n{private_key.export_key().decode('utf-8')}")
```

The two keys that were printed out are the following:

Public key:

```
-----BEGIN PUBLIC KEY-----
MIIBojANBgkqhkiG9w0BAQEFAAOCAY8AMIIBigKCAYEAthNpahyGJeCSUOIqHTcU
5ygNGYHK+9uj00PSpHghP7oN2l2NG9uQtJdgnuFD4CjvR86CqKYli2JjEH00YW9W
MywEqJX1YWACXjeFrl/26XKAdkhebydG8s4TdoJXXpv81N4IUZhuXvMlG5qcSGRL
bbvwO7s5B0/zm5WZ/0ZZEXOQWclVxEaM4JRx3YY8ivk4eQ1cRyYlIGO2qKflSQTX
mZBSrMbzMqJaU7QJHso16KqxbJWJumQO0W5VQgtcNiS/BPx8ITHMg9tCt17kfRaA
zWz085UieR+R+0qLfln8t11cNuRbnXCgwUnc5VN4DDk60EG60r5thlh9xIpbr3ZM
Ex8bQtrmG8IMLjFXKbnuZgAANZAfkGjEsLI59HHG9F7jufFurH0wN4vm/r9l6FtK
cPrDP8Th0+8W1xy+wq13zamcnZP8erfmO/IeUe81+6l+Z+1DuGmsOKlGjrHuQe85
```

```
7r+GF5/+fJSL1Xq3ia/NHjTiyuqj+XwFz//DLEk/E6nxAgMBAAE=
-----END PUBLIC KEY-----
```

Private key:

```
-----BEGIN RSA PRIVATE KEY-----
MIIG5AIBAAKCAYEAthNpahyGJeCSUOIqHTcU5ygNGYHK+9uj00PSpHghP7oN2l2N
G9uQtJdgnuFD4CjvR86CqKYli2JjEH00YW9WMywEqJX1YWACXjeFrl/26XKAdkhe
bydG8s4TdoJXXpv81N4IUZhuXvMlG5qcSGRLbbvwO7s5B0/zm5WZ/0ZZEXOQWclV
xEaM4JRx3YY8ivk4eQ1cRyYlIGO2qKflSQTXmZBSrMbzMqJaU7QJHso16KqxbJWJ
umQO0W5VQgtcNiS/BPx8ITHMg9tCt17kfRaAzWz085UieR+R+0qLfln8t11cNuRb
nXCgwUnc5VN4DDk60EG60r5thlh9xIpbr3ZMEx8bQtrmG8IMLjFXKbnuZgAANZAf
kGjEsLI59HHG9F7jufFurH0wN4vm/r9l6FtKcPrDP8Th0+8W1xy+wq13zamcnZP8
erfmO/IeUe81+6l+Z+1DuGmsOKlGjrHuQe857r+GF5/+fJSL1Xq3ia/NHjTiyuqj
+XwFz//DLEk/E6nxAgMBAAECggGAAI5HT9PrhzABIM2Gk9UVTWjCGutjs0cAHk8d
ewsyMqOH4SAWKa9JTLq0DEB1rt0oEK3SrWsWzBDVG53rsXTQTMrbVi49nr9bvLo3
27KGqvXd4waLKnTkXVrV1b+uNwqyo7GhHopRn23U8seRNidI1o4kz3ZHEoSo/9Ui
mOnX5MAdbT28V9VU5nQcBGnI7c/zEBTL6Cth+Rexppj1kqoyQUvJJg4FKXybiT/S
OkL4ArG/qX4epCglvsyy0cVSu0KQRAwf+g+0i02TUlBncerG8m6iQOx6QIib72QD
huklNkFcNJ/+m1XXwQjfgRvQZzhWVkE7g6tA0NXaFBFQdbGpjOGlVhX+fadK79cM
iwi5kqoacybz2HCJHeBRoehhCXzFnuL/DXM2U2A4FOi5K80TQ2Xzg/B+jMG2eiXp
3xMXSEdQrBu4pO7R1npB0t0KzEDKOIGBmbC6HcSQH3FJzqxvtOukKt9Zc99jlGKR
tWMYklXQRLoKRv8hecFmiAbzdWJhAoHBAMshp+7aC3/ppFvx2b6nsy0UQhISPQyw
PLdeRnsGTA7rRrdjrO/RICbUP37lpiYnu1XubBF1a165OcLeygxqEu6OdXy/zqY5
8LxdRdoWVjzPNLwVgnWpiRjDNXggj2HsvU2+z+C8MzrqHD/vmtL0XbCD+5wn9O6v
Is5vy7JatHuWH4epoX2pCbaQFpqnS/Cht2AWgtCIdr5s38pULzFlyVYZUSkDdxL9
Ik0P+fkeACeUZwRzzYrkt8obmDp03R+d4QKBwQDldtmfcZWi59jsEm5Le/UjIleZ
z7dfiu3Hm8+HPoMihGEMF8jtZkDf0HTHc82mIiyIYkDw4mOTZ8QKk8K+dOzr6dIO
Cs8eOZFlqUDZkHDaZBOHwX+lGMR/Q62MUBVT4bxX/xPTKe68gV1bD8T+HkC0WjD7
bJyhiqXaqWovQhuVjKNJlmtVSkjE0DYN12G3b9LiayDDVAgFomUJvk1in8pBLOBK
PkMupVtqhUv1XECNocRXdxKH1QFvGjOqbeOT7hECgcEAxtXoA2bOcOQsbY/8u6J+
QjcdQYE23y+4DyoqRYxxcP0e2K6p/omvNjL3AGkdTSYBO2lJwYE1m6AmCTl5f9Np
OriaCoXaa1415rxKfuL0gUu2bBGGBVTxjRqwQSlmEM3is7J+25Z9c0Lsai1JWQu4
letrpHx8RhOLN5W5R5mAJ6VYsbv7Bv0rM9gxOCtgq2gxDs6aODQMP/RkzzG+jFT9
UtkvV707lovQQqzL3O1f615ZxMLyRO4DdcOzLC1usd8hAoHBAKg4OhBOp8F2sKtY
U61Y4XxxV8E16xvK8MiN7FUcuewbGj5QTYfkl5i87G+v8MpjcTxGs48kmJVe0/Rh
ILqZY3sLvmd2+yIQWAwsSZN19ZXVGhBDBb3V62/VAKzFpO0KpxXntPPwYMmnGPaC
GAunyA2vtQsNM8KlrzMfUe31S92V7bsr3+H2BGTss0Pwav2cqAA/QxSPTRY8WFAN
```

```
SBQOSqr/KCqlfID8zojH0ci9acGrHxJ0A4y61kNJ9ShzSQyQ0QKBwH7faCa5WdDV
K7mQdtvYhtM508sfb3bGfcgflEVIL4J93dINMWIEWUO+F8P3csnJY12xf8WZ6+DZ
HWCexG0RfhpbA8wBHyv/6tSXhxPwIIfK4PyRGFvjcmMxwBSkgNKGAqEkkzhBqGqf
qS5FQdwNKuF6qevkr8VMl9ahcPWqlD7t6EowUMqq/CjtI0puqrkd3K6wLYWAmgaj
/PbN9CunjzZ3D6mMvRMnHQXjoI2qwXPZjTvDrhyy/H4RUfPb20S4+A==
-----END RSA PRIVATE KEY-----
```

Then, the public key (ceu_key.pub) was sent over to the website visitor through Teams.

Encrypting a message with the public key

Using the sent ceu_key.pub, we can create an encrypted message only the server can decrypt. The message is: PöttyösPenne. The encryption is done through the following Python code:

```python
from Crypto.PublicKey import RSA
from Crypto.Cipher import PKCS1_OAEP
#message to be encrypted
msg = "PöttyösPenne".encode("utf-8")
with open('ceu_key.pub', "rb") as f:
    received_pubk = RSA.import_key(f.read())
#create a cipher object using the public key for encryption
public_key_cipher = PKCS1_OAEP.new(received_pubk)
#encrypt the message
encrypted_message = public_key_cipher.encrypt(msg)
print(f"Encrypted message:")
print(encrypted_message)
#save the encrypted message to a file
with open('encrypted_message.bin', "wb") as f:
    f.write(encrypted_message)
```

The encrypted message is saved into the encrypted_message.bin file. This file has also been attached to our submission email. The encrypted message reads (in binary) as[1]:

```
i-\xa2\xaf/\xa9l\xecq\x0f\xddR\x8f\xffH\x1a\x08O7\xad L2\x00\n\x99\x87\x01zB{\x
ccn=&}\xca\xea\xac\x8bN\xa2r8A\x99M\xf3\xd4\xfc\xc9\x90\x06\x0eG\xd1\x8a&\xe9)
\x85-\x84Jc\x10&C\xc3O+i\xf6\xb9o\x1aj\x9b\x9a\xcb\xb7\xb5\x80\xa8\x99\xc3I\xae
\x01`\xfdhgMO\x90\x9f\xe6\xc8\xbd\t\x89A\xf5\x95$H\x8b\xafR\r\x9f\xe0\x9b\xbcP
~\xf4\xd2\xa3O\xae\xfc\xf33\xde\xf3\x87\xbdQ\x9fH=\x8d\xae\xdbeJ\x80\xe6\n\xbb
p\xf0\x87{\xd0\x0b\x9eJ\xfaL\x8f\xb4/\x17\xca\xfc\x13X\xd4\xc6\x90B\xe8\xb8z\x
8e\xf6P\x05\xec\x01\x82\xc6@a\xf0\x08q\t\xae;\xdc\xf6\xc4\x93b\xb9\x98)\xbc\x8
9\x1c\xff\xaeWd\x93\rm\xb3a\x8f^-|\x0f~l\xa2\xa5Nw\xc1\xa6\xb6\xbc\xfa\x1bz\x13
%\x01\xa94\x92e\x99\x87\x18}\xea\x077\xb8\xdb\xe8>\xac8\xd9\x1fh\xa8Y\xb2\xed\
xec\xd6\xc1\x0co\x06\xe6\xd0o\xcf\xf1L.e2\x97\xf2\xd7\xab\xab\xdd\x19\xaa\x14S
\x95~\xfc\xbe\x83\xe3\xd6\x89\xd0\x95pdVhd\xf9\n(\xb9P\x04\xbal\xd7\x8b\xf3\x9
6A\x0b:P\x9e\x16\xa2\x10\xa4\x1f\x98\x0f\x88\xbe>/_\xc3O/\xcf_\xd9O\xb9\xc6hnk
```

[1] Note that hyphens and spaces have been replaced in this document to their non-breakable equivalents for better formatting, so trying to decrypt this sequence would result in errors. However, the attached encrypted_message.bin file contains the original characters.

```
M\xfb8\xa9\xf0\xa4\x8b\x8b\xa3\xe9\x1b0\r\xcfTi9H\x95\x15\xcbB\xf7\xb4\xa4\xb8
\xae\xfe]^<\xf2\xc5\xe3\r\xb3\x04\x04\xbf=N\xd4J\xc3\xae8\xd2i\xfd\x11\xadNv\x
94W
```

Decrypting the message

After receiving the encrypted message (encrypted_message.bin), we can decrypt it using the private key. This is done through the following Python code:

```
from pathlib import Path
from Crypto.PublicKey import RSA
from Crypto.Cipher import PKCS1_OAEP
#define our key files
pr_key_file = "ceu_key"  #private key
pub_key_file = "ceu_key.pub"  #public key
#checking if the keys really exist
assert Path(pr_key_file).exists(), f"Private key file {pr_key_file} does not
exist!"
assert Path(pub_key_file).exists(), f"Public key file {pub_key_file} does not
exist!"
#loading the private key from the file
with open(pr_key_file, "r", encoding="utf8") as key_file:
    private_key = RSA.import_key(key_file.read())
# Decrypting the received message using the private key.
#opening the encrypted message I have received
with open('encrypted_message.bin', "rb") as f:
    rec_encrypted_msg = f.read()
#create a cipher object using the private key for decryption
private_key_cipher = PKCS1_OAEP.new(private_key)
#decrypt the message using the private key and print out the result
decrypted_message = private_key_cipher.decrypt(rec_encrypted_msg)
print(f"Decrypted message: {decrypted_message.decode('utf-8')}")
#write the decrypted message into a simple txt file
with open('decrypted_message.txt', "w", encoding = 'utf8') as f:
    f.write(decrypted_message.decode('utf-8'))
```

The decryption was successful, and we could read that the message sent was: PöttyösPenne. The decrypted message is also saved to a file.