



System Security Week 3

RSA and Digital Signing

Learning Outcomes

In this lab, you will:

- Learn Generating a private key
- Also learn the generation of public key from private key
- Use public key for encryption
- Use private key for decryption
- Digitally sign a document
- Also learn how a digitally signed document can be communicated over the internet
- Use private key for digitally signing a document to protect its integrity
- Use public key to verify the signature and check the document integrity

Tasks

1. Alice wants to communicate a secret file to Bob in the presence of Eve (the attacker).
2. Alice wants to communicate a digitally signed secret file to Bob in the presence of Eve (the attacker). The aim is to protect the file from any modifications.

Introduction

This Lab consists of two parts:

In Part-1 we will cover RSA encryption and decryption, while in Part-2 we will experiment signing a document.

Part-1:

As we know the asymmetric cryptography uses pairs of keys (i.e., private keys (known only to the owner) and public keys (made publicly available for everyone)). The sender (Alice) will encrypt the secret file with Bobs public key and Bob can then easily decrypt the file or message by using his private key as can be seen in Figure 1. In this example Eve (The attacker) can sniff or capture the encrypted data, but s/he cannot decrypt it. Why?

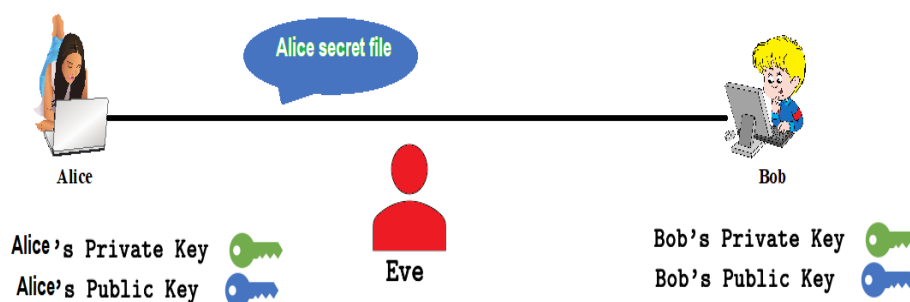


Figure 1 Public-Private key encryption

Part-2:

A digital signature is a mathematical technique used to validate the authenticity and integrity of a digital message/file. A digital signature is the equivalent of a handwritten signature. Digital

signatures can actually be far more secure. The purpose of a digital signature is to prevent the tampering and impersonation in digital communications. In many countries, including the United States, digital signatures have the same legal significance as traditional forms of signed documents. The United States Government now publishes electronic versions of budgets, laws, and congressional bills with digital signatures.

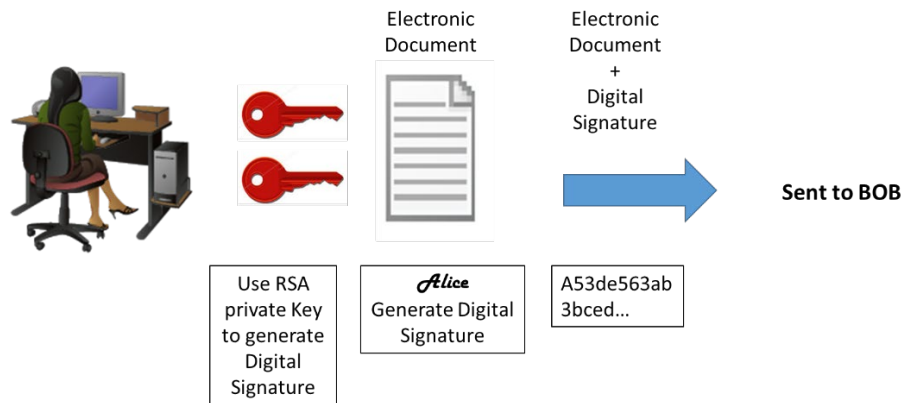


Figure 2 Digital Signatures

Part-1: Secure communication using asymmetric encryption

To accomplish secure communication between Alice and Bob, we need Two keys Bob Private key (Which will be only known to Bob) and Bob public key (which he published online or shared with Alice). This scenario is depicted in Figure 3.

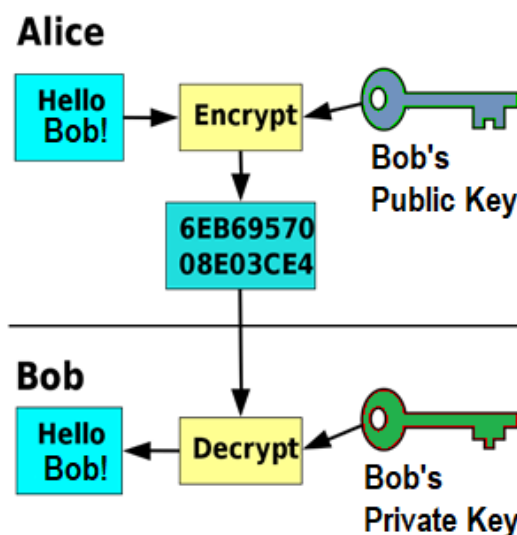


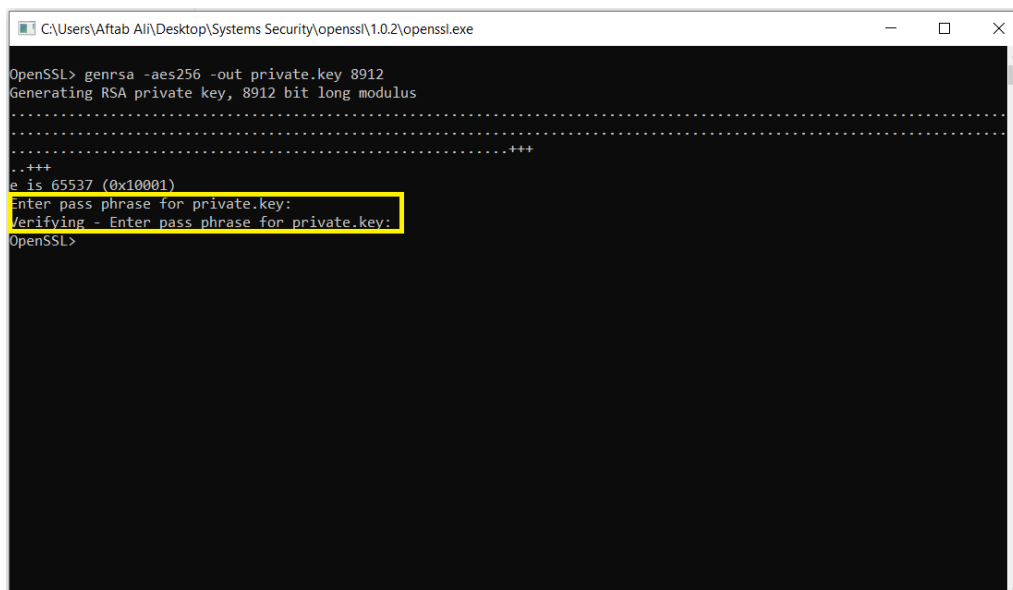
Figure 3 Selecting the executable

Step-1.1: Generation of Private Key for Bob

First, we will generate the Private key by using the following command:

genrsa -aes256 -out private.pem 8912

upon entering this command on the OpenSSL command prompt, the prompt will ask for password and then verifying password. In my case I use the password “AftabAli”. You can use any password, but you will have to remember that, because you will need that during the encryption and decryption process.



```
C:\Users\Aftab Ali\Desktop\System Security\openssl\1.0.2\openssl.exe
OpenSSL> genrsa -aes256 -out private.key 8912
Generating RSA private key, 8912 bit long modulus
.....+
.....+
e is 65537 (0x10001)
Enter pass phrase for private.key:
Verifying - Enter pass phrase for private.key:
OpenSSL>
```

Figure 4 Private key generation

Once, we enter the password the private key generation completes. So, the above command generates a private key (i.e., check your current working directory, there will be a file by the name of private, and its type will be pem as can be seen in Figure 5). Privacy-Enhanced Mail (pem) is a file format for storing and sending cryptographic keys.

| Name | Date modified | Type | Size |
|---------------------------|------------------|-----------------------|----------|
| HashInfo | 22/11/2018 15:48 | Text Document | 3 KB |
| libeay32.dll | 22/11/2018 15:48 | Application extens... | 1,342 KB |
| OpenSSL License | 31/08/2016 12:13 | Text Document | 7 KB |
| openssl | 22/11/2018 15:48 | Application | 507 KB |
| openssl-1.0.2q-i386-win32 | 09/12/2020 15:20 | WinRAR ZIP archive | 1,016 KB |
| Plaintext | 27/12/2020 15:27 | Text Document | 1 KB |
| private | 27/12/2020 20:34 | PEM File | 7 KB |
| ReadMe | 22/11/2018 15:48 | Text Document | 3 KB |
| ssleay32.dll | 22/11/2018 15:48 | Application extens... | 330 KB |

Figure 5 Private key and its type

Step-1.2: Generation of Public Key from the Private key for Bob

Now is the time for the generation of Public key from the private key we have just created. The following command will be used for the generation of public key from the already generated private key.

rsa -in private.pem -pubout -out public.pem

when we press enter, we will be asked for the password (it's the same password we used during the creation of the private key) as can be seen in Figure 6.

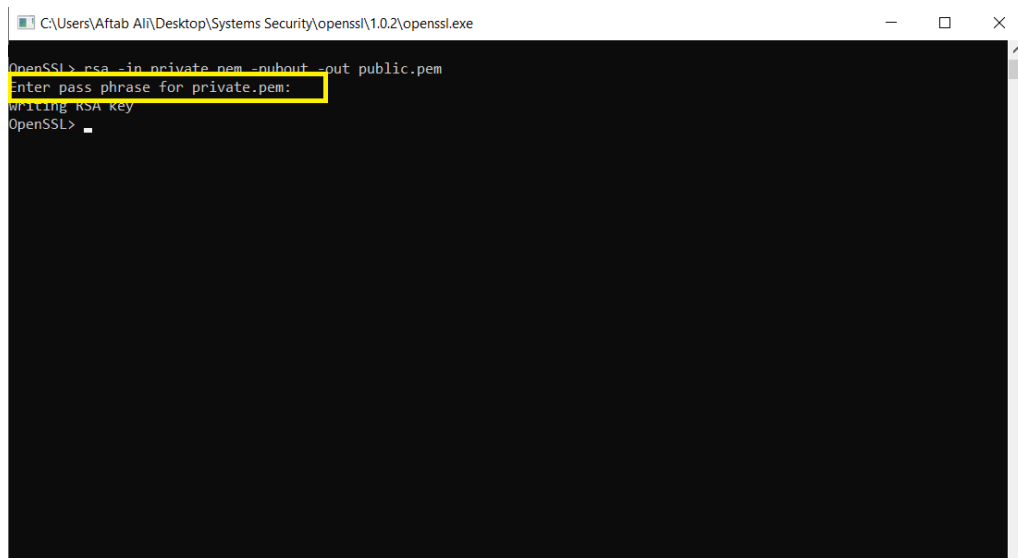


Figure 6 Public key generation

Once we provide the accurate password, the public key will be generated, and its type will pem as can be seen in Figure 7.

| Name | Date modified | Type | Size |
|---------------------------|------------------|-----------------------|----------|
| HashInfo | 22/11/2018 15:48 | Text Document | 3 KB |
| libeay32.dll | 22/11/2018 15:48 | Application extens... | 1,342 KB |
| OpenSSL License | 31/08/2016 12:13 | Text Document | 7 KB |
| openssl | 22/11/2018 15:48 | Application | 507 KB |
| openssl-1.0.2q-i386-win32 | 09/12/2020 15:20 | WinRAR ZIP archive | 1,016 KB |
| Plaintext | 27/12/2020 15:27 | Text Document | 1 KB |
| private | 27/12/2020 20:34 | PEM File | 7 KB |
| public | 27/12/2020 20:44 | PEM File | 2 KB |
| ReadMe | 22/11/2018 15:48 | Text Document | 3 KB |
| ssleay32.dll | 22/11/2018 15:48 | Application extens... | 330 KB |

Figure 7 Public key and its type

Now, Bob have both private and public keys. Bob will send his public key to Alice and will ask her to encrypt the secret file with it.

Step-1.3: Encryption

Alice will encrypt her secret file by using the following command as can be seen in Figure 8.

rsautl -encrypt -pubin -inkey public.pem -in Secretfile.txt -out encrypted.txt

```
C:\Users\Aftab Ali\Desktop\System Security\openssl\1.0.2\openssl.exe
OpenSSL> rsautl -encrypt -pubin -inkey public.pem -in Secretfile.txt -out encrypted.txt
OpenSSL>
```

Figure 8 encryption using public key

When Alice enters this command, the *Secretfile.txt* will be converted to *encrypted.txt* as can be seen in Figure 9 (Again this file will be in the current working directory/folder). The *Secretfile.txt* file should be in the working directory as can be seen in Figure 9.

| Name | Date modified | Type | Size |
|---------------------------|------------------|-----------------------|----------|
| encrypted | 27/12/2020 21:08 | Text Document | 2 KB |
| HashInfo | 22/11/2018 15:48 | Text Document | 3 KB |
| libeay32.dll | 22/11/2018 15:48 | Application extens... | 1,342 KB |
| OpenSSL License | 31/08/2016 12:13 | Text Document | 7 KB |
| openssl | 22/11/2018 15:48 | Application | 507 KB |
| openssl-1.0.2q-i386-win32 | 09/12/2020 15:20 | WinRAR ZIP archive | 1,016 KB |
| private | 27/12/2020 20:34 | PEM File | 7 KB |
| public | 27/12/2020 20:44 | PEM File | 2 KB |
| ReadMe | 22/11/2018 15:48 | Text Document | 3 KB |
| Secretfile | 27/12/2020 15:27 | Text Document | 1 KB |
| ssleay32.dll | 22/11/2018 15:48 | Application extens... | 330 KB |

Figure 9 Secret and encrypted files

Now, the *Secretfile.txt* is secure in the form of *encrypted.txt*. Alice will send this *encrypted.txt* to Bob. If during this communication Eve captured this file, S/he might not be able to read the contents. S/he will require Bobs private key to view the contents of the file or S/he will have to break the encryption algorithm.

Step-1.4: Decryption

Once Bob receives this *encrypted.txt* he will decrypt it by using the following command:

rsautl -decrypt -inkey private.pem -in encrypted.txt -out Secretfile2.txt

```
C:\Users\Aftab Ali\Desktop\System Security\openssl\1.0.2\openssl.exe
OpenSSL> rsautl -decrypt -inkey private.pem -in encrypted.txt -out Secretfile2.txt
Enter pass phrase for private.pem:
OpenSSL> _
```

Figure 10 Decryption using private key

Upon running the above command RSA will decrypt the *encrypted.txt* file and will generate another file *Secretfile2.txt* (again this file will be in the current working directory as can be seen in Figure 11). In this way Bob can view the contents of the secret file sent by Alice.

| Name | Date modified | Type | Size |
|---------------------------|------------------|-----------------------|----------|
| encrypted | 27/12/2020 21:08 | Text Document | 2 KB |
| HashInfo | 22/11/2018 15:48 | Text Document | 3 KB |
| libeay32.dll | 22/11/2018 15:48 | Application extens... | 1,342 KB |
| OpenSSL License | 31/08/2016 12:13 | Text Document | 7 KB |
| openssl | 22/11/2018 15:48 | Application | 507 KB |
| openssl-1.0.2q-i386-win32 | 09/12/2020 15:20 | WinRAR ZIP archive | 1,016 KB |
| private | 27/12/2020 20:34 | PEM File | 7 KB |
| public | 27/12/2020 20:44 | PEM File | 2 KB |
| ReadMe | 22/11/2018 15:48 | Text Document | 3 KB |
| Secretfile | 27/12/2020 15:27 | Text Document | 1 KB |
| Secretfile2 | 27/12/2020 21:32 | Text Document | 1 KB |
| ssleay32.dll | 22/11/2018 15:48 | Application extens... | 330 KB |

Figure 11 Plaintext file after Decryption

Tasks

1. Create an RSA 1024-bit long private key
2. Generate the public key from the private key
3. Next create a file *Secrets.txt*, and write some text into it
4. Encrypt the file with the public key and then decrypt it using your private key. Did you get the same text?

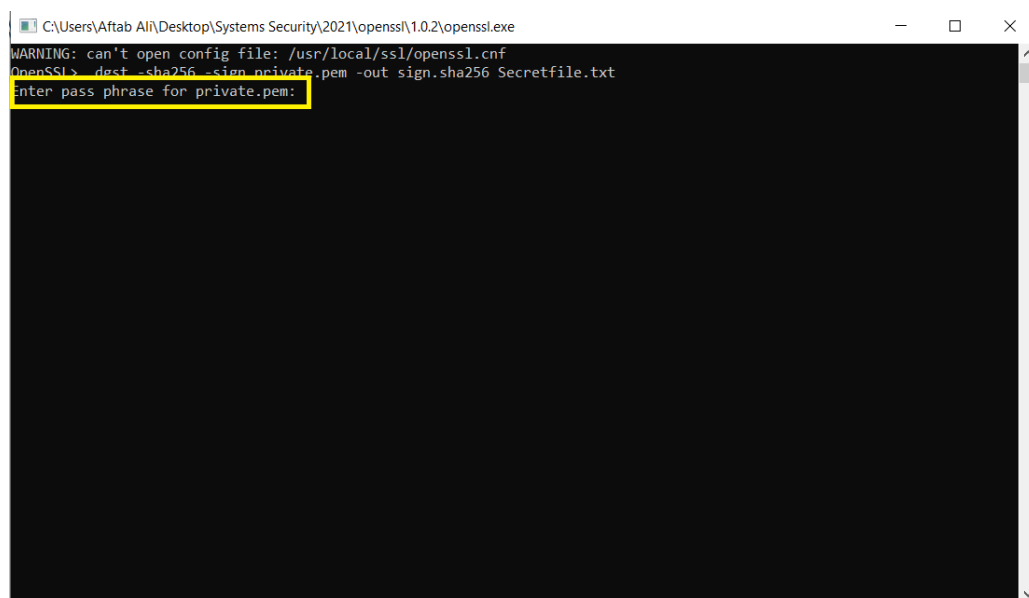
Part-2: Digital Signatures

In this part, you will use OpenSSL to verify a document signature between Alice and Bob as can be seen in Figure 2. Alice and Bob share a pair of private and public RSA keys. Each of them uses their private key to sign a legal document. They then send the documents to each other. Both Alice and Bob can verify each other's signature with the public key. To sign a file using OpenSSL Alice need to use a private key.

Step-2.1 Signing a document

Alice will use her private key (We can generate the private key for Alice as we did for Bob in **step-1.1**) to digitally sign the document by using the following command as can be seen in Figure 12.

dgst -sha256 -sign private.pem -out sign.sha256 Secretfile.txt



```
C:\Users\Aftab Ali\Desktop\System Security\2021\openssl\1.0.2\openssl.exe
WARNING: can't open config file: /usr/local/ssl/openssl.cnf
openssl> dgst -sha256 -sign private.pem -out sign.sha256 Secretfile.txt
Enter pass phrase for private.pem:
```

Figure 12 Signing the document

When Alice enters this command, she will be asked for the password for the private key and the *Secretfile.txt* will be digitally signed as can be seen in Figure 13. Moreover, a binary signature file named *sign.sha256* is also generated as can be seen in Figure 13.

Note: The input file *Secretfile.txt* should be present in the working directory as can be seen in Figure 13.

| Name | Date modified | Type | Size |
|---------------------------|------------------|-----------------------|----------|
| encrypted | 27/12/2020 21:08 | Text Document | 2 KB |
| HashInfo | 22/11/2018 15:48 | Text Document | 3 KB |
| libeay32.dll | 22/11/2018 15:48 | Application extens... | 1,342 KB |
| OpenSSL License | 31/08/2016 12:13 | Text Document | 7 KB |
| openssl | 22/11/2018 15:48 | Application | 507 KB |
| openssl-1.0.2q-i386-win32 | 09/12/2020 15:20 | WinRAR ZIP archive | 1,016 KB |
| private | 09/01/2021 12:17 | PEM File | 4 KB |
| public | 09/01/2021 12:17 | PEM File | 1 KB |
| ReadMe | 22/11/2018 15:48 | Text Document | 3 KB |
| Secretfile | 27/12/2020 15:27 | Text Document | 1 KB |
| sign.sha256 | 09/01/2021 15:23 | SHA256 File | 1 KB |
| signature | 09/01/2021 15:39 | File | 1 KB |
| ssleay32.dll | 22/11/2018 15:48 | Application extens... | 330 KB |

Figure 13 The signature files

The default output format of the OpenSSL signature is binary. If Alice want to share the signature over internet with Bob, she cannot use a binary format. She can use for instance Base64 format for file exchange. The following command will convert the binary file to Base64 format. The generated Base64 file by the name signature can be seen in Figure 13 (in the red box).

base64 -in sign.sha256 -out signature

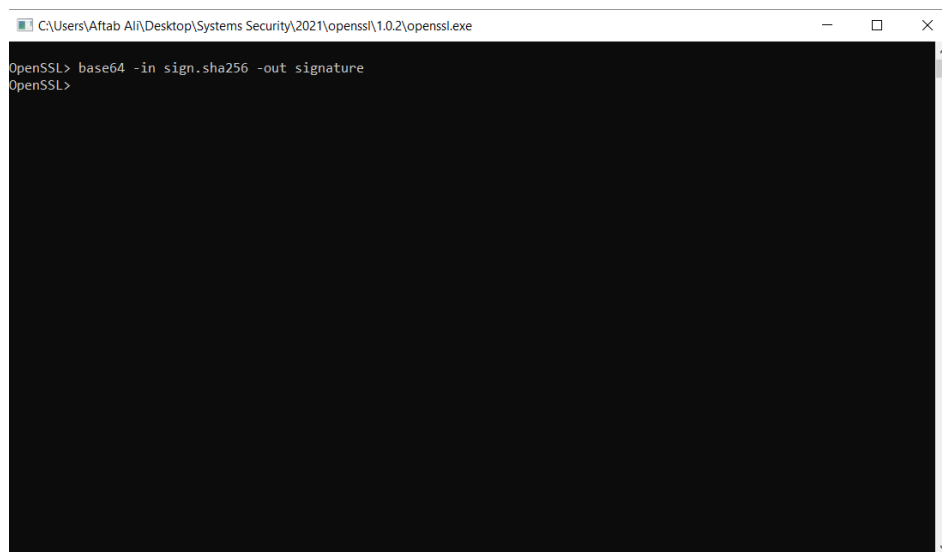
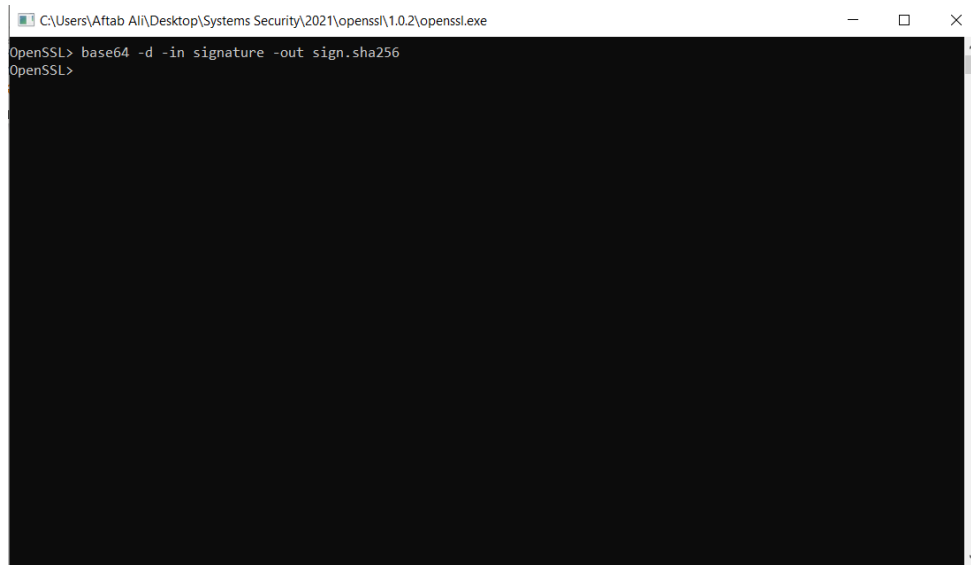


Figure 14 Base64 conversion

Step-2.2: Verifying the signature

To verify the signature, Bob need to convert the signature to binary file and then apply the verification process of OpenSSL. Bob can achieve this by using the following command:

base64 -d -in signature -out sign.sha256

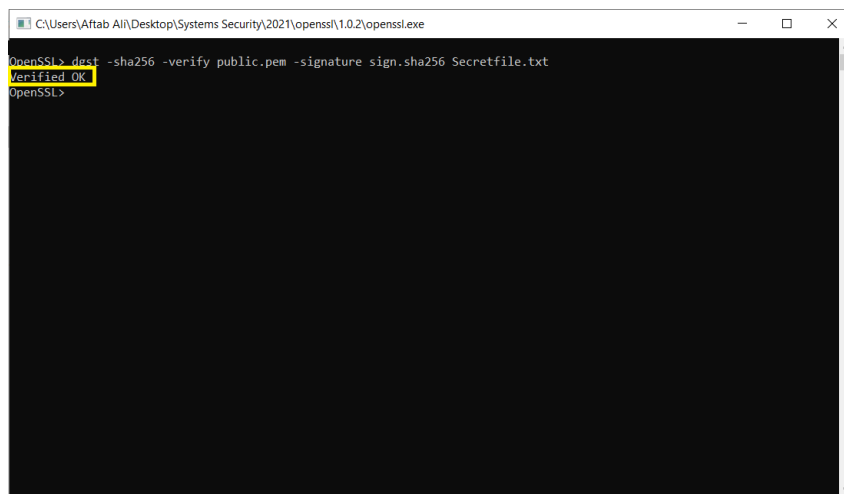


```
C:\Users\Aftab Ali\Desktop\System Security\2021\openssl\1.0.2\openssl.exe
OpenSSL> base64 -d -in signature -out sign.sha256
OpenSSL>
```

Figure 15 Conversion to binary format

Now to verify, Bob will use Alice public key:

dgst -sha256 -verify public.pem -signature sign.sha256 Secretfile.txt



```
C:\Users\Aftab Ali\Desktop\System Security\2021\openssl\1.0.2\openssl.exe
OpenSSL> dgst -sha256 -verify public.pem -signature sign.sha256 Secretfile.txt
Verified OK
OpenSSL>
```

Figure 16 Signature verification

If the verification is successful, the above OpenSSL command will print "Verified OK" message as can be seen in Figure 16, otherwise it will print "Verification Failure".

Tasks

1. Generate a 1024-bit RSA private key
2. Next generate the public key from the private key
3. Digitally sign the document (i.e. contract.txt)
4. Then verify the integrity of the digitally signed document (i.e. contract.txt)
5. Next make some changes to the contract.txt and again try to verify the document