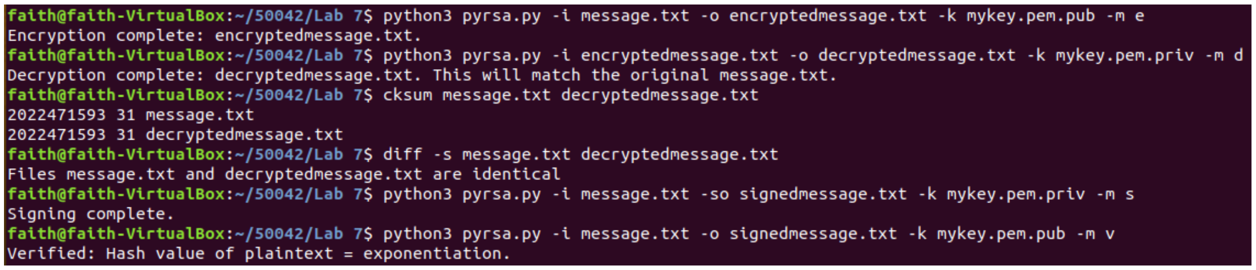
Part I:

RSA Encryption, Decryption & Message Signing, Verification:

Using the `pyrsa.py` file in Part I with `message.txt` as the input file, `mykey.pem.pub` as the public key and `mykey.pem.priv` as the private key, the following shows the output of the encryption of `message.txt`, decryption of `encryptedmessage.txt`, signing of `message.txt` as well as the verification of the signature.



```
faith@faith-VirtualBox:~/50042/Lab 7$ python3 pyrsa.py -i message.txt -o encryptedmessage.txt -k mykey.pem.pub -m e
Encryption complete: encryptedmessage.txt.
faith@faith-VirtualBox:~/50042/Lab 7$ python3 pyrsa.py -i encryptedmessage.txt -o decryptedmessage.txt -k mykey.pem.priv -m d
Decryption complete: decryptedmessage.txt. This will match the original message.txt.
faith@faith-VirtualBox:~/50042/Lab 7$ cksum message.txt decryptedmessage.txt
2022471593 31 message.txt
2022471593 31 decryptedmessage.txt
faith@faith-VirtualBox:~/50042/Lab 7$ diff -s message.txt decryptedmessage.txt
Files message.txt and decryptedmessage.txt are identical
faith@faith-VirtualBox:~/50042/Lab 7$ python3 pyrsa.py -i message.txt -so signedmessage.txt -k mykey.pem.priv -m s
Signing complete.
faith@faith-VirtualBox:~/50042/Lab 7$ python3 pyrsa.py -i message.txt -o signedmessage.txt -k mykey.pem.pub -m v
Verified: Hash value of plaintext = exponentiation.
```

Encryption of RSA (`message.txt`):

The encryption is done with the following command:

```
python3 pyrsa.py -i message.txt -o encryptedmessage.txt -k mykey.pem.pub -m e
```

Decryption of RSA (`encryptedmessage.txt`):

The decryption is done with the following command:

```
python3 pyrsa.py -i encryptedmessage.txt -o decryptedmessage.txt -k
mykey.pem.priv -m d
```

To verify that the decrypted file is the same as the original file, the following two commands are run:

```
cksum message.txt decryptedmessage.txt
diff -s message.txt decryptedmessage.txt
```

Signing of message (`message.txt`):

The signing of the message is done with the following command:

```
python3 pyrsa.py -i message.txt -so signedmessage.txt -k mykey.pem.priv -m s
```

Verifying signature (`signedmessage.txt`):

The verification of the message is done with the following command:

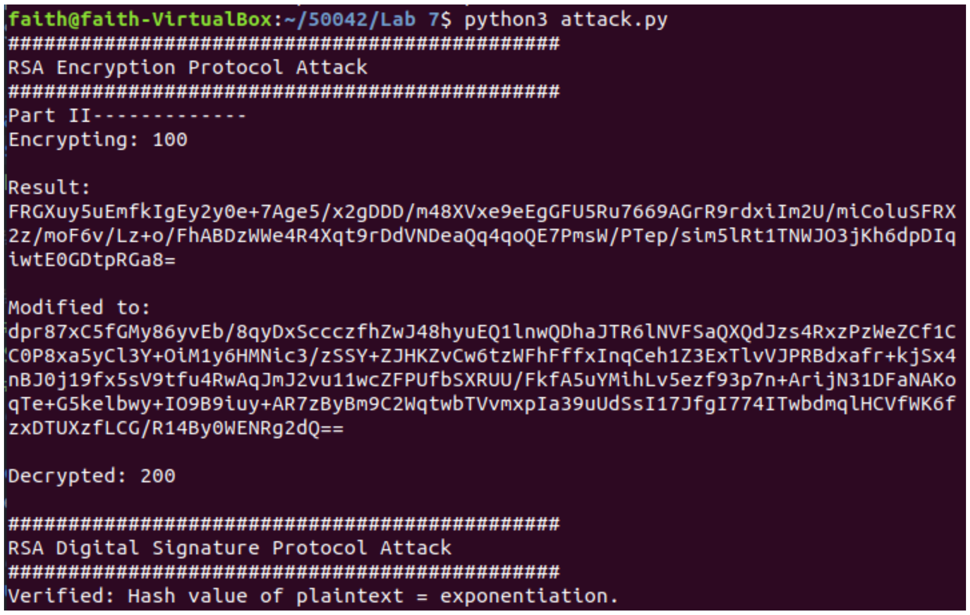
```
python3 pyrsa.py -i message.txt -o signedmessage.txt -k mykey.pem.pub -m v
```

A Boolean value is returned to determine if the signature is verified. If the signature is verified, the message, "`Verified: Hash value of plaintext = exponentiation.`" is printed onto the console. If the signature is not verified, the message, "`Not verified.`" is printed onto the console.

Part II:

<u>Demo protocol attack:</u>

Run `attack.py` in the zip folder.



```
faith@faith-VirtualBox:~/50042/Lab 7$ python3 attack.py
##############################################
RSA Encryption Protocol Attack
##############################################
Part II------------
Encrypting: 100

Result:
FRGXuy5uEmfkIgEy2y0e+7Age5/x2gDDD/m48XVxe9eEgGFU5Ru7669AGrR9rdxiIm2U/miColuSFRX
2z/moF6v/Lz+o/FhABDzWWe4R4Xqt9rDdVNDeaQq4qoQE7PmsW/PTep/sim5lRt1TNWJO3jKh6dpDIq
iwtE0GDtpRGa8=

Modified to:
dpr87xC5fGMy86yvEb/8qyDxScccz fhZwJ48hyuEQ1lnwQDhaJTR6lNVFSaQXQdJzs4RxzPzWeZCf1C
C0P8xa5yCl3Y+OiM1y6HMNic3/zSSY+ZJHKZvCw6tzWFhFffxInqCeh1Z3ExTlvVJPRBdxafr+kjSx4
nBJ0j19fx5sV9tfu4RwAqJmJ2vu11wcZFPUfbSXRUU/FkfA5uYMihLv5ezf93p7n+ArijN31DFaNAKo
qTe+G5kelbwy+IO9B9iuy+AR7zByBm9C2WqtwbTVvmxpIa39uUdSsI17JfgI774ITwbdmqlHCVfWK6f
zxDTUXzfLCG/R14By0WENRg2dQ==

Decrypted: 200

##############################################
RSA Digital Signature Protocol Attack
##############################################
Verified: Hash value of plaintext = exponentiation.
```

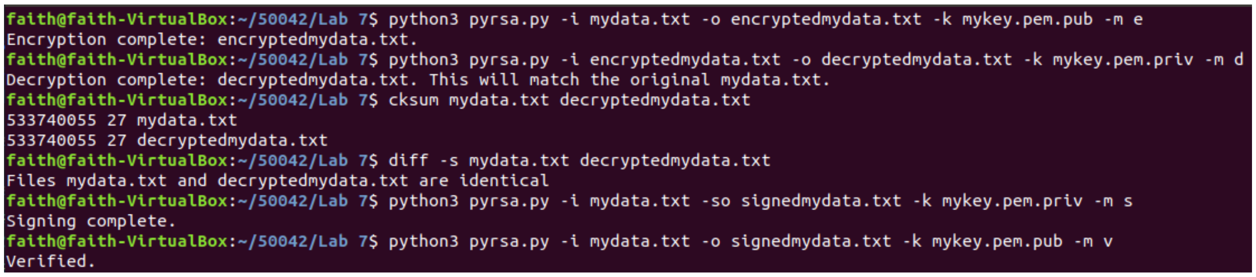
The limitation of RSA Encryption Protocol Attack is that the modified cipher text can be verified with the signature and we can easily know that the cipher text was altered.

The limitation of the RSA Digital Signature Protocol Attack is that the message should be verified with the signature which is exponentiation applied to the hash of the message and not just the exponentiation of the message itself, thus you would observe that the public key and not the private key is being applied.

Part III:

RSA Encryption, Decryption & Message Signing, Verification with OAEP & PSS with the given public & private keys:

Using the `pyrsa.py` file in Part III with `mydata.txt` as the input file, `mykey.pem.pub` as the public key and `mykey.pem.priv` as the private key, the following shows the output of the encryption of `mydata.txt`, decryption of `encryptedmydata.txt`, signing of `mydata.txt` as well as the verification of the signature.



```
faith@faith-VirtualBox:~/50042/Lab 7$ python3 pyrsa.py -i mydata.txt -o encryptedmydata.txt -k mykey.pem.pub -m e
Encryption complete: encryptedmydata.txt.
faith@faith-VirtualBox:~/50042/Lab 7$ python3 pyrsa.py -i encryptedmydata.txt -o decryptedmydata.txt -k mykey.pem.priv -m d
Decryption complete: decryptedmydata.txt. This will match the original mydata.txt.
faith@faith-VirtualBox:~/50042/Lab 7$ cksum mydata.txt decryptedmydata.txt
533740055 27 mydata.txt
533740055 27 decryptedmydata.txt
faith@faith-VirtualBox:~/50042/Lab 7$ diff -s mydata.txt decryptedmydata.txt
Files mydata.txt and decryptedmydata.txt are identical
faith@faith-VirtualBox:~/50042/Lab 7$ python3 pyrsa.py -i mydata.txt -so signedmydata.txt -k mykey.pem.priv -m s
Signing complete.
faith@faith-VirtualBox:~/50042/Lab 7$ python3 pyrsa.py -i mydata.txt -o signedmydata.txt -k mykey.pem.pub -m v
Verified.
```

Encryption of RSA (`mydata.txt`):

The encryption is done with the following command:

```
python3 pyrsa.py -i mydata.txt -o encryptedmydata.txt -k mykey.pem.pub -m e
```

Decryption of RSA (`encryptedmydata.txt`):

The decryption is done with the following command:

```
python3 pyrsa.py -i encryptedmydata.txt -o decryptedmydata.txt -k mykey.pem.priv -m d
```

To verify that the decrypted file is the same as the original file, the following two commands are run:

```
cksum mydata.txt decryptedmydata.txt
diff -s mydata.txt decryptedmydata.txt
```

Signing of message (`mydata.txt`):

The signing of the message is done with the following command:

```
python3 pyrsa.py -i mydata.txt -so signedmydata.txt -k mykey.pem.priv -m s
```

Verifying signature (`signedmydata.txt`):

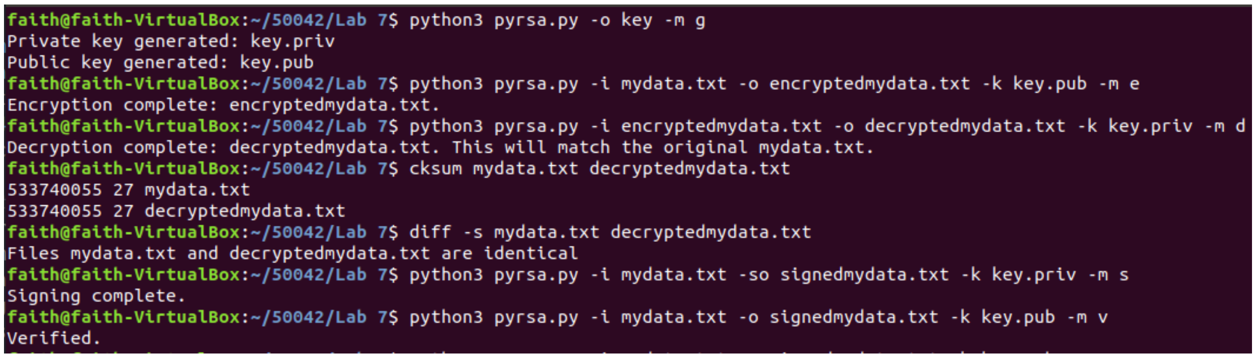
The verification of the message is done with the following command:

```
python3 pyrsa.py -i mydata.txt -o signedmydata.txt -k mykey.pem.pub -m v
```

A Boolean value is returned to determine if the signature is verified. If the signature is verified, the message, "`Verified.`" is printed onto the console. If the signature is not verified, the message, "`Not verified.`" is printed onto the console.

RSA Encryption, Decryption & Message Signing, Verification with OAEP & PSS with generated public & private keys:

Using the `pyrsa.py` file in Part III with `mydata.txt` as the input file, `key.pub` as the generated public key and `key.priv` as the generated private key, the following shows the output of the encryption of `mydata.txt`, decryption of `encryptedmydata.txt`, signing of `mydata.txt` as well as the verification of the signature.



```
faith@faith-VirtualBox:~/50042/Lab 7$ python3 pyrsa.py -o key -m g
Private key generated: key.priv
Public key generated: key.pub
faith@faith-VirtualBox:~/50042/Lab 7$ python3 pyrsa.py -i mydata.txt -o encryptedmydata.txt -k key.pub -m e
Encryption complete: encryptedmydata.txt.
faith@faith-VirtualBox:~/50042/Lab 7$ python3 pyrsa.py -i encryptedmydata.txt -o decryptedmydata.txt -k key.priv -m d
Decryption complete: decryptedmydata.txt. This will match the original mydata.txt.
faith@faith-VirtualBox:~/50042/Lab 7$ cksum mydata.txt decryptedmydata.txt
533740055 27 mydata.txt
533740055 27 decryptedmydata.txt
faith@faith-VirtualBox:~/50042/Lab 7$ diff -s mydata.txt decryptedmydata.txt
Files mydata.txt and decryptedmydata.txt are identical
faith@faith-VirtualBox:~/50042/Lab 7$ python3 pyrsa.py -i mydata.txt -so signedmydata.txt -k key.priv -m s
Signing complete.
faith@faith-VirtualBox:~/50042/Lab 7$ python3 pyrsa.py -i mydata.txt -o signedmydata.txt -k key.pub -m v
Verified.
```

Generation of keys (`key.pub` & `key.priv`):

The key generation is done with the following command:

```
python3 pyrsa.py -o key -m g
```

Encryption of RSA (`mydata.txt`):

The encryption is done with the following command:

```
python3 pyrsa.py -i mydata.txt -o encryptedmydata.txt -k key.pub -m e
```

Decryption of RSA (`encryptedmydata.txt`):

The decryption is done with the following command:

```
python3 pyrsa.py -i encryptedmydata.txt -o decryptedmydata.txt -k key.priv -m
d
```

To verify that the decrypted file is the same as the original file, the following two commands are run:

```
cksum mydata.txt decryptedmydata.txt
diff -s mydata.txt decryptedmydata.txt
```

Signing of message (`mydata.txt`):

The signing of the message is done with the following command:

```
python3 pyrsa.py -i mydata.txt -so signedmydata.txt -k key.priv -m s
```

Verifying signature (`signedmydata.txt`):

The verification of the message is done with the following command:

```
python3 pyrsa.py -i mydata.txt -o signedmydata.txt -k key.pub -m v
```

A Boolean value is returned to determine if the signature is verified. If the signature is verified, the message, "`Verified.`" is printed onto the console. If the signature is not verified, the message, "`Not verified.`" is printed onto the console.

Explain the purpose of Optimal Asymmetric Encryption Padding (OAEP) to encrypt and decrypt using RSA. Explain how it works.

The OAEP algorithm is a form of Feistel network which uses a pair of random oracles G and H to process the plaintext prior to asymmetric encryption. OAEP can be used to build an all-or-nothing transform. OAEP adds an element of randomness which can be used to convert a deterministic encryption scheme (e.g., traditional RSA) into a probabilistic scheme.

Legend:

- n is the number of bits in the RSA modulus.
- k_0 and k_1 are integers fixed by the protocol.
- m is the plaintext message, an $(n - k_0 - k_1)$-bit string
- G and H are random oracles such as cryptographic hash functions.
- $\oplus$ is an XOR operation.

For encryption:

- messages are padded with k_1 zeros to be $n - k_0$ bits in length.
- r is a randomly generated k_0-bit string
- G expands the k_0 bits of r to $n - k_0$ bits.
- $X = m00..0 \oplus G(r)$
- H reduces the $n - k_0$ bits of X to k_0 bits.
- $Y = r \oplus H(X)$
- The output is X || Y where X is shown in the diagram as the leftmost block and Y as the rightmost block.

For decryption:

- recover the random string as $r = Y \oplus H(X)$
- recover the message as $m00..0 = X \oplus G(r)$

Explain the purpose of Probabilistic Signature Scheme (PSS) to sign and verify using RSA. Explain how it works.

PSS is a cryptographic signature scheme based on the RSA cryptography system and provides enhanced security.

Like other digital signature schemes, the following processing steps take place:

- The signature scheme hashes the message to be signed by using a hash function.
- The resulting hash is transformed into an encoded message.
- A signature primitive is applied to the encoded message by using the private key to produce the signature.

Unlike other digital signature schemes, the transform operation uses padding that is much more random. During the signature verification process, the following processing steps take place:

- The scheme hashes the message to be signed by using the same hash function that was used to sign the message.
- A verification primitive is then applied to the signature by using the public key of the key pair to recover the message.
- The scheme verifies that the encoded message is a valid transformation of the hash value.

PSS takes the input message and a salt (a random number) and runs them through a hash function. This hash H is used as the beginning part of the output. Then, a mask of H is calculated, which has the length of the RSA modulus minus the length of H. This mask is then XOR-ed with the salt (and some zero padding) and the output will be called maskedDB. Then, maskedDB is appended to H to generate the input for the RSA function.