The following project was done in Pycharm using the coding language Python. The project utilizes RSA encryption and a Caesar cipher to encrypt a message the user enters.

The code prints out the public and private keys once they are created. This was to ensure that it was working and was displayed. For the Caesar cipher, I had it pick a random number that it would be shifted by. -- This is also printed out to see what number it picked.

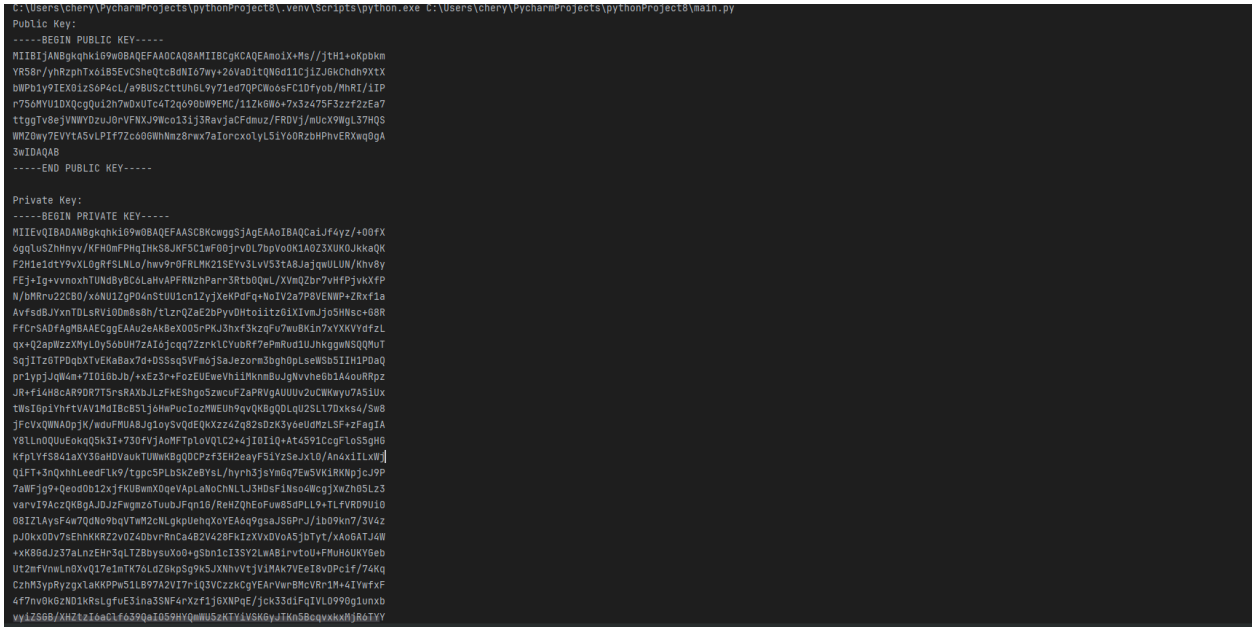
-After the RSA keys are created and the Caesar cipher number is picked it asks for the user's message.
-Once the user enters the message, it will be encrypted first by the Caesar cipher and then by the public key.
-To decrypt the message the private key is used then the Caesar shift is reversed to get the original message.

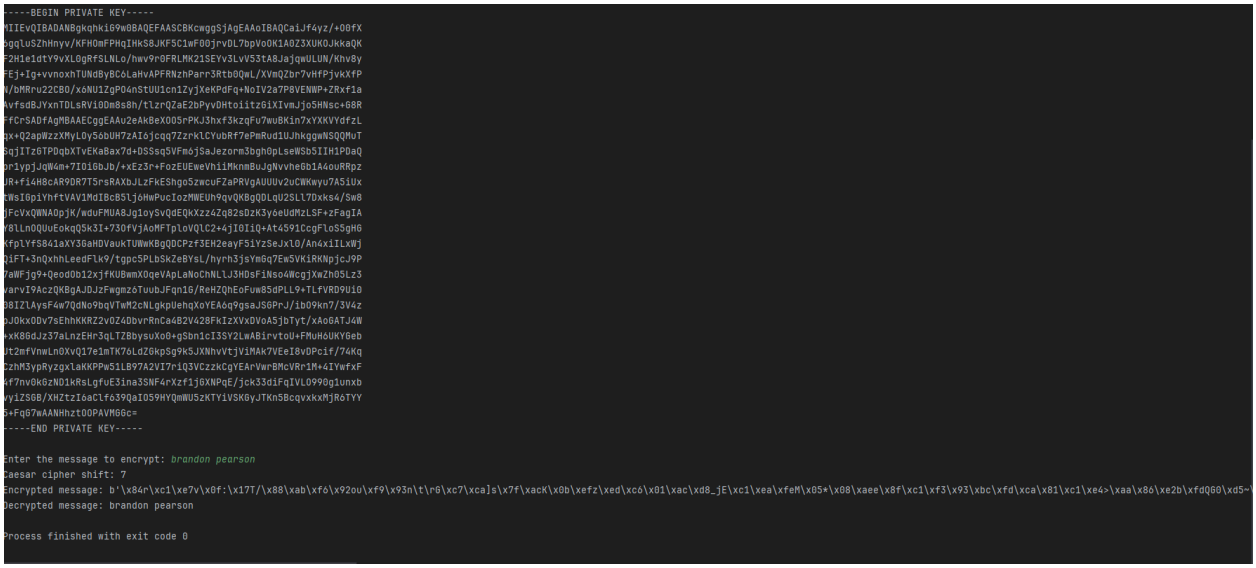
Ecyption = 1st Caesar cipher, 2nd public key
Decryption = 1st private key, 2nd reversed Caesar cipher

```
from cryptography.hazmat.primitives import serialization
from cryptography.hazmat.primitives.asymmetric import rsa
from cryptography.hazmat.primitives import hashes
from cryptography.hazmat.primitives import padding
from cryptography.hazmat.primitives.asymmetric import padding
import random


# RSA key pairs
private_key = rsa.generate_private_key(
    public_exponent=65537,
    key_size=2048, )
public_key = private_key.public_key()

# Serialize and print public key
public_key_pem = public_key.public_bytes(
    encoding=serialization.Encoding.PEM,
    format=serialization.PublicFormat.SubjectPublicKeyInfo
)
print(f"Public Key:\n{public_key_pem.decode()}")
```

```
# Serialize and print private key
private_key_pem = private_key.private_bytes(
   encoding=serialization.Encoding.PEM,
   format=serialization.PrivateFormat.PKCS8,
   encryption_algorithm=serialization.NoEncryption()
)
print(f"Private Key:\n{private_key_pem.decode()}")


# Caesar cipher
def caesar_cipher(text, shift):
   result = ""
   for char in text:
       if char.isalpha():
           ascii_offset = 65 if char.isupper() else 97
           shifted_char = chr((ord(char) - ascii_offset + shift) % 26 +
ascii_offset)
           result += shifted_char
       else:
           result += char
   return result


# Encrypt a message
def encrypt_message(message, public_key, caesar_shift):
   caesar_encrypted_message = caesar_cipher(message, caesar_shift)
   cipher_text = public_key.encrypt(
       caesar_encrypted_message.encode(),
       padding.OAEP(
           mgf=padding.MGF1(algorithm=hashes.SHA256()),
           algorithm=hashes.SHA256(),
           label=None
       )
   )
   return cipher_text, caesar_shift

# Decrypt the message
def decrypt_message(cipher_text, private_key, caesar_shift):
   plain_text = private_key.decrypt(
       cipher_text,
       padding.OAEP(
           mgf=padding.MGF1(algorithm=hashes.SHA256()),
           algorithm=hashes.SHA256(),
           label=None
       )
   )
   decrypted_message = caesar_cipher(plain_text.decode(), -caesar_shift)
   return decrypted_message
```

```
# User input / message
message = input("Enter the message to encrypt: ")

# Encrypt the message
cipher_text, caesar_shift = encrypt_message(message, public_key,
random.randint(1, 25))
print(f"Caesar cipher shift: {caesar_shift}")
print(f"Encrypted message: {cipher_text}")

# Decrypt the message
decrypted_message = decrypt_message(cipher_text, private_key, caesar_shift)
print(f"Decrypted message: {decrypted_message}")
```






**FROM IMAGE ABOVE**

**Public Key**:

-----BEGIN PUBLIC KEY-----
MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAmoiX+Ms//jtH1+oKpbkm
YR58r/yhRzphTx6iB5EvCSheQtcBdNI67wy+26VaDitQNGd11CjiZJGkChdh9XtX
bWPb1y9IEX0izS6P4cL/a9BUSzCttUhGL9y71ed7QPCWo6sFC1Dfyob/MhRI/iIP
r756MYU1DXQcgQui2h7wDxUTc4T2q690bW9EMC/11ZkGW6+7x3z475F3zzf2zEa7
ttggTv8ejVNWYDzuJ0rVFNXJ9Wco13ij3RavjaCFdmuz/FRDVj/mUcX9WgL37HQS
WMZ0wy7EVYtA5vLPlf7Zc60GWhNmz8rwx7aIorcxolyL5iY6ORzbHPhvERXwq0gA
3wIDAQAB
-----END PUBLIC KEY-----

**Private Key:**

-----BEGIN PRIVATE KEY-----
MIIEvQIBADANBgkqhkiG9w0BAQEFAASCBKcwggSjAgEAAoIBAQCaiJf4yz/+O0fX
6gqluSZhHnyv/KFHOmFPHqIHkS8JKF5C1wF00jrvDL7bpVoOK1A0Z3XUKOJkkaQK
F2H1e1dtY9vXL0gRfSLNLo/hwv9r0FRLMK21SEYv3LvV53tA8JajqwULUN/Khv8y
FEj+Ig+vvnoxhTUNdByBC6LaHvAPFRNzhParr3Rtb0QwL/XVmQZbr7vHfPjvkXfP
N/bMRru22CBO/x6NU1ZgPO4nStUU1cn1ZyjXeKPdFq+NoIV2a7P8VENWP+ZRxf1a
AvfsdBJYxnTDLsRVi0Dm8s8h/tlzrQZaE2bPyvDHtoiitzGiXIvmJjo5HNsc+G8R
FfCrSADfAgMBAAECggEAAu2eAkBeXOO5rPKJ3hxf3kzqFu7wuBKin7xYXKVYdfzL
qx+Q2apWzzXMyLOy56bUH7zAI6jcqq7ZzrklCYubRf7ePmRud1UJhkggwNSQQMuT
SqjITzGTPDqbXTvEKaBax7d+DSSsq5VFm6jSaJezorm3bgh0pLseWSb5IIH1PDaQ
pr1ypjJqW4m+7IOiGbJb/+xEz3r+FozEUEweVhiiMknmBuJgNvvheGb1A4ouRRpz
JR+fi4H8cAR9DR7T5rsRAXbJLzFkEShgo5zwcuFZaPRVgAUUUv2uCWKwyu7A5iUx
tWsIGpiYhftVAV1MdIBcB5lj6HwPucIozMWEUh9qvQKBgQDLqU2SLl7Dxks4/Sw8
jFcVxQWNAOpjK/wduFMUA8Jg1oySvQdEQkXzz4Zq82sDzK3y6eUdMzLSF+zFagIA
Y8lLnOQUuEokqQ5k3l+73OfVjAoMFTploVQlC2+4jI0IiQ+At4591CcgFloS5gHG
KfplYfS841aXY3GaHDVaukTUWwKBgQDCPzf3EH2eayF5iYzSeJxl0/An4xiILxWj
QiFT+3nQxhhLeedFlk9/tgpc5PLbSkZeBYsL/hyrh3jsYmGq7Ew5VKiRKNpjcJ9P
7aWFjg9+QeodOb12xjfKUBwmXOqeVApLaNoChNLlJ3HDsFiNso4WcgjXwZh05Lz3

varvI9AczQKBgAJDJzFwgmz6TuubJFqn1G/ReHZQhEoFuw85dPLL9+TLfVRD9Ui0
08IZlAysF4w7QdNo9bqVTwM2cNLgkpUehqXoYEA6q9gsaJSGPrJ/ibO9kn7/3V4z
pJOkxODv7sEhhKKRZ2vOZ4DbvrRnCa4B2V428FkIzXVxDVoA5jbTyt/xAoGATJ4W
+xK8GdJz37aLnzEHr3qLTZBbysuXo0+gSbn1cI3SY2LwABirvtoU+FMuH6UKYGeb
Ut2mfVnwLn0XvQ17e1mTK76LdZGkpSg9k5JXNhvVtjViMAk7VEeI8vDPcif/74Kq
CzhM3ypRyzgxlaKKPPw51LB97A2VI7riQ3VCzzkCgYEArVwrBMcVRr1M+4IYwfxF
4f7nv0kGzND1kRsLgfuE3ina3SNF4rXzf1jGXNPqE/jck33diFqIVLO990g1unxb
vyiZSGB/XHZtzI6aCIf639QaIO59HYQmWU5zKTYiVSKGyJTKn5BcqvxkxMjR6TYY
5+FqG7wAANHhztOOPAVMGGc=
-----END PRIVATE KEY-----

**Enter the message to encrypt: brandon pearson**
**Caesar cipher shift: 7**
**Encrypted message:**
b'\x84r\xc1\xe7v\x0f:\x17T/\x88\xab\xf6\x92ou\xf9\x93n\t\rG\xc7\xca]s\x7f\xacK\x0b\xefz\xed\xc6\x01\xac\
xd8_jE\xc1\xea\xfeM\x05*\x08\xaee\x8f\xc1\xf3\x93\xbc\xfd\xca\x81\xc1\xe4>\xaa\x86\xe2b\xfdQG0\xd5~
\xda\x01\x8b\x90\x91\xf4\xe6\xfb\x14\xf9\xc0\xaf\xa3%\xa4b\x11\xa0iYf\x8e\xe5\xe8\x0f\x02\xff2\x03\xde
O+\xa9\xec\x04\xfa\xfe\xcc\x1cKt\xb8B\xc8Z\x0fg[W\x86\x12\xe7R\x83K=CF\xb6)\xc7\xf9\xe0\x1e\xeeX9
}\xf4\x85\xa2\x9du\xa5\x90\x87\xfe\xad\xab\x03{l\x87]]l_3a\x1a\x0c\x02\rd\x8d\x8d==J\x16\n\xeb\xfe/\xe3
\n\x9c(\x9f\xcds\xd7\x8f\xa9+.\xc2l\x1b\xf8\xfb@\xa5\x18ax^T\xf5\xb8\x07\xfa\xce\xaa\xbe\xc0\x1a\x9b\x
84V\xef\x99\x0e\x8ft\xf8\xcb\xd9s\xff\xb7\xe2\xb7\xce\xbfL\xd0V\x13\xd9\x821e\x01\xfd\x892\x83\x1e\xb
3\xc4\xe5\xe7\x91\xc9z\xb9\xdb\tF\x08\r\xc4\x19\x19\x0b'
**Decrypted message: brandon pearson**