FORMAN CHRISTIAN COLLEGE
(A CHARTERED UNIVERSITY)

BY LOVE SERVE ONE ANOTHER
Estd. 1864

Course Title:

COMP 421 – Information Security

Section: A

Instructor:

Dr Saad Bin Saleem

Instrument:

Assignment 2

Student name & roll number:

Sundas Javaid 19-10685

Semester: SP 22'

Submission Date:

May 28th, 2022

Introduction

The RSA algorithm has two keys: a public key and a private key for each pair. The message is encrypted using one key, which can only be decoded with the other key.

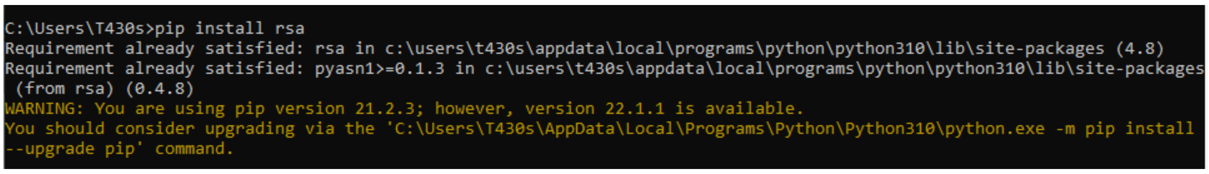
Let's imagine we have two peers interacting on a channel protected by the RSA algorithm. The plain text will be encrypted by the sender using the recipient's public key. This ensures that the message can only be decrypted by the receiver using their private key.

A public key repository will be used to store the public key. The private key, on the other hand, is kept secret at the recipient's end, as the name implies.

Explanation

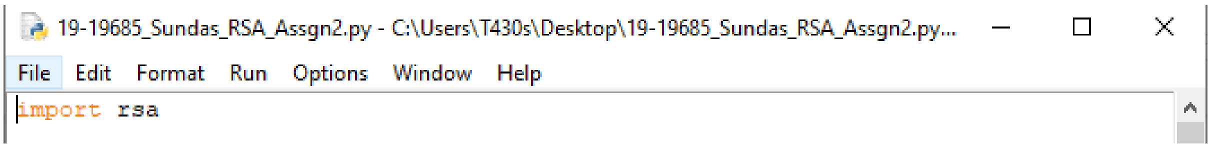
This section will focus on implementation of RSA algorithms using `python`. The code will read input from user and encrypt and decrypt it afterwards.

The first step is to open `cmd` and install `rsa`. The package is significant in terms of cryptography.



```
C:\Users\T430s>pip install rsa
Requirement already satisfied: rsa in c:\users\t430s\appdata\local\programs\python\python310\lib\site-packages (4.8)
Requirement already satisfied: pyasn1>=0.1.3 in c:\users\t430s\appdata\local\programs\python\python310\lib\site-packages
 (from rsa) (0.4.8)
WARNING: You are using pip version 21.2.3; however, version 22.1.1 is available.
You should consider upgrading via the 'C:\Users\T430s\AppData\Local\Programs\Python\Python310\python.exe -m pip install
--upgrade pip' command.
```

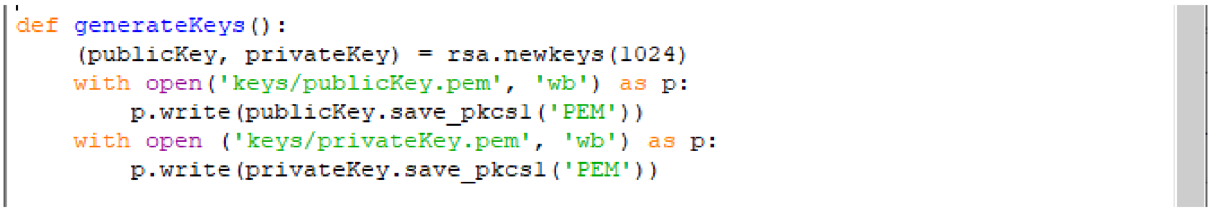
The package is already installed hence the message "`requirement already satisfies`". After that, on python IDLE, the following library must be imported.



19-19685_Sundas_RSA_Assgn2.py - C:\Users\T430s\Desktop\19-19685_Sundas_RSA_Assgn2.py...
File Edit Format Run Options Window Help

```
import rsa
```

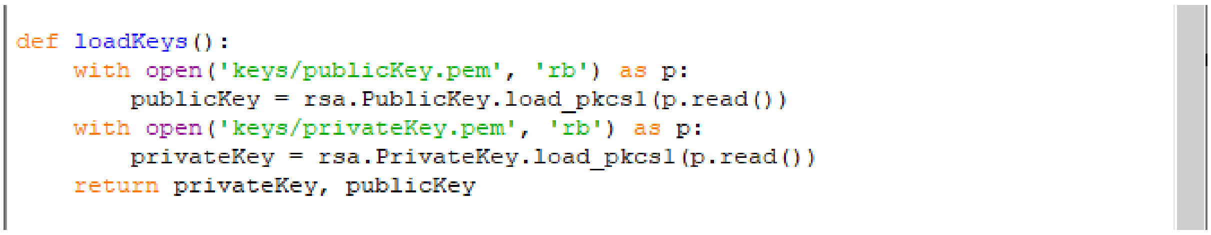
To produce the private and public keys, we'll first construct two helper functions. The keys will be a tuple of public and private keys, with the keys being written into files after that. A folder named `Keys` must be created which will include two files, one for private and one for public keys. A function called "`generateKeys()`" will be made that will write public and private keys in text files in the folder previously mentioned `Keys`.



```
def generateKeys():
    (publicKey, privateKey) = rsa.newkeys(1024)
    with open('keys/publicKey.pem', 'wb') as p:
        p.write(publicKey.save_pkcs1('PEM'))
    with open ('keys/privateKey.pem', 'wb') as p:
        p.write(privateKey.save_pkcs1('PEM'))
```

Here, short key length is used to keep the sample input short, but in a real-world scenario it is recommended to use `3072-bit` or `4096-bit` keys as `1024-bit` key limits the maximum message length. Next is to make a public key and private key.

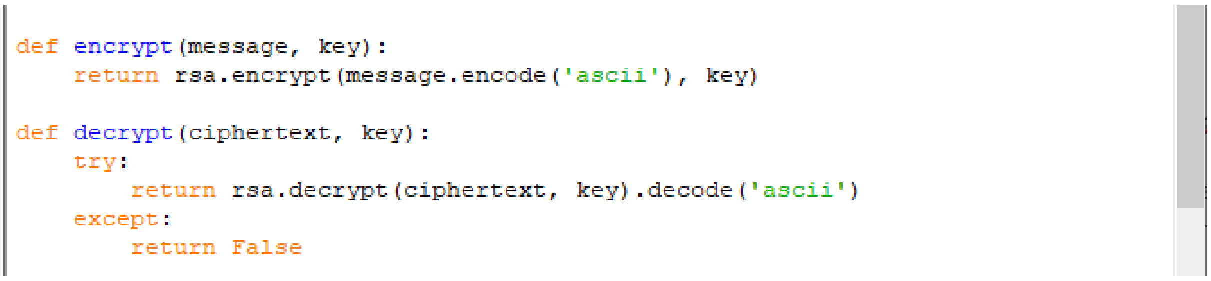
A function will be created by the name `loadKeys()`. This will fetch the public and private keys.



```python
def loadKeys():
    with open('keys/publicKey.pem', 'rb') as p:
        publicKey = rsa.PublicKey.load_pkcs1(p.read())
    with open('keys/privateKey.pem', 'rb') as p:
        privateKey = rsa.PrivateKey.load_pkcs1(p.read())
    return privateKey, publicKey
```

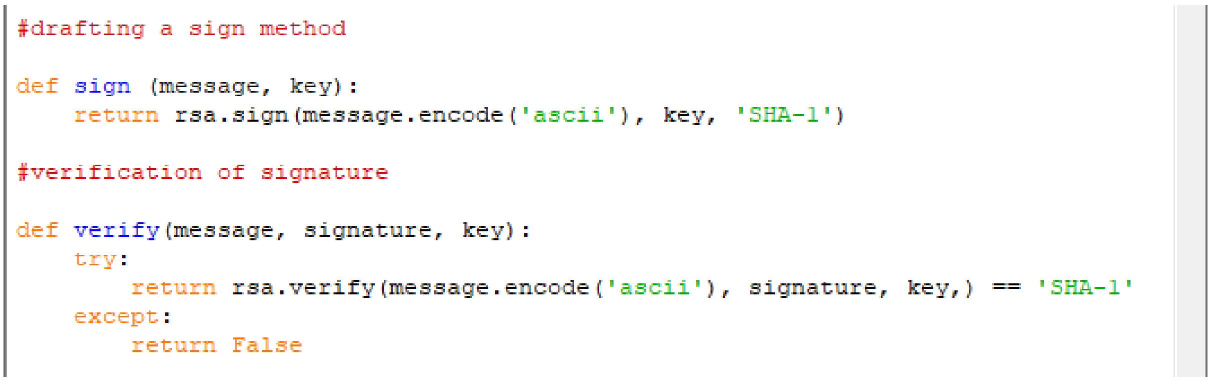
Next is to draft an encryption technique. The message and the encryption key will be sent to the encrypt function. The encrypted message must be returned after defining the `encrypt` function. The message will be encoded in ASCII and a key will be assigned to it.

For the decryption method, we'll decrypt the ciphertext using the key. The message will be decoded and then returned as decrypted message. Because we're using ASCII encoding, we'll also use ASCII decoding. If the key was unable to decode the message, the function will return `False`.



```python
def encrypt(message, key):
    return rsa.encrypt(message.encode('ascii'), key)

def decrypt(ciphertext, key):
    try:
        return rsa.decrypt(ciphertext, key).decode('ascii')
    except:
        return False
```

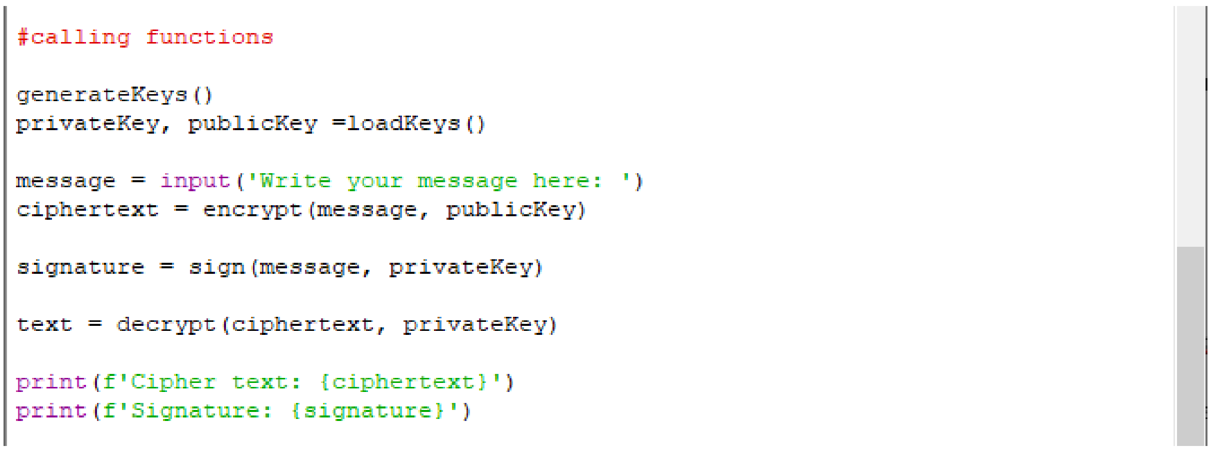
Two more functions will be created, namely `sign` and `verify`. They will be using a hash function "`SHA1`". The key and our hashing method will be transferred to the message that we will encode. The verification will occur by the verify function. It will take message, key and signature as input.



```python
#drafting a sign method

def sign (message, key):
    return rsa.sign(message.encode('ascii'), key, 'SHA-1')

#verification of signature

def verify(message, signature, key):
    try:
        return rsa.verify(message.encode('ascii'), signature, key,) == 'SHA-1'
    except:
        return False
```

The verify function returns the hash algorithm used in the signature. We'll check if the signature equals the hash algorithm.

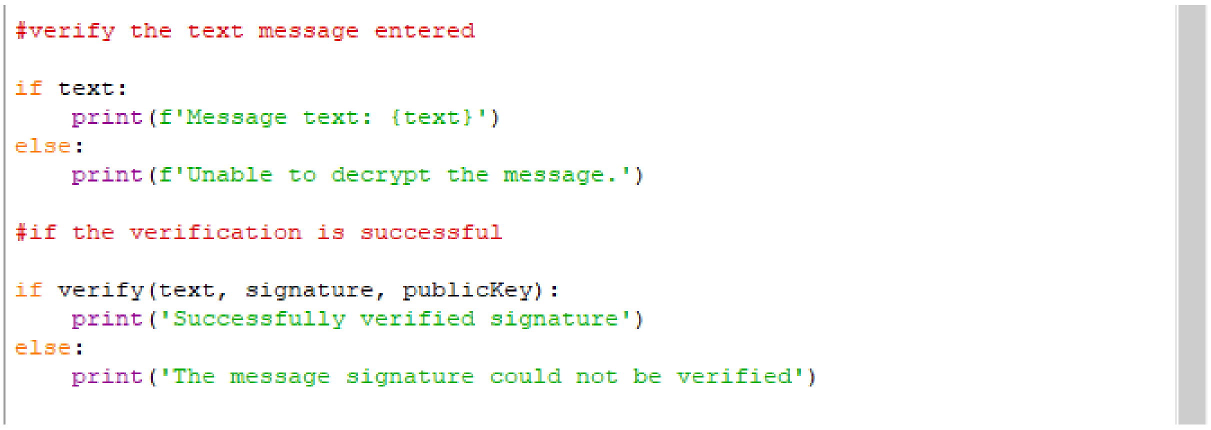
The RSA algorithm has been completed. The next task is to generate keys and take message input from the user. The same message is to be encrypted with the use of public key. Then the signatures will be generated so that the message can be signed with our private key. The message will be authenticated by verifying the message with the public key.



```
#calling functions

generateKeys()
privateKey, publicKey =loadKeys()

message = input('Write your message here: ')
ciphertext = encrypt(message, publicKey)

signature = sign(message, privateKey)

text = decrypt(ciphertext, privateKey)

print(f'Cipher text: {ciphertext}')
print(f'Signature: {signature}')
```

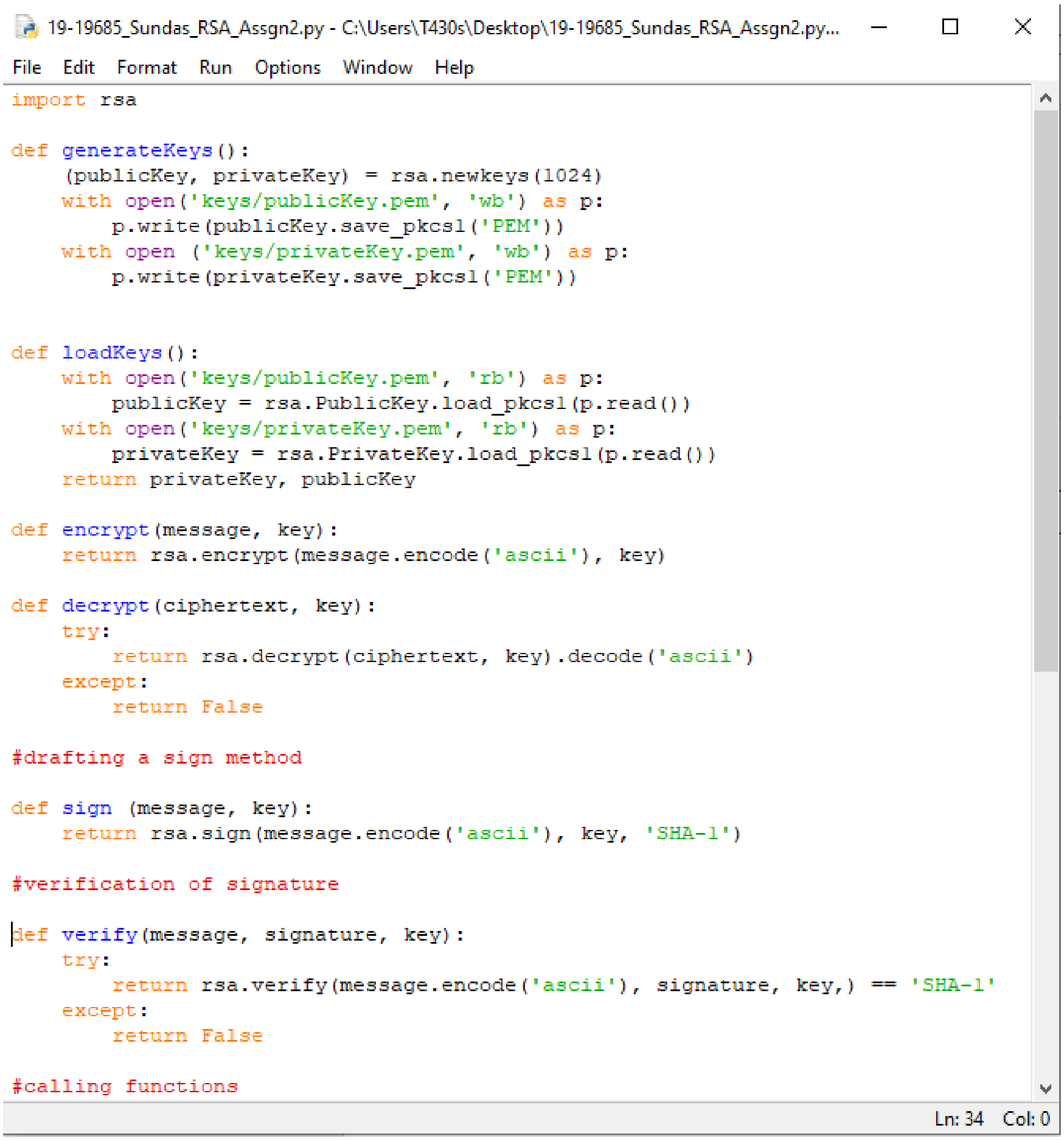
The text function will decrypt our encrypted message into plain text. The decryption method will be created, and it will take ciphertext and private key as input. The ciphertext and signature will be printed.

The plain text must be validated too. If the text is valid, we'll get the messaged that has been entered. Otherwise, the function will return "`Unable to decrypt the message`".

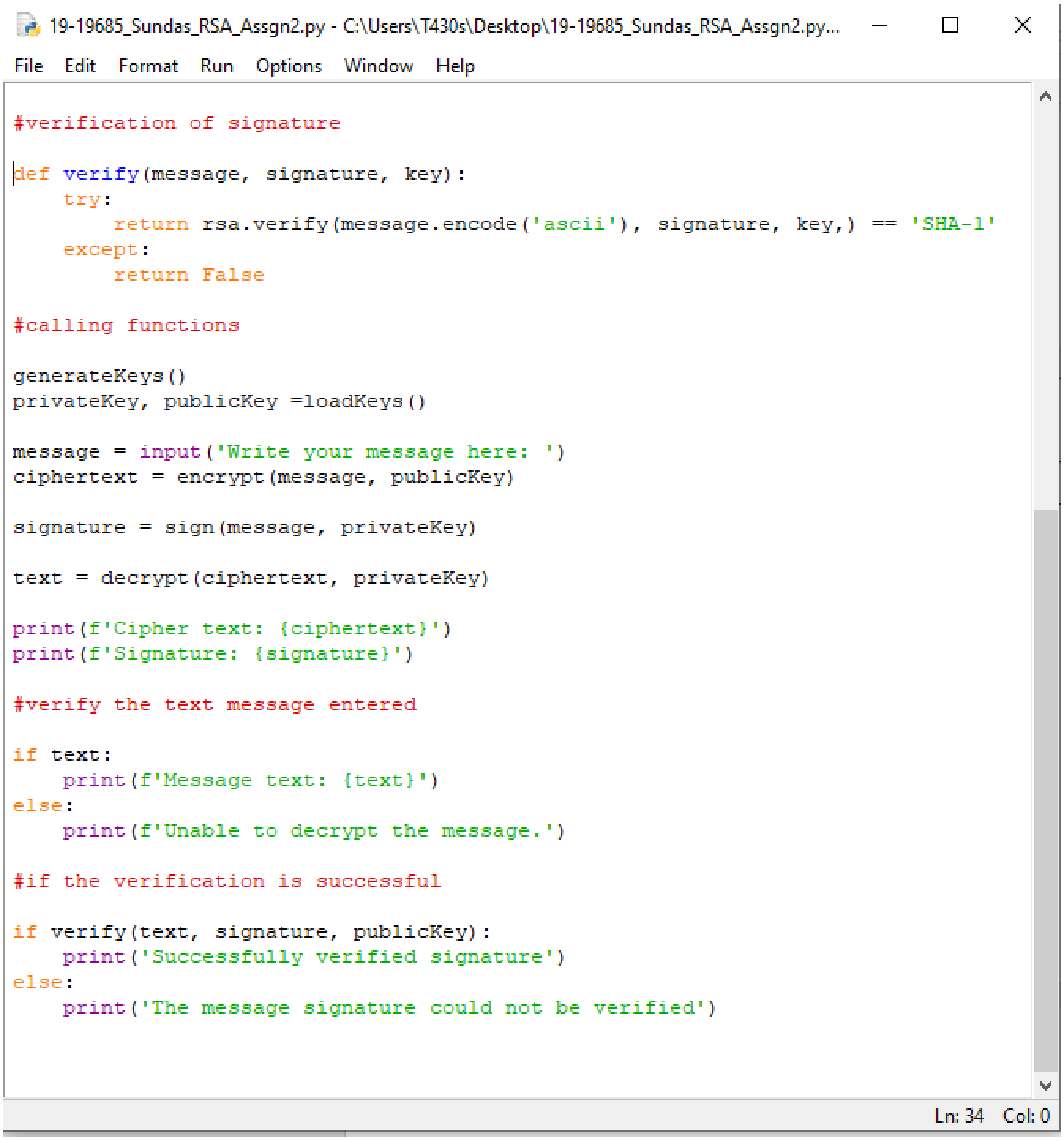


```
#verify the text message entered

if text:
    print(f'Message text: {text}')
else:
    print(f'Unable to decrypt the message.')

#if the verification is successful

if verify(text, signature, publicKey):
    print('Successfully verified signature')
else:
    print('The message signature could not be verified')
```

The full code will look like this:

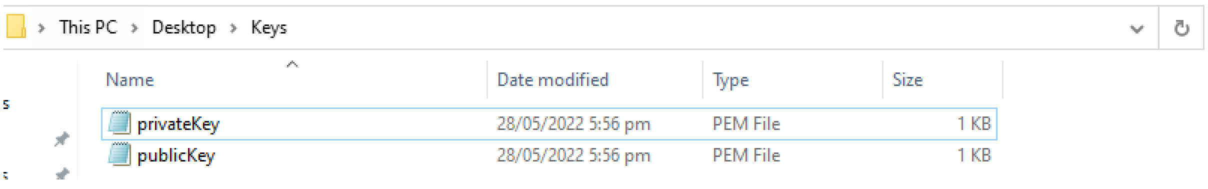

19-19685_Sundas_RSA_Assgn2.py - C:\Users\T430s\Desktop\19-19685_Sundas_RSA_Assgn2.py...
File Edit Format Run Options Window Help
Ln: 34 Col: 0

```python
import rsa

def generateKeys():
    (publicKey, privateKey) = rsa.newkeys(1024)
    with open('keys/publicKey.pem', 'wb') as p:
        p.write(publicKey.save_pkcs1('PEM'))
    with open ('keys/privateKey.pem', 'wb') as p:
        p.write(privateKey.save_pkcs1('PEM'))


def loadKeys():
    with open('keys/publicKey.pem', 'rb') as p:
        publicKey = rsa.PublicKey.load_pkcs1(p.read())
    with open('keys/privateKey.pem', 'rb') as p:
        privateKey = rsa.PrivateKey.load_pkcs1(p.read())
    return privateKey, publicKey

def encrypt(message, key):
    return rsa.encrypt(message.encode('ascii'), key)

def decrypt(ciphertext, key):
    try:
        return rsa.decrypt(ciphertext, key).decode('ascii')
    except:
        return False

#drafting a sign method

def sign (message, key):
    return rsa.sign(message.encode('ascii'), key, 'SHA-1')

#verification of signature

def verify(message, signature, key):
    try:
        return rsa.verify(message.encode('ascii'), signature, key,) == 'SHA-1'
    except:
        return False

#calling functions
```


19-19685_Sundas_RSA_Assgn2.py - C:\Users\T430s\Desktop\19-19685_Sundas_RSA_Assgn2.py...
File Edit Format Run Options Window Help
Ln: 34 Col: 0

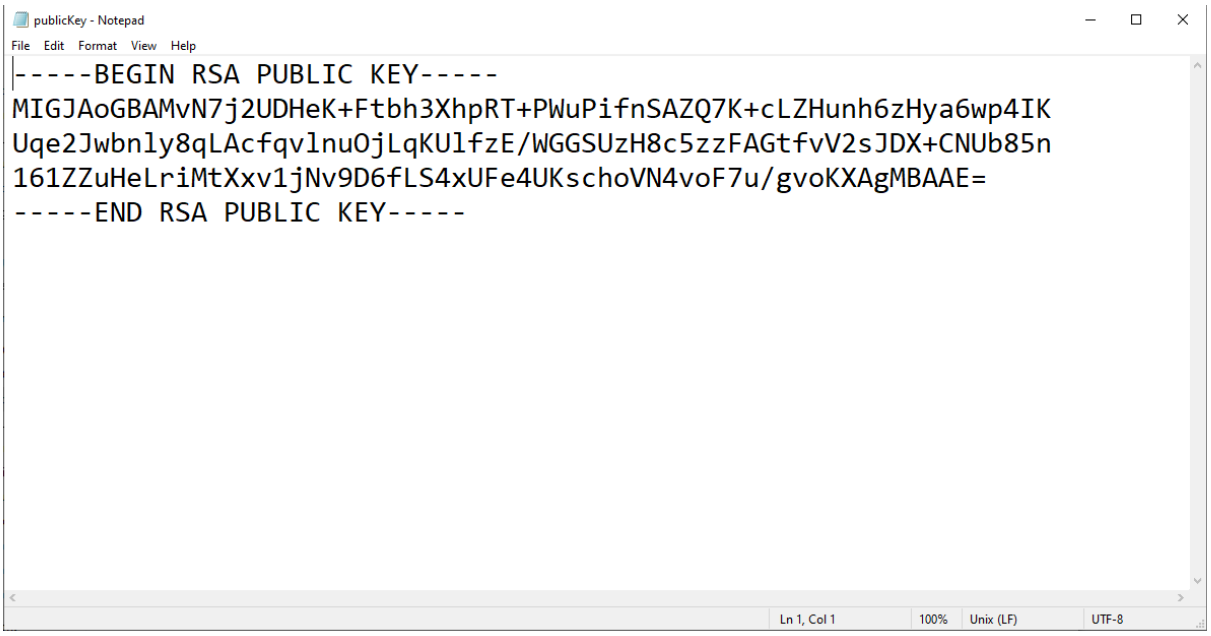
```python
#verification of signature

def verify(message, signature, key):
    try:
        return rsa.verify(message.encode('ascii'), signature, key,) == 'SHA-1'
    except:
        return False

#calling functions

generateKeys()
privateKey, publicKey =loadKeys()

message = input('Write your message here: ')
ciphertext = encrypt(message, publicKey)

signature = sign(message, privateKey)

text = decrypt(ciphertext, privateKey)

print(f'Cipher text: {ciphertext}')
print(f'Signature: {signature}')

#verify the text message entered

if text:
    print(f'Message text: {text}')
else:
    print(f'Unable to decrypt the message.')

#if the verification is successful

if verify(text, signature, publicKey):
    print('Successfully verified signature')
else:
    print('The message signature could not be verified')
```

When the code is executed, `python IDLE shell` will show the encrypted and decrypted data for public and private keys. The folder “`Keys`” will also contain public and private keys in text files.



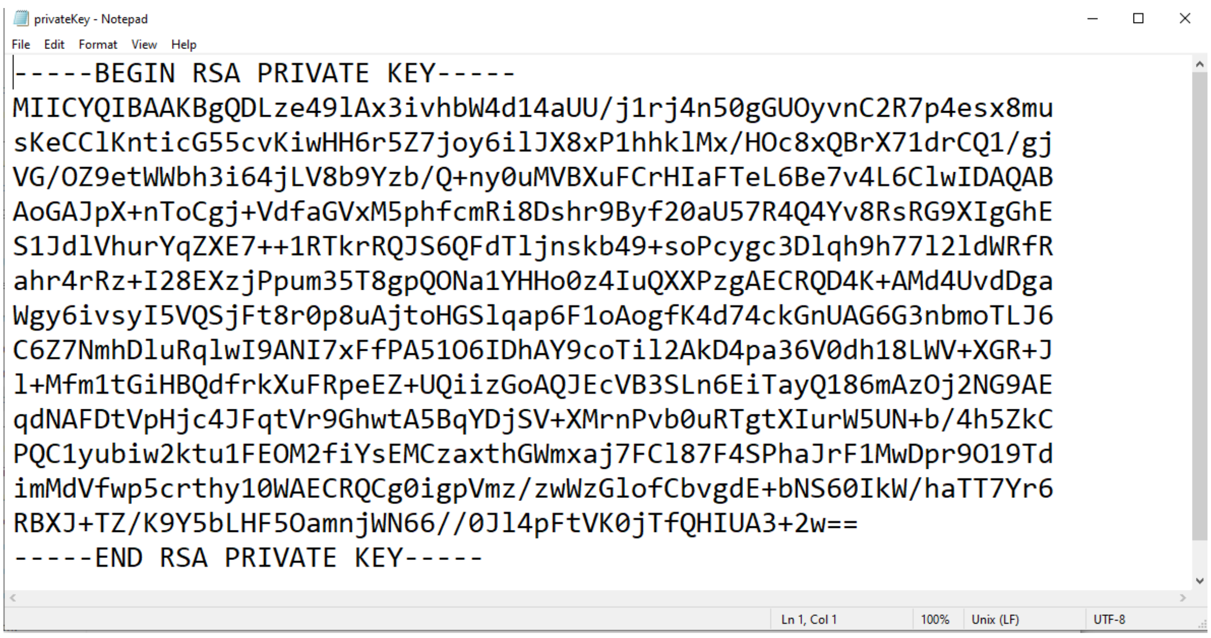
Name	Date modified	Type	Size
privateKey	28/05/2022 5:56 pm	PEM File	1 KB
publicKey	28/05/2022 5:56 pm	PEM File	1 KB

`publicKey`:



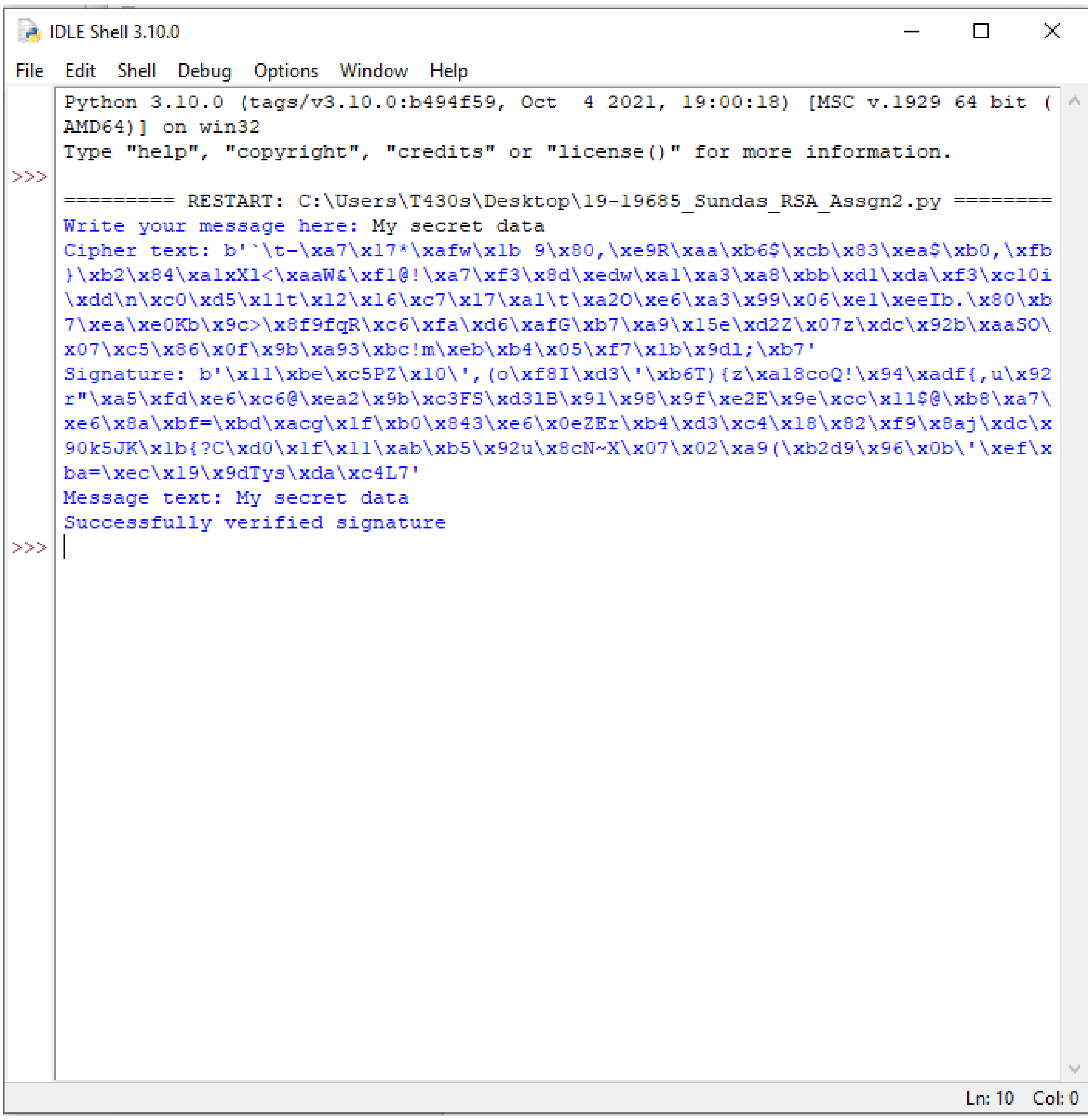
```
-----BEGIN RSA PUBLIC KEY-----
MIGJAoGBAMvN7j2UDHeK+Ftbh3XhpRT+PWuPifnSAZQ7K+cLZHunh6zHya6wp4IK
Uqe2Jwbnly8qLAcfqvlnuOjLqKUlfzE/WGGSUzH8c5zzFAGtfvV2sJDX+CNUb85n
161ZZuHeLriMtXxv1jNv9D6fLS4xUFe4UKschoVN4voF7u/gvoKXAgMBAAE=
-----END RSA PUBLIC KEY-----
```

privateKey:



```
-----BEGIN RSA PRIVATE KEY-----
MIICYQIBAAKBgQDLze49lAx3ivhbW4d14aUU/j1rj4n50gGUOyvnC2R7p4esx8mu
sKeCClKnticG55cvKiwHH6r5Z7joy6ilJX8xP1hhklMx/HOc8xQBrX71drCQ1/gj
VG/OZ9etWWbh3i64jLV8b9Yzb/Q+ny0uMVBXuFCrHIaFTeL6Be7v4L6ClwIDAQAB
AoGAJpX+nToCgj+VdfaGVxM5phfcmRi8Dshr9Byf20aU57R4Q4Yv8RsRG9XIgGhE
S1JdlVhurYqZXE7++1RTkrRQJS6QFdTljnskb49+soPcygc3Dlqh9h77l2ldWRfR
ahr4rRz+I28EXzjPpum35T8gpQONa1YHHo0z4IuQXXPzgAECRQD4K+AMd4UvdDga
Wgy6ivsyI5VQSjFt8r0p8uAjtoHGSlqap6F1oAogfK4d74ckGnUAG6G3nbmoTLJ6
C6Z7NmhDluRqlwI9ANI7xFfPA51O6IDhAY9coTil2AkD4pa36V0dh18LWV+XGR+J
l+Mfm1tGiHBQdfrkXuFRpeEZ+UQiizGoAQJEcVB3SLn6EiTayQ186mAzOj2NG9AE
qdNAFDtVpHjc4JFqtVr9GhwtA5BqYDjSV+XMrnPvb0uRTgtXIurW5UN+b/4h5ZkC
PQC1yubiw2ktu1FEOM2fiYsEMCzaxthGWmxaj7FCl87F4SPhaJrF1MwDpr9O19Td
imMdVfwp5crthy10WAECRQCg0igpVmz/zwWzGlofCbvgdE+bNS60IkW/haTT7Yr6
RBXJ+TZ/K9Y5bLHF5OamnjWN66//0Jl4pFtVK0jTfQHIUA3+2w==
-----END RSA PRIVATE KEY-----
```

IDLE shell:



```
Python 3.10.0 (tags/v3.10.0:b494f59, Oct  4 2021, 19:00:18) [MSC v.1929 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
========= RESTART: C:\Users\T430s\Desktop\19-19685_Sundas_RSA_Assgn2.py ========
Write your message here: My secret data
Cipher text: b'`\t-\xa7\x17*\xafw\x1b 9\x80,\xe9R\xaa\xb6$\xcb\x83\xea$\xb0,\xfb
}\xb2\x84\xa1xX1<\xaaW&\xf1@!\xa7\xf3\x8d\xedw\xa1\xa3\xa8\xbb\xd1\xda\xf3\xc10i
\xdd\n\xc0\xd5\x11t\x12\x16\xc7\x17\xa1\t\xa2O\xe6\xa3\x99\x06\xe1\xeeIb.\x80\xb
7\xea\xe0Kb\x9c>\x8f9fqR\xc6\xfa\xd6\xafG\xb7\xa9\x15e\xd2Z\x07z\xdc\x92b\xaaSO\
x07\xc5\x86\x0f\x9b\xa93\xbc!m\xeb\xb4\x05\xf7\x1b\x9d1;\xb7'
Signature: b'\x11\xbe\xc5PZ\x10\',(o\xf8I\xd3\'\xb6T){z\xa18coQ!\x94\xadf{,u\x92
r"\xa5\xfd\xe6\xc6@\xea2\x9b\xc3FS\xd31B\x91\x98\x9f\xe2E\x9e\xcc\x11$@\xb8\xa7\
xe6\x8a\xbf=\xbd\xacg\x1f\xb0\x843\xe6\x0eZEr\xb4\xd3\xc4\x18\x82\xf9\x8aj\xdc\x
90k5JK\x1b{?C\xd0\x1f\x11\xab\xb5\x92u\x8cN~X\x07\x02\xa9(\xb2d9\x96\x0b\'\xef\x
ba=\xec\x19\x9dTys\xda\xc4L7'
Message text: My secret data
Successfully verified signature
>>> |
```

Conclusion

The beforementioned procedure encrypted a message using a public key and our private key has signed it. The message of a sender is encrypted using the public key and the reciver uses the signature verified by the private key to access the message sent. RSA algorithm provides confidentiality, integrity, authenticity and security of data.

*****