Assymetric_encryption_in_python

April 10, 2023

1 Assymetric Encryption in Python

For this example we will be using the rsa library

[1]:
```
import rsa
```

Using rsa, we can create a public and private key

[5]:
```
publicKey, privateKey = rsa.newkeys(nbits=512)
```

The generated publicKey and privateKey are unique 512 bit keys. The valuable thing is, they are linked.

Now I can give the publicKey to whomever I want. They can then use that public key to encrypt a message before sending it to me.

[12]:
```
plaintext = b'This Message is Private'

ciphertext = rsa.encrypt(plaintext, publicKey)
```

[8]:
```
ciphertext
```

[8]:
```
b"V\xea>\xaa\x07\xbb\xc7\x9d\re\x1c\xc2=u\x03mC'\xa0\xb5\x90\xb5m\xaa\x90\xea~#O
\x8e\xe5P\x10a\x86n\x9e\xda\x89%G\xa7\x80\xb6o&=9|t<\x8c\x96P\xe8\x14\x9eG\xbc\x
92\x05O\xc9\xbd"
```

Because the public and private keys are linked, I can then use the privateKey to decrypt the same message that the sender encrypted

[19]:
```
decrypted_message = rsa.decrypt(ciphertext, privateKey)
decrypted_message
```

[19]:
```
b'This Message is Private'
```

Now we can test that this does not work with any key:

[16]:
```
new_publicKey, new_privateKey = rsa.newkeys(nbits=512)

new_ciphertext = rsa.encrypt(plaintext, new_publicKey)
```

Now we have encrypted the same plaintext message with the original publicKey. Next we try to decrypt it with the newly made privateKey

[17]:
```
new_decrypted_message = rsa.decrypt(new_ciphertext, privateKey)
```

```
---------------------------------------------------------------------------
DecryptionError                           Traceback (most recent call last)
Cell In[17], line 1
----> 1 new_decrypted_message = rsa.decrypt(new_ciphertext, privateKey)

File ~\AppData\Local\Programs\Python\Python310\lib\site-packages\rsa\pkcs1.py:
 ↪281, in decrypt(crypto, priv_key)
    279 anything_bad = cleartext_marker_bad | sep_idx_bad
    280 if anything_bad:
--> 281     raise DecryptionError("Decryption failed")
    283 return cleartext[sep_idx + 1 :]

DecryptionError: Decryption failed
```

As expected, the decryption failed.

This is because the public key used to encrypt and the private key we attempt to decrypt with, are not linked …

1.1 Saving your Keys for later use

[20]:
```
# Private key
with open(r"./private.pem", 'wb') as f:

    f.write(privateKey.save_pkcs1())

# Public key
with open(r"./public.pem", 'wb') as f:
    f.write(publicKey.save_pkcs1())
```

The saved files looks like this:

—–BEGIN RSA PRIVATE KEY—– MIIBPAIBAAJBAKPcEb5D3XNK9Bge+WRVSw6+zud6sylN8bblI3YhOJpc
A26Yfbdu4+13hQR96Kp9/moky7zGWvneQzMCAwEAAQJAGycgmZLNY4mMIWMgmIu5
wFvGFX2zqqiL4W7fDXg7r25SSOZbrjhckgT3aQWy04DKs7YjIlSJC+bMD4xZ2BpW
AQIjAOKW376w/b/k7b8J0s160k8443j2hSiOqGQe0YiBIGfc7kECHwC5IM74px7g 8zA-
sjXz4DKXQ9n8FCaM1yKVo7qfIPHMCIwDYu3NcVLMCP5jgqCbIgHh1ana6FyKa
/ok26vuE0+4JKpuBAh5RTweYTk5Rnn3aE4p/p45saNcA72qvhKChpbArUeECImuS
1HimjiF3MbJYsXQRFSeUPVUOv4rsgEEQrvkoHxY6jYM= —–END RSA PRIVATE KEY—–

1.2 Loading saved keys

We will now save the key to a .pem file so we are able to use it later.

Now, lets imagine we are both the sender of the message and the recipient - where the recipient is the one who made the key-pairs.

In reality, the sender will load the public key, and the recipient will load the private key.

[22]:
```
# Load the public key from a file
with open(r"./public.pem", 'rb') as f:
    publicKeyData = f.read()
    publicKey = rsa.PublicKey.load_pkcs1(publicKeyData)

# Load the private key from a file
with open(r"./private.pem", 'rb') as f:
    privateKeyData = f.read()
    privateKey = rsa.PrivateKey.load_pkcs1(privateKeyData)
```

[23]:
```
plaintext = b'Can we decrypt this?'
ciphertext = rsa.encrypt(plaintext, publicKey)
ciphertext
```

[23]:
```
b'd\r]\xb6\xc8\xee\x95Z]\x1a\x80\xb2\x83\xcd\xda\\-
\x19\xc4\xf3\x8ao\xdf\x8d\xad/\xf8\xcd\xb4\xf4.\xa0\xb4\x9f+\xaa7\xcb\xde~f\xe42
Yo\xc0\xae\xe7F\xa0m\xdd%\x99\xbd\xc8z\xbfP\x8a\xf60\xeb.'
```

Once again, we have successfully encrypted our plaintext message. Now we decrypt it with the privatekey:

[24]:
```
decrypted_message = rsa.decrypt(ciphertext, privateKey)
decrypted_message
```

[24]:
```
b'Can we decrypt this?'
```

1.2.1 Success!

Thanks for reading, I hope this was useful :)

[ ]:
```

```