Openssl Tutorial

Assignment 2

Kamal Shrestha, Srivathsa L. Rao

CS21MTECH16001, CS21MTECH12007

Feb 6, 2022

PART A: Secure file transfer between Alice (student A) and Bob (student B)

The two communicating parties for a secure file transfer are:

1. **Alice_CS21MTECH12007**
2. **Bob_CS21MTECH16001**

The following commands are to be run at each end of the communicating subjects with corresponding filenames until there is a mention of a specific code to be run at only one end.

Create RSA (2048) key pairs for Alice and Bob and exchange public keys over email. Password protect your respective private keys.

Each of the two communicating subjects generates the RSA(2048 bits) public and private key pairs with the following commands:

a. Creating private key

```
openssl genrsa -out key.pem 2048
```

b. Protecting the private key with DES-3 that requires a password to use the key

```
openssl rsa -in key.pem -des3 -out alice_private_key.pem
```

c. Generating the public key (alice_public_key.pem) from the private key (alice_private_key.pem)

```
openssl rsa -in alice_private_key.pem -pubout -out alice_public_key.pem
```

d. To view the contents of how the keys were generated

```
openssl rsa -in alice_private_key.pem -text -noout
```

```
Enter pass phrase for alice_private_key.pem:
RSA Private-Key: (2048 bit, 2 primes)
modulus:
    00:d6:20:c6:5d:ca:80:46:53:af:ef:48:22:be:14:
    b4:ce:c6:60:56:e0:19:3f:af:c8:b1:3a:04:ae:9f:
    88:2d:9e:bb:82:25:76:7f:4d:6b:e7:e4:0a:91:e0:
    1a:9a:0a:c7:2b:38:72:bb:0a:be:93:44:1d:bf:11:
    3f:21:ed:b4:d0:2b:35:b6:56:94:25:81:76:03:0d:
    73:da:a4:e0:c4:9a:65:76:68:a6:bb:16:cd:5a:1e:
    7e:0f:ea:09:36:a2:fa:3a:4d:b4:4b:c2:84:d2:e3:
    ad:4d:14:e4:03:2a:30:28:93:e4:de:90:27:ed:1e:
    03:b5:64:96:ba:1d:e8:d6:9c:ad:24:eb:db:25:68:
    6d:52:cf:3a:a8:8c:d8:17:34:77:ad:57:53:66:47:
    96:8d:8b:ec:8b:73:38:82:05:37:7a:b5:b1:f2:f9:
    ef:65:b4:5e:94:8a:1c:53:e4:54:b5:2e:10:df:b1:
    b4:2e:2f:49:41:51:d3:88:c6:39:85:39:0e:fb:40:
    44:73:ec:6a:19:64:4a:c0:ed:78:15:fc:65:73:92:
    7a:ad:9d:5f:74:74:3b:f7:fa:b3:c6:ae:16:a1:ec:
    2e:f9:f0:15:e2:e6:17:07:17:28:5a:db:ce:f3:99:
    04:3e:32:b6:03:e9:e3:10:92:42:65:84:47:ff:c0:
```

```
    06:61
publicExponent: 65537 (0x10001)
privateExponent:
    00:b3:66:6a:4c:12:69:3e:82:c7:18:77:1b:84:9d:
    dc:8e:91:6e:0d:db:5b:cc:1b:fe:cd:5d:8a:a6:b3:
    4a:68:ce:60:d3:3d:87:f6:48:6b:6b:b1:87:99:88:
    65:41:42:5e:74:3b:ab:04:42:90:40:da:05:02:1b:
    f5:04:fc:1f:4d:79:a6:5a:ca:56:fc:c5:66:96:50:
    23:0c:c0:af:00:89:4b:10:d5:c5:3b:62:73:52:59:
    d1:cc:8b:8c:84:96:2f:87:c1:aa:98:fe:b4:41:ec:
    37:3a:29:92:8f:84:3d:2d:02:1b:f2:cc:5c:67:31:
    8a:71:e3:80:d5:16:52:09:db:5a:63:0f:9d:45:18:
    f9:36:1f:ae:f8:28:75:5b:2c:66:e5:4e:5b:c5:9b:
    d7:96:a4:90:b7:79:b8:72:25:7e:c0:22:d2:0c:23:
    d9:95:57:38:57:60:be:52:ce:a0:50:b0:2f:c0:ef:
    f1:27:88:ea:53:7e:94:6e:9f:14:6b:65:5a:23:0b:
    87:93:24:53:36:d7:5f:e7:90:45:84:e1:54:0c:88:
    5c:6d:e6:81:d7:7a:83:66:5a:58:25:6c:ea:2b:d0:
    9a:bd:a6:70:4d:41:74:16:32:2f:be:64:06:04:0a:
    c5:3c:69:3c:df:dc:d4:1e:d2:f6:a2:3c:ee:76:5b:
    e1:f1
prime1:
    00:ee:56:97:b5:a2:17:6b:1f:a2:f0:6c:76:a4:55:
    91:57:35:23:c9:fe:2d:b8:c3:cf:89:c4:0c:62:3a:
    b1:9d:79:39:1b:e3:46:a3:07:46:f2:c1:0e:b9:cc:
    dd:f7:a8:6a:e1:3b:e4:1f:70:af:f5:fe:25:75:dc:
    c2:ba:07:26:e7:98:f2:11:62:b8:cd:34:33:49:a4:
```

```
    64:cc:c0:05:d5:79:76:cb:77:c3:1c:3b:9c:bf:a8:
    b3:8c:07:a2:39:17:fc:21:60:f5:26:65:2f:3b:37:
    32:28:ce:60:9a:2f:f3:4b:31:a9:48:21:c6:0b:6b:
    78:73:6d:39:74:7d:ad:0b:97
prime2:
    00:e5:fe:e6:a7:8d:3f:00:4d:47:28:9c:51:f7:63:
    dc:25:a0:a0:97:e0:f3:67:e0:05:6b:78:26:2e:14:
    9c:1f:f4:e8:df:78:5a:12:58:58:0f:50:25:e3:60:
    ca:09:6d:e2:d4:ff:a7:14:36:0e:51:ea:f0:5a:40:
    e4:d7:c9:25:21:bb:3c:90:48:71:08:6e:75:19:5c:
    50:5c:9c:84:e3:df:31:f7:47:97:d5:af:5b:5c:4d:
    a5:b6:ef:f0:a8:67:41:6d:55:3d:7c:6d:03:d1:80:
    28:68:1c:f5:86:85:d6:f3:84:c2:01:65:11:c0:a6:
    e0:74:70:b1:d7:93:b4:9c:c7
exponent1:
    00:b9:8c:46:6d:8c:34:69:1c:67:10:7f:90:59:dd:
    97:d9:e9:af:e4:18:72:e5:ed:e3:4b:a0:89:f7:8b:
    34:2d:a1:6b:39:6f:d5:d5:23:dc:33:2e:e3:54:f8:
    ce:31:79:37:44:04:09:54:04:b9:a8:6b:e4:23:fe:
    ea:c6:42:bd:21:fe:6b:2d:e7:ca:71:4e:db:42:d0:
    ad:fc:cc:dd:7d:d5:23:0b:c2:3f:ee:61:e6:65:3b:
    64:14:76:f8:ef:33:e2:00:e6:67:d4:2d:5f:f8:dc:
    be:bb:0f:f8:1a:f1:8b:9a:15:9e:71:5e:81:bc:f0:
    3d:04:b7:9a:cc:3a:ef:16:1f
exponent2:
    00:da:79:6e:45:0e:e9:1f:b2:48:bc:0c:f1:d7:9c:
    66:4c:df:ee:6d:17:64:5f:f1:ef:74:0c:e7:c7:b2:
```

```
    10:34:53:02:ba:f4:aa:2a:ee:fc:87:5f:4c:fe:56:
    bc:d4:84:2b:8a:c5:66:c2:ce:2e:80:26:3a:36:a1:
    9a:40:58:74:0b:3c:be:e5:17:cb:37:85:25:7a:f7:
    b3:e6:a5:4f:9e:de:2f:aa:83:b9:79:64:5f:d6:a8:
    73:97:f9:08:94:0f:b1:98:d7:f5:d6:32:00:04:8c:
    46:d5:cf:5c:73:72:c3:a3:03:22:ff:0c:30:f6:de:
    0e:2c:cd:b8:41:dd:af:1d:95
coefficient:
    00:96:8a:e1:ab:75:15:4e:f4:42:cc:80:5b:85:b2:
    b2:78:27:85:9b:e5:6f:ae:a7:7c:b1:f7:82:d4:8e:
    61:ba:cb:9d:75:98:a3:a5:70:02:f0:84:1b:ec:8d:
    ca:31:d3:be:5f:7b:f2:d1:ca:b9:7e:f7:00:5e:b6:
    06:25:88:7a:ee:f7:e6:58:2d:88:7f:ad:d5:48:04:
    9e:74:21:70:f4:42:8f:2c:84:17:f3:cf:bc:64:5e:
    94:15:db:ec:ce:0b:90:cc:b3:6f:65:88:64:dc:6c:
    cc:50:2e:ee:27:f0:50:3e:7f:aa:bc:28:60:c2:ef:
    33:18:4e:9d:bc:ea:fd:e1:bb
```

e. To view the contents of the public and private key we use the bash “cat” command:

```
cat alice_public_key.pem
```

```
-----BEGIN PUBLIC KEY-----
MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEA1iDGXcqARlOv70givhS0zsZgVuAZP6/IsToErp+ILZ67giV2f01r5+QKkeAa
mgrHKzhyuwq+k0QdvxE/Ie200Cs1tlaUJYF2Aw1z2qTgxJpldmimuxbNWh5+D+oJNqL6Ok20S8KE0uOtTRTkAyowKJPk3pAn7R4DtWSW
uh3o1pytJOvbJWhtUs86qIzYFzR3rVdTZkeWjYvsi3M4ggU3erWx8vnvZbRelIocU+RUtS4Q37G0Li9JQVHTiMY5hTkO+0BEc+xqGWRK
```

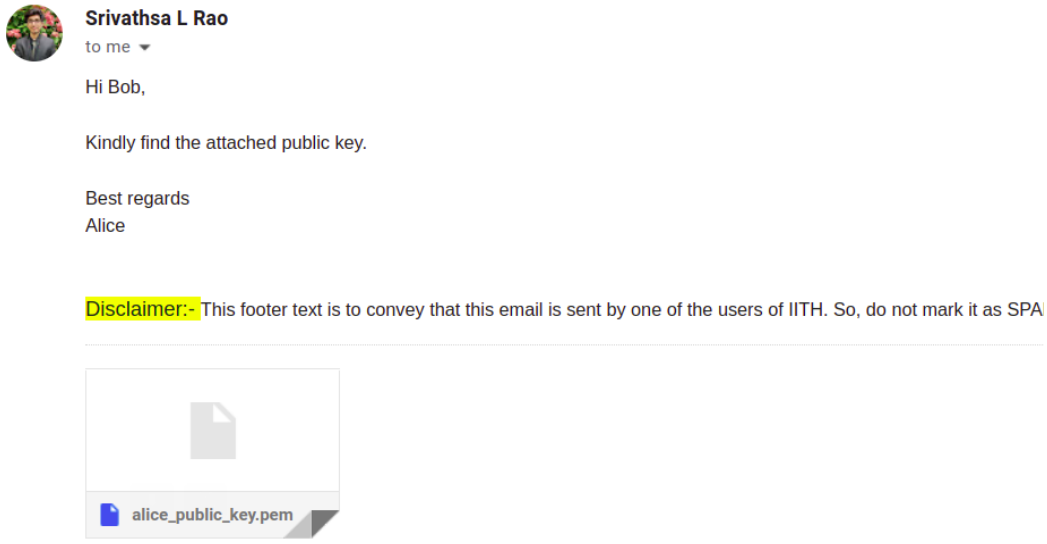
```
wO14Ffxlc5J6rZ1fdHQ79/qzxq4Woewu+fAV4uYXBxcoWtvO85kEPjK2A+njEJJCZYRH/8AGYQIDAQAB
-----END PUBLIC KEY-----
```

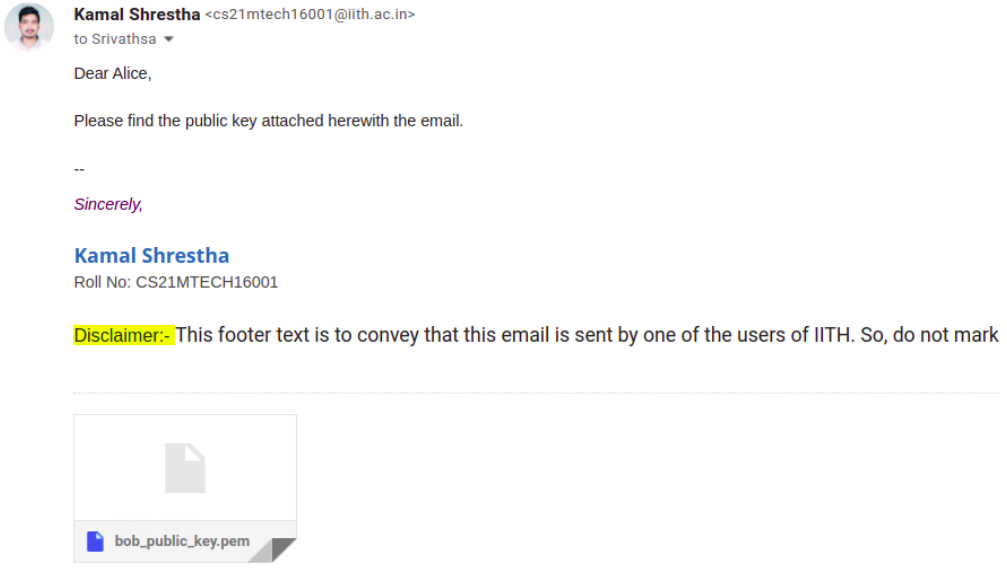
Similarly for Bob we have:

```
cat bob_public_key.pem
```

```
-----BEGIN PUBLIC KEY-----
MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEA8wK1lm2cV6UvAFvhd6B+y9AvELsOF/3teyFvLtPoGL1St5v2yxiTkyxG5O1z
pkFU7N2Yn//jVkKCC25nXF4rLs6k4TQuec2oq1qVrBWrI4fYsNghHEcpKF+4FNRGQ91yuhkRwiQDSSY2qlfVNWYUq+KKEdHdyjZRsMOo
RuCL2qaffbi2s9tRK3ZrGc5D8M6/O1dmUyB26IIZI80G6yQyCwPg2JHFqiEyRC8+83G+m8Du7Pz9FFKh08DuLXlAV4s+ujUDpSXKAnZq
hVVPUXlEs7v30T3cQjO+Ffbo48YG+Ed+3V7L+EjFJlOagBZ2iwsc2I8OVf7S7DgOtYtjI6yPjQIDAQAB
-----END PUBLIC KEY-----
```

So at the end of the first step, each of the communicating subjects has a pair of keys, public and password protected private key and the counter part pubic key (as descirbed in the figure below as well). The public keys are now exchanged via email.





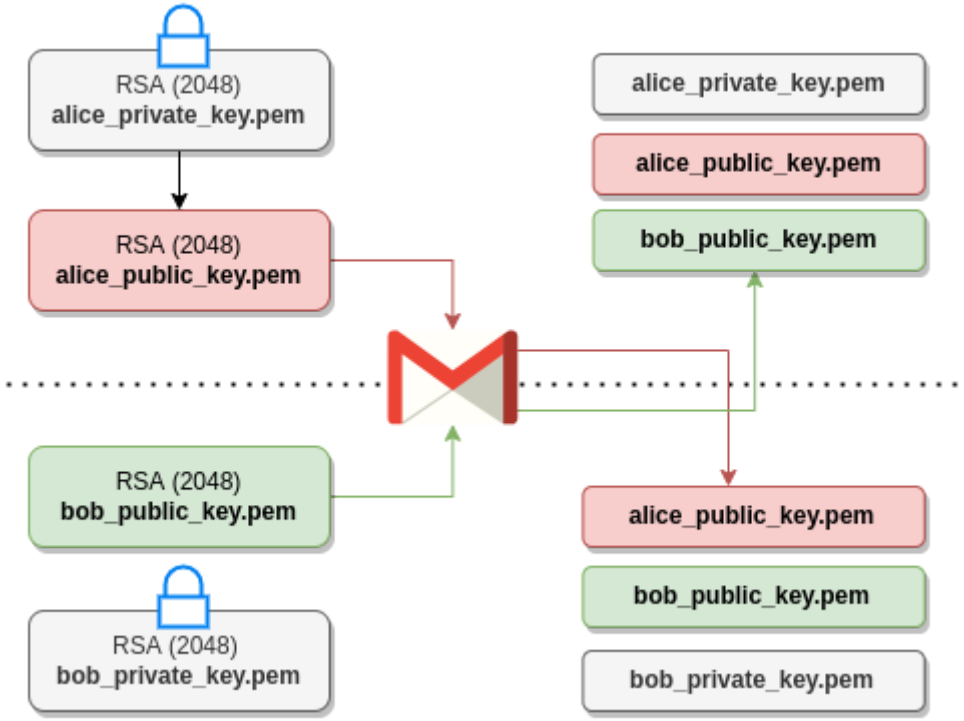
Files generated till now:

Alice_CS21MTECH12007

```
alice_private_key.pem, alice_public_key.pem, bob_public_key.pem
```

Bob_CS21MTECH16001

```
bob_private_key.pem, bob_public_key.pem, alice_public_key.pem
```



Alice creates a text file named SA.key with this info <symmetric encryption algo, its parameters, and passphrase>. Bob also does the same thing. These serve as keys for decrypting files exchanged in each way.

Now, both of the communicating parties create a file (SA.key or SB.key) containing information about the encryption algorithms, its parameter, and passphrase so that this information can be used to communicate securely.

Creating a passphrase txt file

```
gedit passphrase.txt
```

```
Hi! I am Alice
```

Creating an Initialization Vector (IV) and Key using the passphrase that will be used in the Symmetric Encryption Algorithm (AES-CBC mode) to encrypt and decrypt the files.

Alice's run:

```
openssl enc -aes-256-cbc -pbkdf2 -k passphrase.txt -P -md sha384


salt=EAE917758FAC3AF8
key=E1D0D0C702CB5329E088EA5A1BB35736CF9F513DC6FF2C3885D8F2FFF0E7A0D2
iv =0878CCA96402D95542E7DAA61C605BC8
```

Bob's run:

```
openssl enc -aes-256-cbc -pbkdf2 -k passphrase.txt -P -md sha384

salt=1C522614E5A95E08
key=05E572E36325669247EC5A59C6368B49FDB3270F24A87CE2C59262F18721B18F
iv =8D948654B542F218C77224ABBF2F8EE7
```

Each of them have the key, IV and passphrase for each communicating parties, and now will create their respective SA.key/SB.key files to be shared with each other. The files contained the following contents.

SA.key

```
Encryption algorithm=AES-256-CBC

salt=6DC46002AADEA0B1
key=4CAB24F0C3787F8BCEB62E057477AE54AF53B00B705F877A2AD0A4C8DACB2AE2
iv =D737F2FA6448BDC39DD41FE9D96D4196

Passphrase=Hi! I am Alice
```

SB.key

```
Encryption algorithm: AES-256-CBC

salt=05CC7A860A438ABC
key=5FD990D370E815AFEF47CC5452DFB610DA8C08D1DBFDF192019EC3EC8D962A50
iv =069BAE5053B5B297A454600C53E78A9D

Passphrase=Hi! I am Bob
```

Alice has to securely send SA.key to Bob. Devise a mechanism in such a way that only Bob can see that message and verify it indeed came from Alice without any tampering. Similarly, Bob has to securely send his SB.key to Alice and prove its authenticity and integrity.

Now, This information is used by both the communicating parties as the sole encryption and decryption algorithm during the entire communication. So, they need to securely send these files to their counterparts and for that, each of them:

From Alice Perspective (Same for Bob with change in filenames):

a. Sign the Digest of SA.key using Alice's Private Key = To prove that they are the ones sending the file and not Trudy (any intruders)
b. Encrypt with Bob's Public Key = To make sure noone can view or tamper the contents of the file except Bob

First we create a signed digest using a hashing algorithms:

Creating Digest of the SA.key/SB.key using **sha1 hashing algorithm:**

```
openssl dgst -sha1 -out sa_digest sa.key
```

Output:

```
SHA1(sa.key)= bd8b788a0e48f88208203bb831adfd3e503a43fc
```

Signing the digest using Alice's Private Key

```
openssl pkeyutl -sign -in sa_digest -out sa_sign -inkey alice_private_key.pem
```

Output of signed digest (sa_sign):

```
(:T'''x4'
        U'u}b 'N'/'8杏;''']('Q;'T''''
''8!E''''}'''%'';l'''''%'I'x':''ȝd~'h'''ϑ&'''+'>Qh7'q'''%''QWh~'''V]'' 8B'U''(M'4['8NC'/O''7t,9''''ڳj''
-'h0

'='V?'¯+

hJ''?'Z'A''(t'_'''N'M'fZ(}'e''''-'c'b''~'' ''<h*Wj%
```

Now, Alice encrypts the file SA.key using Bob's Public Key (*which was received earlier from mail*):

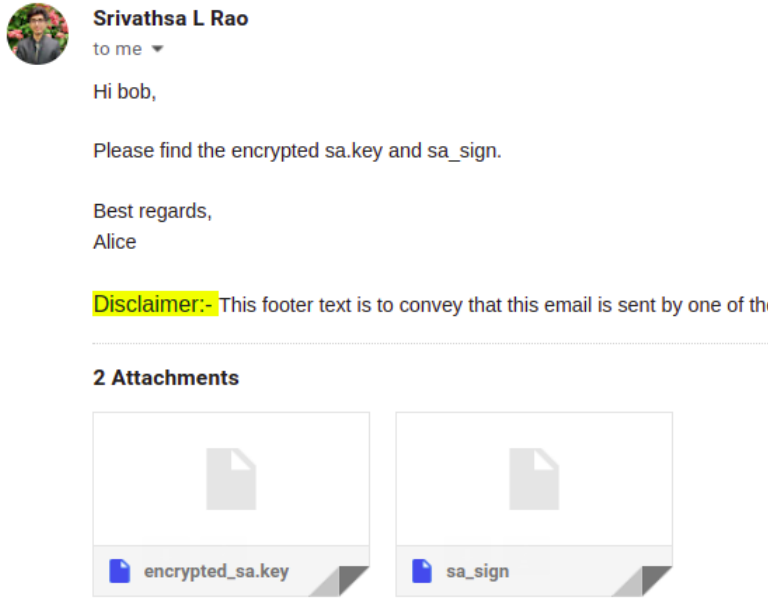
```
openssl pkeyutl -encrypt -in sa.key -inkey bob_public_key.pem -out encrypted_sa.key -pubin
```

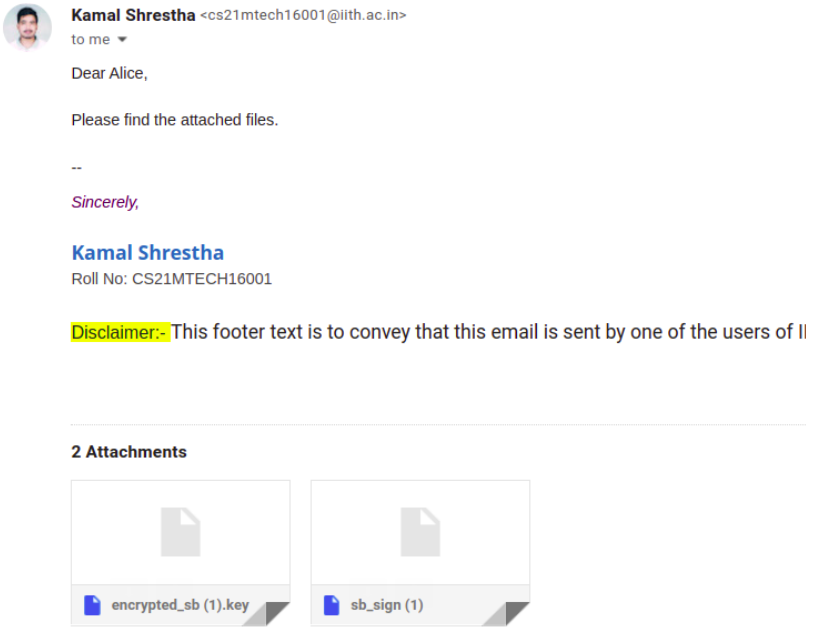
Ouput of encrypted_sa.key:

```
37
  ']i'''L%'''''''D''''?'''?'Y?3'}Ꮋ'''P粿
''''T?'C''.b̨'Ń''F''v'~'{9'd''|'k'g''D'0'[Mfn''['/''''9q\'m''xW$8'Ⅎ'%.jE'Jfsic?'9''['s'D'AÇ1/4'wq

'5aK'+'    '0'c
'Zyz'ɲ'Q''Z'''=,''''kO''vگ|' y'%n''R''''Y''dɵE''''''%
```

These encrypted and signed files from each communicating parties are exchanged over emails.





Complete set of files generated till now:

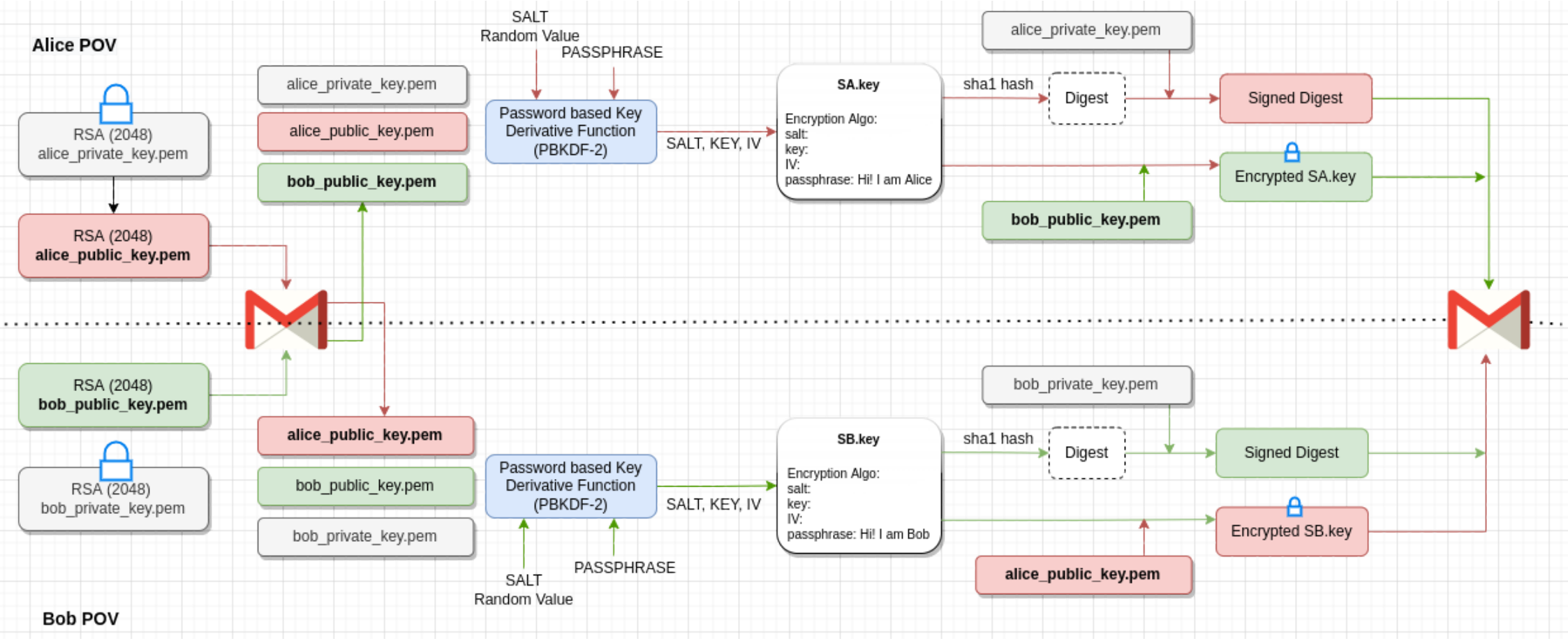
Alice_CS21MTECH12007

```
alice_private_key.pem, alice_public_key.pem, bob_public_key.pem, sa.key,sa_digest, sa_sign, encrypted_sa
```

Bob_CS21MTECH16001

```
bob_private_key.pem, bob_public_key.pem, alice_public_key.pem, sb.key, sb_digest, sb_sign, encrypted_sb
```

The complete steps till now can be visualized in the figure below:



After completing all these steps, each of them now has respective parameters for Symmetric Algorithm (AES-CBC-256) to encrypt and decrypt files and send over a communication channel securely.

Verification of SA.key/SB.key

Now each of them will conduct the verification of SA.key/SB.key on whether it has been tampered with or not and who is it sent by.
For that, they will:

a. Decrypt the SA/SB.key file using their own private key to get the actual SA/SB.key
b. Generate the Digest from SA/SB.key file
c. Verify the signed file containing digest with the digest generated using counterpart's public key.

From Alice's point of view:

Decrypt the SA/SB.key file using their own private key to get the actual SA/SB.key

```
openssl pkeyutl -decrypt -in encrypted_sb.key -inkey alice_private_key.pem -out sb.key
```

Output: Contents of sb.key

```
Encryption algorithm: AES-256-CBC

salt=05CC7A860A438ABC
key=5FD990D370E815AFEF47CC5452DFB610DA8C08D1DBFDF192019EC3EC8D962A50
iv =069BAE5053B5B297A454600C53E78A9D

Passphrase=Hi! I am Bob
```

Generate the Digest from SA/SB.key file

```
openssl dgst -sha1 -out sb_digest sb.key
```

Output: Contents of sb_digest

```
SHA1(sb.key)= 10fed27669e2d5584095cb027bcbb714e6e9db84
```

Verify the signed file containing digest with the digest generated using counterpart's public key.

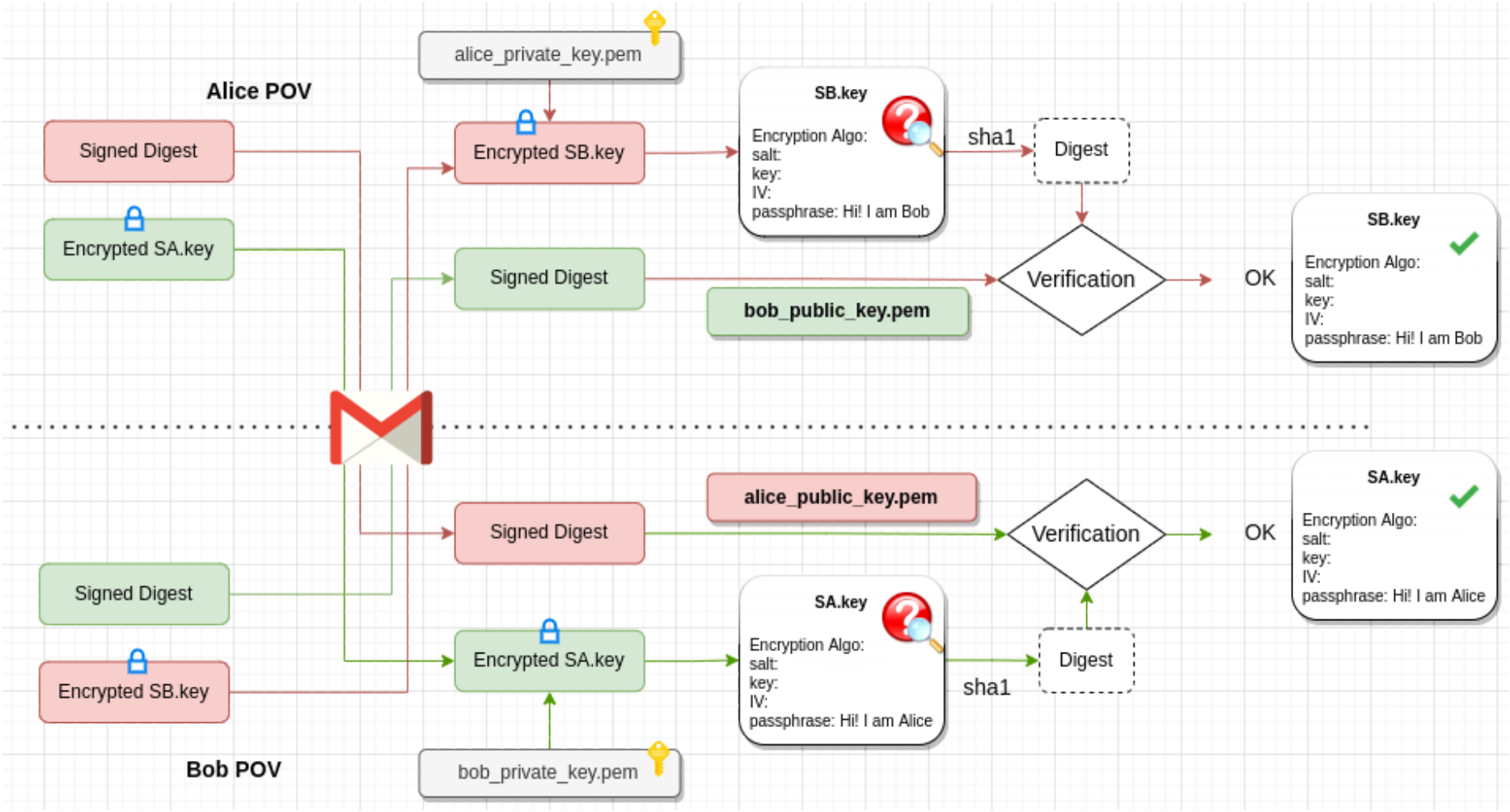
```
openssl pkeyutl -verify -sigfile sb_sign -in sb_digest -inkey bob_public_key.pem -pubin
```

Output:

```
Signature Verified Successfully
```

Now after completion of these steps, Alice has the SB.key sent by Bod by successfully verified and successfully checked for any sorts of tampering.

The verification process for the corresponding SA/SB.key is shown in the figure below:



At this point, **Alice has verified SB.key and the public key of Bob.** Similarly, **Bob has verified SA.key and public key of Alice.**
Now they can proceed to send files encrypting with corresponding SA/SB.key parameters using Symmetric Algorithms like AES in CBC mode.

Alice encrypts a large file (some PDF) with SA.key and sends it along with a signature to Bob so that he could decrypt it with the same SA.key and verify it indeed came from Alice. Similarly, Bob should send some large files securely to Alice without any tampering.

Each of the communicating parties will now use the counter-parts SA/SB.key to encrypt the large file. After the encryption, they will also sign the files using their own private key to send the signed and encrypted files over the communication channel (mail).

ENCRYPTION

Alice will be using SB.key to encrypt a PDF format file named: L4.pdf

Reference: Output of SB.key

```
Encryption algorithm: AES-256-CBC

salt=05CC7A860A438ABC
key=5FD990D370E815AFEF47CC5452DFB610DA8C08D1DBFDF192019EC3EC8D962A50
iv =069BAE5053B5B297A454600C53E78A9D

Passphrase=Hi! I am Bob
```

Encrypting L4.pdf using SB.key

```
openssl aes-256-cbc -K 5FD990D370E815AFEF47CC5452DFB610DA8C08D1DBFDF192019EC3EC8D962A50 -iv
069BAE5053B5B297A454600C53E78A9D -in L4.pdf -out encrypted_L4.bin
```

Generate the digest of the large file

```
openssl dgst -sha1 -out file-digest L4.pdf
```

Signing the encrypted file with Alice's Private key

```
openssl pkeyutl -sign -in file_digest -inkey alice_private_key.pem -out sign_file
```

Bob will be using SA.key to encrypt a PDF format file named: L3.pdf

Reference: Output of SA.key

```
Encryption algorithm=AES-256-CBC

salt=6DC46002AADEA0B1
key=4CAB24F0C3787F8BCEB62E057477AE54AF53B00B705F877A2AD0A4C8DACB2AE2
iv =D737F2FA6448BDC39DD41FE9D96D4196

Passphrase=Hi! I am Alice
```

Encrypting L3.pdf using SA.key

```
openssl aes-256-cbc -K 4CAB24F0C3787F8BCEB62E057477AE54AF53B00B705F877A2AD0A4C8DACB2AE2 -iv
D737F2FA6448BDC39DD41FE9D96D4196 -in L3.pdf -out encrypted_L3.bin
```

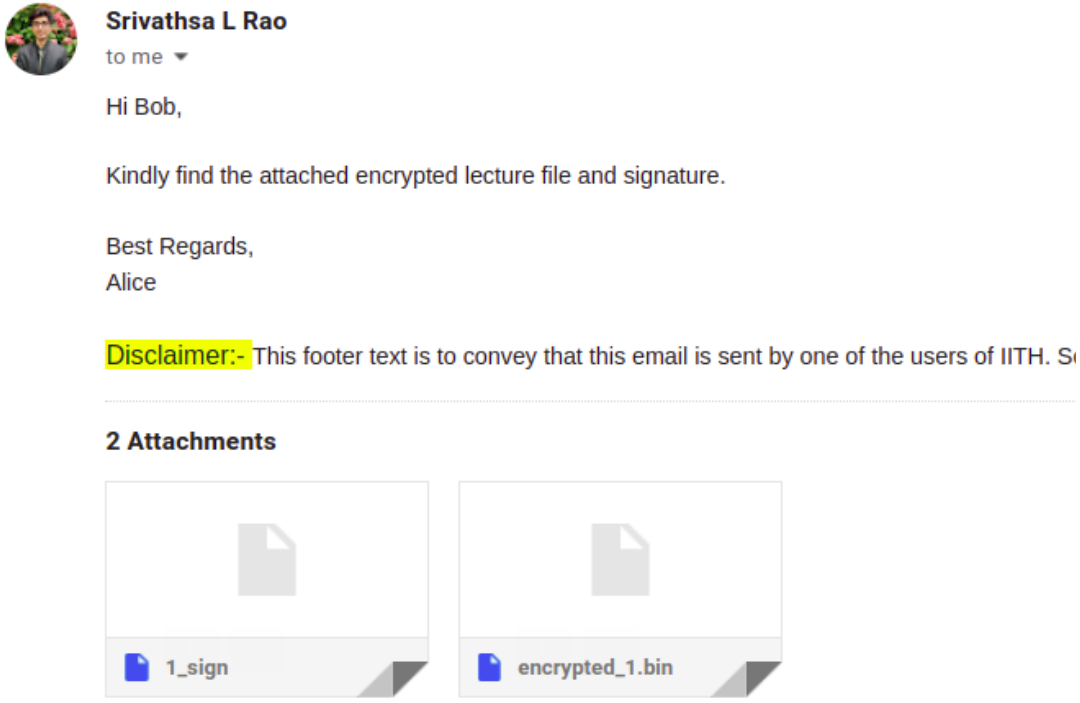
Generate the digest of the large file

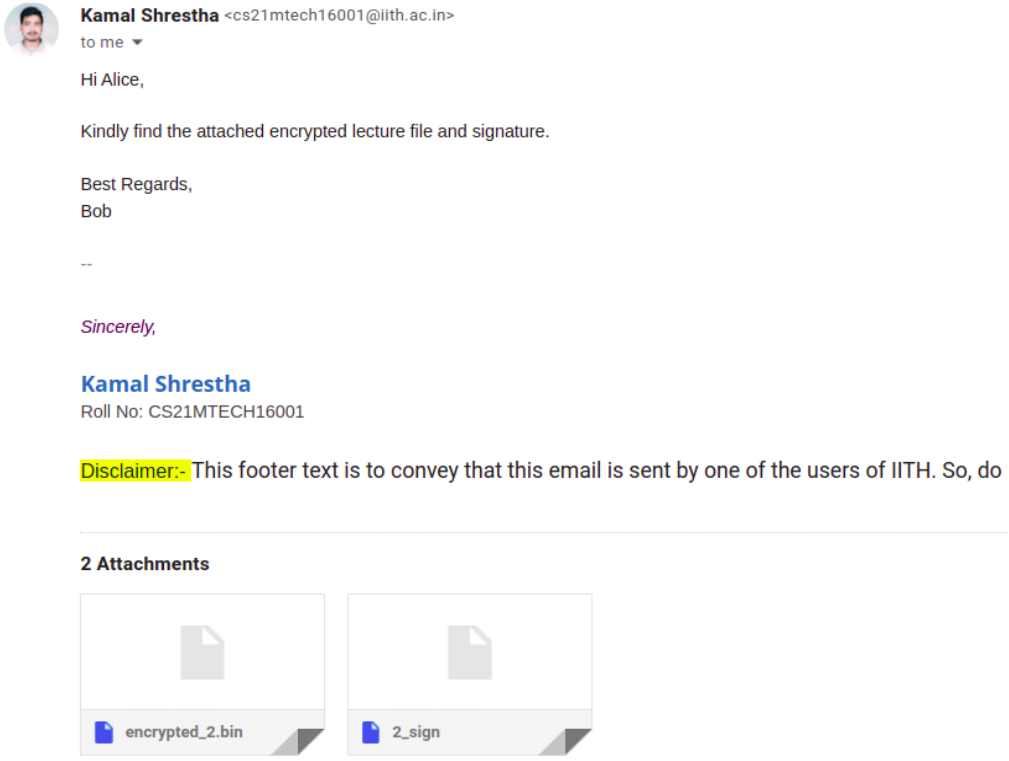
```
openssl dgst -sha1 -out file-digest L3.pdf
```

Signing the encrypted file with Alice's Private key

```
openssl pkeyutl -sign -in file_digest -inkey bob_private_key.pem -out sign_file
```

These encrypted and signed files are now shared via the communication channel, mail.





DECRYPTION

Alice will be using SA.key to decrypt a PDF format file named: L3.pdf

Reference: Output of SA.key

```
Encryption algorithm=AES-256-CBC

salt=6DC46002AADEA0B1
key=4CAB24F0C3787F8BCEB62E057477AE54AF53B00B705F877A2AD0A4C8DACB2AE2
iv =D737F2FA6448BDC39DD41FE9D96D4196

Passphrase=Hi! I am Alice
```

Decryption

```
openssl aes-256-cbc -d -K 4CAB24F0C3787F8BCEB62E057477AE54AF53B00B705F877A2AD0A4C8DACB2AE2 -iv
D737F2FA6448BDC39DD41FE9D96D4196 -in encrypted_L3.bin -out L3.pdf
```

Generating the digest

```
openssl dgst -sha1 -out L3_digest L3.pdf
```

Verifying the digest

```
openssl pkeyutl -verify -sigfile L3_sign -in L3_digest -inkey bob_public_key.pem -pubin
```

```
Signature Verified Successfully
```

Bob will be using SB.key to decrypt a PDF format file named: L4.pdf

Reference: Output of SB.key

```
Encryption algorithm: AES-256-CBC

salt=05CC7A860A438ABC
key=5FD990D370E815AFEF47CC5452DFB610DA8C08D1DBFDF192019EC3EC8D962A50
iv =069BAE5053B5B297A454600C53E78A9D

Passphrase=Hi! I am Bob
```

Decryption

```
openssl aes-256-cbc -d -K 5FD990D370E815AFEF47CC5452DFB610DA8C08D1DBFDF192019EC3EC8D962A50 -iv
069BAE5053B5B297A454600C53E78A9D -in encrypted_L4.bin -out L4.pdf
```

Generating the digest

```
openssl dgst -sha1 -out L4_digest L4.pdf
```

Verifying the digest

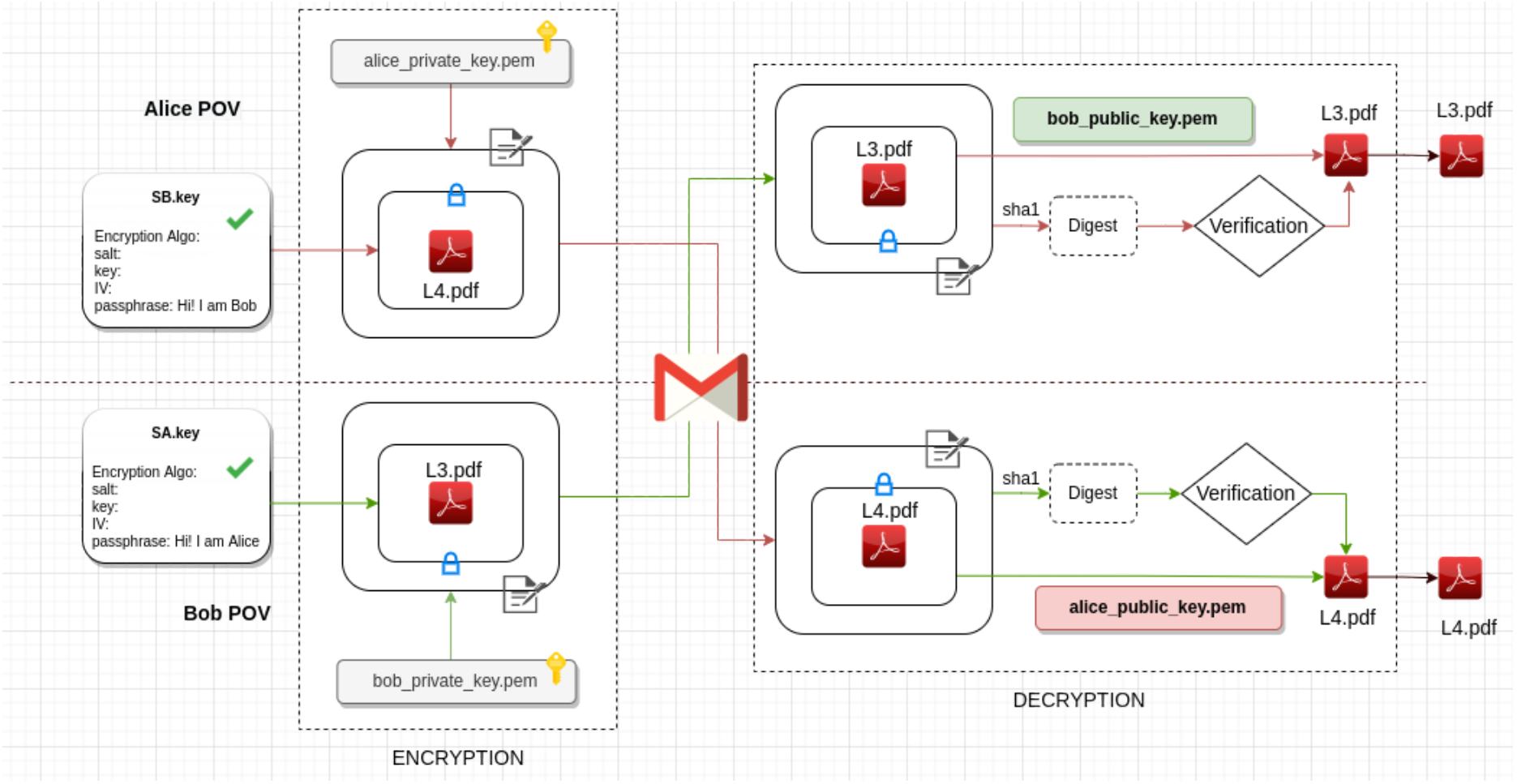
```
openssl pkeyutl -verify -sigfile L4_sign -in L4_digest -inkey alice_public_key.pem -pubin
```

Output:

```
Signature Verified Successfully
```

In this way, Alice and Bob has successfully sent an encrypted file over communication channel and verified its integrity and source.

The following figure shows the process of encryption and decryption of a large file between two parties after the SA/SB.key is verified.

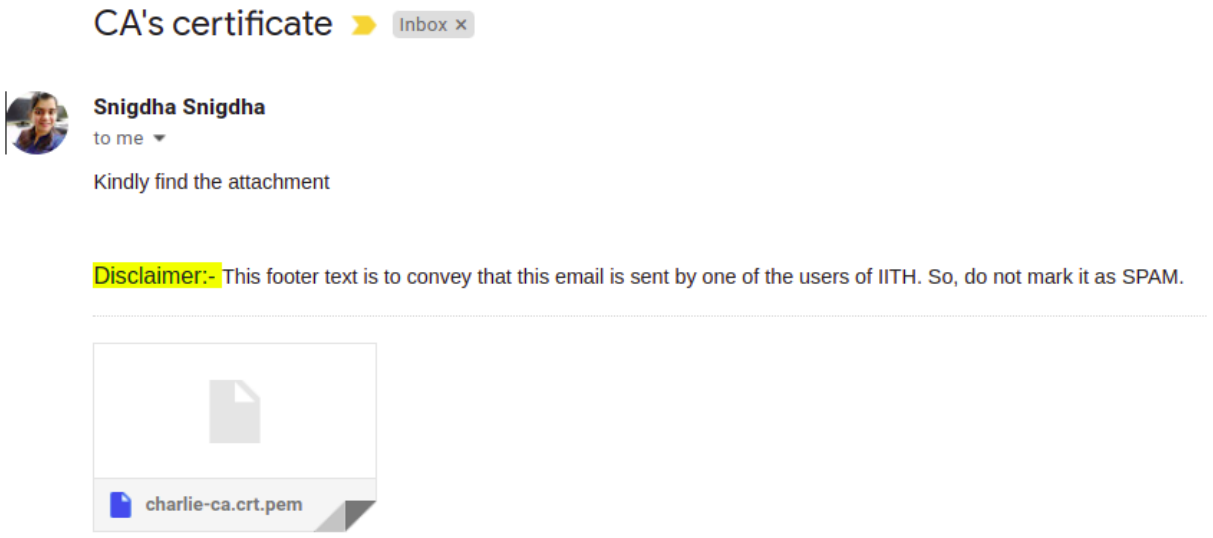


The complete workflow for PART A can be found here: [PART A]

Part B: Alice (Browser), Bob (webserver), and Charlie (Root CA)

Charlie (one of the TAs of this course) generates a self-signed certificate named charlie-ca.pem or charlie-ca.crt as s/he is the root CA.

Alice (Kamal Shrestha), has received the self-signed certificate from Charlie (Root CA/ TA).



Bob generates CSR named bob-browser.csr and emails it to Charlie for getting the end-user cert named bob-browser.crt. Note that CSR should contain all relevant details of Bob (Student B) like Country, Organization, CN/SAN/etc. Bob verifies bob-browser.crt is valid and signed by the root CA, Charlie.

Generating Certificate Signing Request

```
openssl req -newkey rsa:2048 -nodes -keyout bob_public_key.key -out bob.csr -config caconfig.cnf -reqexts v3_req
```

Output: Contents of CSR

```
-----BEGIN CERTIFICATE REQUEST-----
MIIDJDCCAgwCAQAwgYIxCzAJBgNVBAYTAklOMRIwEAYDVQQIDAlLYXJuYXRha2Ex
EjAQBgNVBAcMCUJhbmdhbG9yZTEUMBIGA1UECgwLSUlUSCBOUyBzdGQxCzAJBgNV
BAMMAktWMSgwJgYJKoZIhvcNAQkBFhljczIxbXRlY2gxMjAwN0BpaXRoLmFjLmlu
MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEA7cuzhUBIRPPYWPUpGQFr
wkP3wsx2bMnLTpjsJO9QWJy7R3CZbuZp2RQPn5rEbBBvI0cOIQaynIA8BdmeAMJ1
aLb4JHHtbJRZPexQQk5TqaUW1pYbXtSIUxvJ3jkWwwxI+JT9Gasu+vK1Si0PPhBB
/Q3PfahjLAjX0WC9pWKCmIqsL0gxXiwANFLV1SvAul79nfGMDhXn4bdBm8fCAjVg
b0coAC9qeoFglsu6RST+zixts7kyrmOhquQa7wrhHKE0CdF60e2wY2JurZLTLSIw
evjFh1YmqPkHSdPIoekW2LWS0exu2/gUNEN0B+EZFOVIQJeg5jAwOAT3DMwTAEkF
VwIDAQABoFwwWgYJKoZIhvcNAQkOMU0wSzAJBgNVHRMEAjAAMAsGA1UdDwQEAwIF
4DAxBgNVHSUEKjAoBggrBgEFBQcDAQYIKwYBBQUHAwIGCCsGAQUFBwMDBggrBgEF
BQcDBDANBgkqhkiG9w0BAQsFAAOCAQEA3pLMGZ52LKXdKpq2di89UAqJpcC6HQDX
oFSRNR6EQDw8xLhCz/VNiqSooHw+GRrIlBTsIjxmC852794sWk3BhS370liteAqi
sM6ZHITI6L798xqgQ9wPFJ4E3tXwYZeFZGTRUS1BUQexJtMkgNHFdMMFWiscCpar
qnBOhbLmA8c6vjDPH1sMOyDrv3Bnx3Icjn7wYuFuieBlbPau8TSOVtHhOn5QGp+z
A9jERMnl7ZO6zhxDrHsvs2pvByjFjEdwW0mTdX/XJnKE7uWMnOHppkekN3zwl7iG
wjq8qbVjVyIMqdGArsY+vo/FZzwyvRAa2/b9ZgfWjSyodEMyCA4S8A==
-----END CERTIFICATE REQUEST-----
```

Screenshot of the contents of CSR

KV

Certificate request
Identity: KV

Details

Subject Name

C (Country):	IN
ST (State):	Karnataka
L (Locality):	Bangalore
O (Organization):	IITH NS std
CN (Common Name):	KV
EMAIL (Email Address):	cs21mtech12007@iith.ac.in

Certificate request

Type:	PKCS#10
Version:	1

Public Key Info

Key Algorithm:	RSA
Key Parameters:	05 00
Key Size:	2048
Key SHA1 Fingerprint:	B3 B5 DF 65 85 68 F7 B9 AD 32 31 4D 50 E9 9F 50 5E DE 11 FE
Public Key:	30 82 01 0A 02 82 01 01 00 ED CB B3 85 40 48 44 F3 D8 58 F5 29 19 01 6B C2 43 F7 C2 CC 76 6C C9 CB 4E 98 EC 24 EF 50 58 9C BB 47 70 99 6E E6 69 D9 14 0F 9F 9A C4 6C 10 6F 23 47 0E 21 06 B2 9C 80 3C 05 D9 9E 00 C2 75 68 B6 F8 24 71 ED 6C 94 59 3D EC 50 42 4E 53 A9 A5 16 D6 96 1B 5E D4 88 53 1B C9 DE 39 16 C3 0C 48 F8 94 FD 19 AB 2E FA F2 B5 4A 2D 0F 3E 10 41 FD 0D CF 7D A8 63 2C 08 D7 D1 60 BD A5 62 82 98 8A AC 2F 48 31 5E 2C 00 34 52 D5 D5 2B C0 BA 5E FD 9D F1 8C 0E 15 E7 E1 B7 41 9B C7 C2 02 35 60 6F 47 28 00 2F 6A 7A 81 60 96 CB BA 45 24 FE CE 2C 6D B3 B9 32 AE 63 A1 AA E4 1A EF 0A E1 1C A1 34 09 D1 7A D1 ED B0 63 62 6E AD 92 D3 2D 22 30 7A F8 C5 87 56 26 A8 F9 07 49 D3 C8 A1 E9 16 D8 B5 92 D1 EC 6E DB F8 14 34 43 74 07 E1 19 14 E5 48 40 97 A0 E6 30 30 38 04 F7 0C CC 13 00 49 05 57 02 03 01 00 01

Basic Constraints

Certificate Authority:	No
Max Path Length:	Unlimited
Critical:	No

Key Usage

Usages:	Digital signature ↵ Key encipherment
Critical:	No

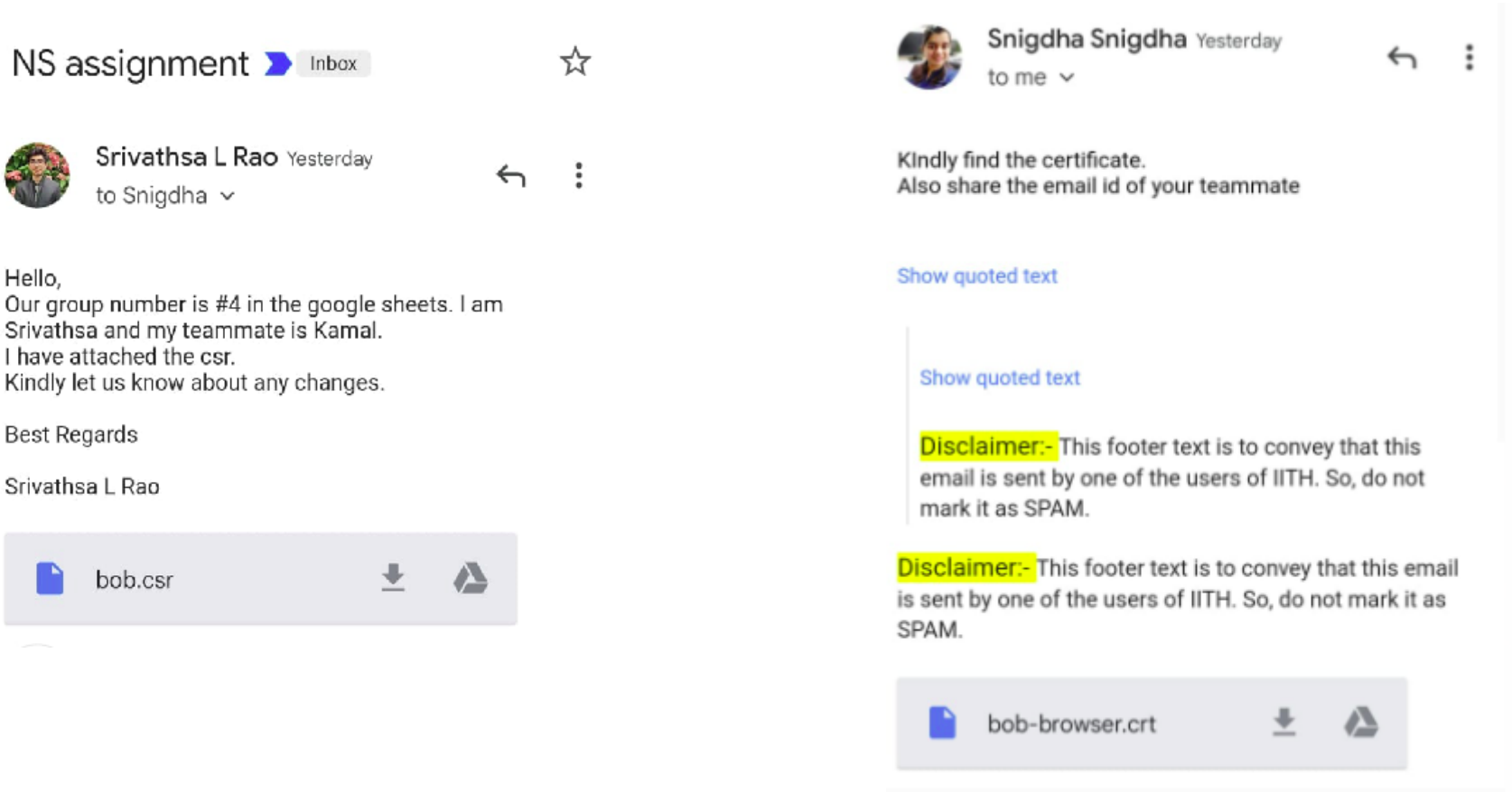
Extended Key Usage

Allowed Purposes:	Server Authentication ↵ Client Authentication ↵ Code Signing ↵ Email Protection
Critical:	No

Signature

Signature Algorithm:	1.2.840.113549.1.1.11
Signature Parameters:	05 00
Signature:	DE 92 CC 19 9E 76 2C A5 DD 2A 9A B6 76 2F 3D 50 0A 89 A5 C0 BA 1D 00 D7 A0 54 91 35 1E 84 40 3C 3C C4 B8 42 CF F5 4D 8A A4 A8 A0 7C 3E 19 1A C8 94 14 EC 22 3C 66 0B CE 76 EF DE 2C 5A 4D C1 85 2D FB D2 58 AD 78 0A A2 B0 CE 99 1C 84 C8 E8 BE FD F3 1A A0 43 DC 0F 14 9E 04 DE D5 F0 61 97 85 64 64 D1 51 2D 41 51 07 B1 26 D3 24 80 D1 C5 74 C3 05 5A 2B 1C 0A 96 AB AA 70 4E 85 B2 E6 03 C7 3A BE 30 CF 1F 5B 0C 3B 20 EB BF 70 67 C7 72 1C 8E 7E F0 62 E1 6E 89 E0 65 6C F6 AE F1 34 8E 56 D1 E1 3A 7E 50 1A 9F B3 03 D8 C4 44 C9 E5 ED 93 BA CE 1C 43 AC 7B 2F B3 6A 6F

This CSR is now sent to the CA to get it certified and Bob will be receiving the certificate signed by CA.



Contents of the **bob-browser.crt**

```
-----BEGIN CERTIFICATE-----
MIIE5jCCAs6gAwIBAgIUbSOHElopHWPwB65I7etNUdC+i2wwDQYJKoZIhvcNAQEL
BQAwgYcxCzAJBgNVBAYTAklOMRIwEAYDVQQIDAlUZWxhbmdhbmExEzARBgNVBAcM
ClNhbmdhcmVkZHkxDTALBgNVBAoMBElJVEgxDDAKBgNVBAsMA0NTRTEQMA4GA1UE
AwwHUm9vdF9DQTEgMB4GCSqGSIb3DQEJARYRY2hhcmxpZUBlbWFpbC5jb20wHhcN
MjIwMjA1MDQyNTA1WhcNMzIwMjAzMDQyNTA1WjCBgjELMAkGA1UEBhMCSU4xEjAQ
BgNVBAgMCUthcm5hdGFrYTESMBAGA1UEBwwJQmFuZ2Fsb3JlMRQwEgYDVQQKDAtJ
SVRIIE5TIHN0ZDELMAkGA1UEAwwCS1YxKDAmBgkqhkiG9w0BCQEWGWNzMjFtdGVj
aDEyMDA3QGlpdGguYWMuaW4wggEiMA0GCSqGSIb3DQEBAQUAA4IBDwAwggEKAoIB
AQDty7OFQEhE89hY9SkZAWvCQ/fCzHZsyctOmOwk71BYnLtHcJlu5mnZFA+fmsRs
EG8jRw4hBrKcgDwF2Z4AwnVotvgkce1slFk97FBCTlOppRbWlhte1IhTG8neORbD
DEj4lP0Zqy768rVKLQ8+EEH9Dc99qGMsCNfRYL2lYoKYiqwvSDFeLAA0UtXVK8C6
Xv2d8YwOFefht0Gbx8ICNWBvRygAL2p6gWCWy7pFJP7OLG2zuTKuY6Gq5BrvCuEc
oTQJ0XrR7bBjYm6tktMtIjB6+MWHViao+QdJ08ih6RbYtZLR7G7b+BQ0Q3QH4RkU
5UhAl6DmMDA4BPcMzBMASQVXAgMBAAGjTTBLMDEGA1UdJQQqMCgGCCsGAQUFBwMB
BggrBgEFBQcDAgYIKwYBBQUHAwMGCCsGAQUFBwMEMAkGA1UdEwQCMAAwCwYDVR0P
BAQDAgXgMA0GCSqGSIb3DQEBCwUAA4ICAQCKPfkpkh4l+qZ254Es+pvC+tNQQ4tE
vq1Q7LKSbasXPG4sRFHfSCkOJmTMxdfP9NWE6fzzu46gCSwa198Ov3kZG5bNw79m
nZQJ+Fl8mar0CXvfj2e7Lr+96JGkyO0sFrn7CsNCon10Ua64UosS3/6dY2KWb8o0
mwn9FPaMgGMk0V3oQrexPV7ZN4v1gDm+8YuI46goUYJ9TFT8iBfJkINiPIOk8zl/
xIEAmcUzGEaz+hIv9+FRbv/BaF3nDpOYiFoPbjV+DFz2eFS9Gg06UGBdA+SFMJ26
cmWHlkn1C8i1eYCD9wnbPGNemvBXxBA817p8h/e1jOHl+WvY8LwEOaX0zHmlxJGo
lGVlwojjueORTy6SFFaY3JCjdIqN7h799B7oWAw9Fjf9NeU+ybi2r537zMFDaWZC
7KJVPySIXcTlqoLUTKCx4jZ72ghaAEb1EhnXeJOOf8NXdAE5pwNVhhj0j/pHqOe7
zrdzhOxelkK4tyolp6Ccy+YLtyEKAkHbmcABw1LTpQajK+4/Wdcn6kIpFWZNRDT5
4PuNgVUe5txNvN99/ZxZPq5y+V5Y+MM1jyEV6bZUrRLtKRZ16ILPqPs8Yo0toF34
bbqtMouYPB28Hi+CRVi3NOPRirL5RU/nQu9J9uGoO6peszj4Tq1dgFpsC4qPRTQn
6XAWRR+8hAr6AA==
-----END CERTIFICATE-----
```

Screenshot of the **bob-browser.crt**

KV

Identity: KV
Verified by: Root_CA
Expires: 3/2/32

Details

Subject Name

C (Country):	IN
ST (State):	Karnataka
L (Locality):	Bangalore
O (Organization):	IITH NS std
CN (Common Name):	KV
EMAIL (Email Address):	cs21mtech12007@iith.ac.in

Issuer Name

C (Country):	IN
ST (State):	Telangana
L (Locality):	Sangareddy
O (Organization):	IITH
OU (Organizational Unit):	CSE
CN (Common Name):	Root_CA
EMAIL (Email Address):	charlie@email.com

Issued Certificate

Version:	3
Serial Number:	6D 23 87 12 5A 29 1D 63 F0 07 AE 48 ED EB 4D 51 D0 BE 8B 6C
Not Valid Before:	2022-02-05
Not Valid After:	2032-02-03

Certificate Fingerprints

SHA1:	03 DA E0 35 03 80 25 72 B3 3B 8E 5C 0F 10 2C CC 6C 7D E0 BD
MD5:	67 7B DD 9E A1 86 2D 42 47 3F 1D C6 BD 95 EA 57

Public Key Info

Key Algorithm:	RSA
Key Parameters:	05 00
Key Size:	2048
Key SHA1 Fingerprint:	B3 B5 DF 65 85 68 F7 B9 AD 32 31 4D 50 E9 9F 50 5E DE 11 FE
Public Key:	30 82 01 0A 02 82 01 01 00 ED CB B3 85 40 48 44 F3 D8 58 F5 29 19 01 6B C2 43 F7 C2 CC 76 6C C9 CB 4E 98 EC 24 EF 50 58 9C BB 47 70 99 6E E6 69 D9 14 0F 9F 9A C4 6C 10 6F 23 47 0E 21 06 B2 9C 80 3C 05 D9 9E 00 C2 75 68 B6 F8 24 71 ED 6C 94 59 3D EC 50 42 4E 53 A9 A5 16 D6 96 1B 5E D4 88 53 1B C9 DE 39 16 C3 0C 48 F8 94 FD 19 AB 2E FA F2 B5 4A 2D 0F 3E 10 41 FD 0D CF 7D A8 63 2C 08 D7 D1 60 BD A5 62 82 98 8A AC 2F 48 31 5E 2C 00 34 52 D5 D5 2B C0 BA 5E FD 9D F1 8C 0E 15 E7 E1 B7 41 9B C7 C2 02 35 60 6F 47 28 00 2F 6A 7A 81 60 96 CB BA 45 24 FE CE 2C 6D B3 B9 32 AE 63 A1 AA E4 1A EF 0A E1 1C A1 34 09 D1 7A D1 ED B0 63 62 6E AD 92 D3 2D 22 30 7A F8 C5 87 56 26 A8 F9 07 49 D3 C8 A1 E9 16 D8 B5 92 D1 EC 6E DB F8 14 34 43 74 07 E1 19 14 E5 48 40 97 A0 E6 30 30 38 04 F7 0C CC 13 00 49 05 57 02 03 01 00 01

Extended Key Usage

Allowed Purposes:	Server Authentication ↵ Client Authentication ↵ Code Signing ↵ Email Protection
Critical:	No

Extended Key Usage

Allowed Purposes:	Server Authentication ↵ Client Authentication ↵ Code Signing ↵ Email Protection
Critical:	No

Basic Constraints

Certificate Authority:	No
Max Path Length:	Unlimited
Critical:	No

Key Usage

Usages:	Digital signature ↵ Key encipherment
Critical:	No

Signature

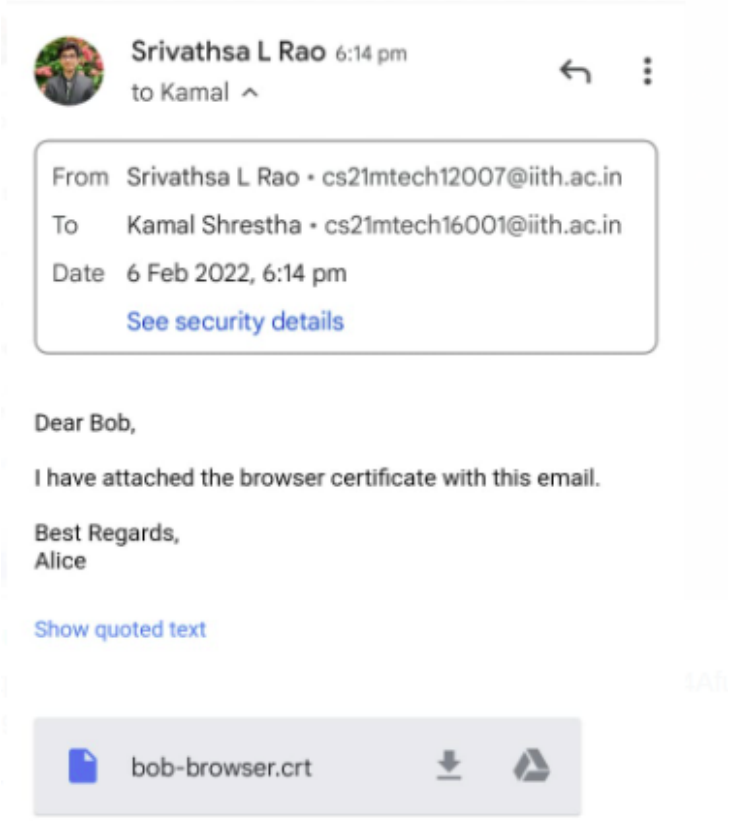
Signature Algorithm:	1.2.840.113549.1.1.11
Signature Parameters:	05 00
Signature:	8A 3D F9 29 92 1E 25 FA A6 76 E7 81 2C FA 9B C2 FA D3 50 43 8B 44 BE AD 50 EC B2 92 6D AB 17 3C 6E 2C 44 51 DF 48 29 0E 26 64 CC C5 D7 CF F4 D5 84 E9 FC F3 BB 8E A0 09 2C 1A D7 DF 0E BF 79 19 1B 96 CD C3 BF 66 9D 94 09 F8 59 7C 99 AA F4 09 7B DF 8F 67 BB 2E BF BD E8 91 A4 C8 ED 2C 16 B9 FB 0A C3 42 A2 7D 74 51 AE B8 52 8B 12 DF FE 9D 63 62 96 6F CA 34 9B 09 FD 14 F6 8C 80 63 24 D1 5D E8 42 B7 B1 3D 5E D9 37 8B F5 80 39 BE F1 8B 88 E3 A8 28 51 82 7D 4C 54 FC 88 17 C9 90 83 62 3C 83 A4 F3 39 7F C4 81 00 99 C5 33 18 46 B3 FA 12 2F F7 E1 51 6E FF C1 68 5D E7 0E 93 98 88 5A 0F 6E 35 7E 0C 5C F6 78 54 BD 1A 0D 3A 50 60 5D 03 E4 85 30 9D BA 72 65 87 96 49 F5 0B C8 B5 79 80 83 F7 09 DB 3C 63 5E 9A F0 57 C4 10 3C D7 BA 7C 87 F7 B5 8C E1 E5 F9 6B D8 F0 BC 04 39 A5 F4 CC 79 A5 C4 91 A8 94 65 65 C2 88 E3 B9 E3 91 4F 2E 92 14 56 98 DC 90 A3 74 8A 8D EE 1E FD F4 1E E8 58 0C 3D 16 37 FD 35 E5 3E C9 B8 B6 AF 9D FB CC C1 43 69 66 42 EC A2 55 3F 24 88 5D C4 E5 AA 82 D4 4C A0 B1 E2 36 7B DA 08 5A 00 46 F5 12 19 D7 78 93 8E 7F C3 57 74 01 39 A7 03 55 86 18 F4 8F FA 47 A8 E7 BB CE B7 73 84 EC 5E 96 42 B8 B7 2A 25 A7 A0 9C CB E6 0B B7 21 0A 02 41 DB 99 C0 01 C3 52 D3 A5 06 A3 2B EE 3F 59 D7 27 EA 42 29 15 66 4D 44 34 F9 E0 FB 8D 81 55 1E E6 DC 4D BC DF 7D FD 9C 59 3E AE 72 F9 5E 58 F8 C3 35 8F 21 15 E9 B6 54 AD 12 ED 29 16 75 E8 82 CF A8 FB 3C 62 8D 2D A0 5D F8 6D BA AD 32 8B 98 3C 1D BC 1E 2F 82 45 58 B7 34 E3 D1 8A B2 F9 45 4F E7 42 EF 49 F6 E1 A8 3B AA 5E B3 38 F8 4E AD 5D 80 5A 6C 0B 8A 8F 45 34 27 E9 70 16 45 1F BC 84 0A FA 00

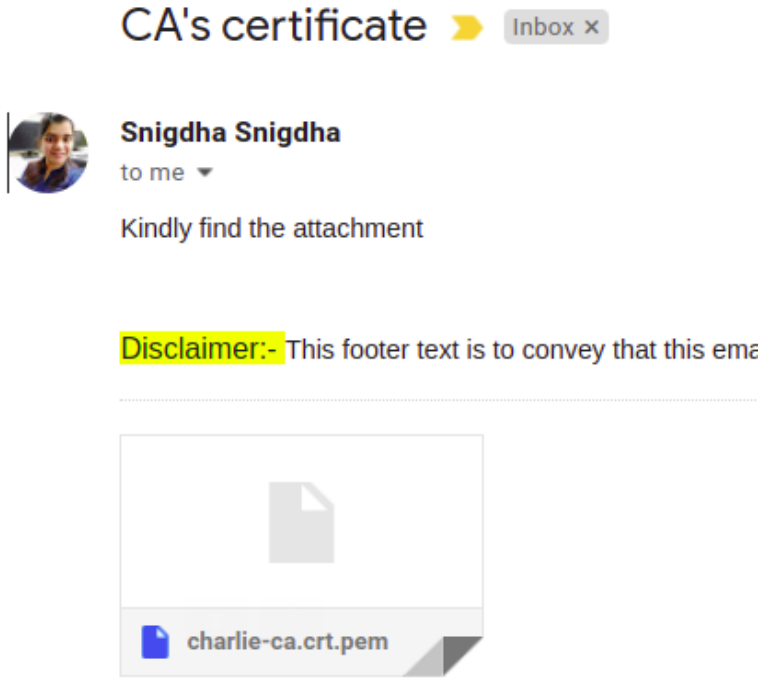
Now Bob has received the Certificate for the CSR he submitted to the CA. The Certificate has been signed by the CA and can be now used to establish a secure connection.

Now, the only thing left is for Alice verify that the certificate that bob is using for his webserver is valid.

Alice (Student A) gets charlie-ca.crt over email from Charlie and bob-browser.crt over email from Bob and verify that Bob's certificate is valid and signed by the root CA, Charlie. Comment on whether Bob's cert is of type X.509 V3, what is the serial no assigned, and what are the key usages/constraints associated with the cert.

Alice (Kamal Shrestha) receives the certificate (bob-browser.crt) from Bob (Sri Vathsa L. Rao) and CA's Certificate from TA.





Verification of Certificate

```
openssl x509 -verify -CAfile charlie-ca.crt.pem bob-browser.crt
```

Output:

```
bob-browser.crt: OK
```

Contents of bob-browser.crt

```
openssl x509 -text -noout -in bob-browser.crt
```

Output:

```
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            6d:23:87:12:5a:29:1d:63:f0:07:ae:48:ed:eb:4d:51:d0:be:8b:6c
        Signature Algorithm: sha256WithRSAEncryption
        Issuer: C = IN, ST = Telangana, L = Sangareddy, O = IITH, OU = CSE, CN = Root_CA, emailAddress = charlie@email.com
        Validity
            Not Before: Feb  5 04:25:05 2022 GMT
            Not After : Feb  3 04:25:05 2032 GMT
        Subject: C = IN, ST = Karnataka, L = Bangalore, O = IITH NS std, CN = KV, emailAddress = cs21mtech12007@iith.ac.in
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                RSA Public-Key: (2048 bit)
                Modulus:
                    00:ed:cb:b3:85:40:48:44:f3:d8:58:f5:29:19:01:
                    6b:c2:43:f7:c2:cc:76:6c:c9:cb:4e:98:ec:24:ef:
```

```
                50:58:9c:bb:47:70:99:6e:e6:69:d9:14:0f:9f:9a:
                c4:6c:10:6f:23:47:0e:21:06:b2:9c:80:3c:05:d9:
                9e:00:c2:75:68:b6:f8:24:71:ed:6c:94:59:3d:ec:
                50:42:4e:53:a9:a5:16:d6:96:1b:5e:d4:88:53:1b:
                c9:de:39:16:c3:0c:48:f8:94:fd:19:ab:2e:fa:f2:
                b5:4a:2d:0f:3e:10:41:fd:0d:cf:7d:a8:63:2c:08:
                d7:d1:60:bd:a5:62:82:98:8a:ac:2f:48:31:5e:2c:
                00:34:52:d5:d5:2b:c0:ba:5e:fd:9d:f1:8c:0e:15:
                e7:e1:b7:41:9b:c7:c2:02:35:60:6f:47:28:00:2f:
                6a:7a:81:60:96:cb:ba:45:24:fe:ce:2c:6d:b3:b9:
                32:ae:63:a1:aa:e4:1a:ef:0a:e1:1c:a1:34:09:d1:
                7a:d1:ed:b0:63:62:6e:ad:92:d3:2d:22:30:7a:f8:
                c5:87:56:26:a8:f9:07:49:d3:c8:a1:e9:16:d8:b5:
                92:d1:ec:6e:db:f8:14:34:43:74:07:e1:19:14:e5:
                48:40:97:a0:e6:30:30:38:04:f7:0c:cc:13:00:49:
                05:57
            Exponent: 65537 (0x10001)
    X509v3 extensions:
        X509v3 Extended Key Usage:
            TLS Web Server Authentication, TLS Web Client Authentication, Code Signing, E-mail Protection
        X509v3 Basic Constraints:
            CA:FALSE
        X509v3 Key Usage:
            Digital Signature, Non Repudiation, Key Encipherment
Signature Algorithm: sha256WithRSAEncryption
     8a:3d:f9:29:92:1e:25:fa:a6:76:e7:81:2c:fa:9b:c2:fa:d3:
     50:43:8b:44:be:ad:50:ec:b2:92:6d:ab:17:3c:6e:2c:44:51:
     df:48:29:0e:26:64:cc:c5:d7:cf:f4:d5:84:e9:fc:f3:bb:8e:
     a0:09:2c:1a:d7:df:0e:bf:79:19:1b:96:cd:c3:bf:66:9d:94:
     09:f8:59:7c:99:aa:f4:09:7b:df:8f:67:bb:2e:bf:bd:e8:91:
     a4:c8:ed:2c:16:b9:fb:0a:c3:42:a2:7d:74:51:ae:b8:52:8b:
     12:df:fe:9d:63:62:96:6f:ca:34:9b:09:fd:14:f6:8c:80:63:
```

```
        24:d1:5d:e8:42:b7:b1:3d:5e:d9:37:8b:f5:80:39:be:f1:8b:
        88:e3:a8:28:51:82:7d:4c:54:fc:88:17:c9:90:83:62:3c:83:
        a4:f3:39:7f:c4:81:00:99:c5:33:18:46:b3:fa:12:2f:f7:e1:
        51:6e:ff:c1:68:5d:e7:0e:93:98:88:5a:0f:6e:35:7e:0c:5c:
        f6:78:54:bd:1a:0d:3a:50:60:5d:03:e4:85:30:9d:ba:72:65:
        87:96:49:f5:0b:c8:b5:79:80:83:f7:09:db:3c:63:5e:9a:f0:
        57:c4:10:3c:d7:ba:7c:87:f7:b5:8c:e1:e5:f9:6b:d8:f0:bc:
        04:39:a5:f4:cc:79:a5:c4:91:a8:94:65:65:c2:88:e3:b9:e3:
        91:4f:2e:92:14:56:98:dc:90:a3:74:8a:8d:ee:1e:fd:f4:1e:
        e8:58:0c:3d:16:37:fd:35:e5:3e:c9:b8:b6:af:9d:fb:cc:c1:
        43:69:66:42:ec:a2:55:3f:24:88:5d:c4:e5:aa:82:d4:4c:a0:
        b1:e2:36:7b:da:08:5a:00:46:f5:12:19:d7:78:93:8e:7f:c3:
        57:74:01:39:a7:03:55:86:18:f4:8f:fa:47:a8:e7:bb:ce:b7:
        73:84:ec:5e:96:42:b8:b7:2a:25:a7:a0:9c:cb:e6:0b:b7:21:
        0a:02:41:db:99:c0:01:c3:52:d3:a5:06:a3:2b:ee:3f:59:d7:
        27:ea:42:29:15:66:4d:44:34:f9:e0:fb:8d:81:55:1e:e6:dc:
        4d:bc:df:7d:fd:9c:59:3e:ae:72:f9:5e:58:f8:c3:35:8f:21:
        15:e9:b6:54:ad:12:ed:29:16:75:e8:82:cf:a8:fb:3c:62:8d:
        2d:a0:5d:f8:6d:ba:ad:32:8b:98:3c:1d:bc:1e:2f:82:45:58:
        b7:34:e3:d1:8a:b2:f9:45:4f:e7:42:ef:49:f6:e1:a8:3b:aa:
        5e:b3:38:f8:4e:ad:5d:80:5a:6c:0b:8a:8f:45:34:27:e9:70:
        16:45:1f:bc:84:0a:fa:00
```

As we can see from the above output, the version of Bob's certificate is:

```
 Version: 3 (0x2)
```

assigned serial number is :

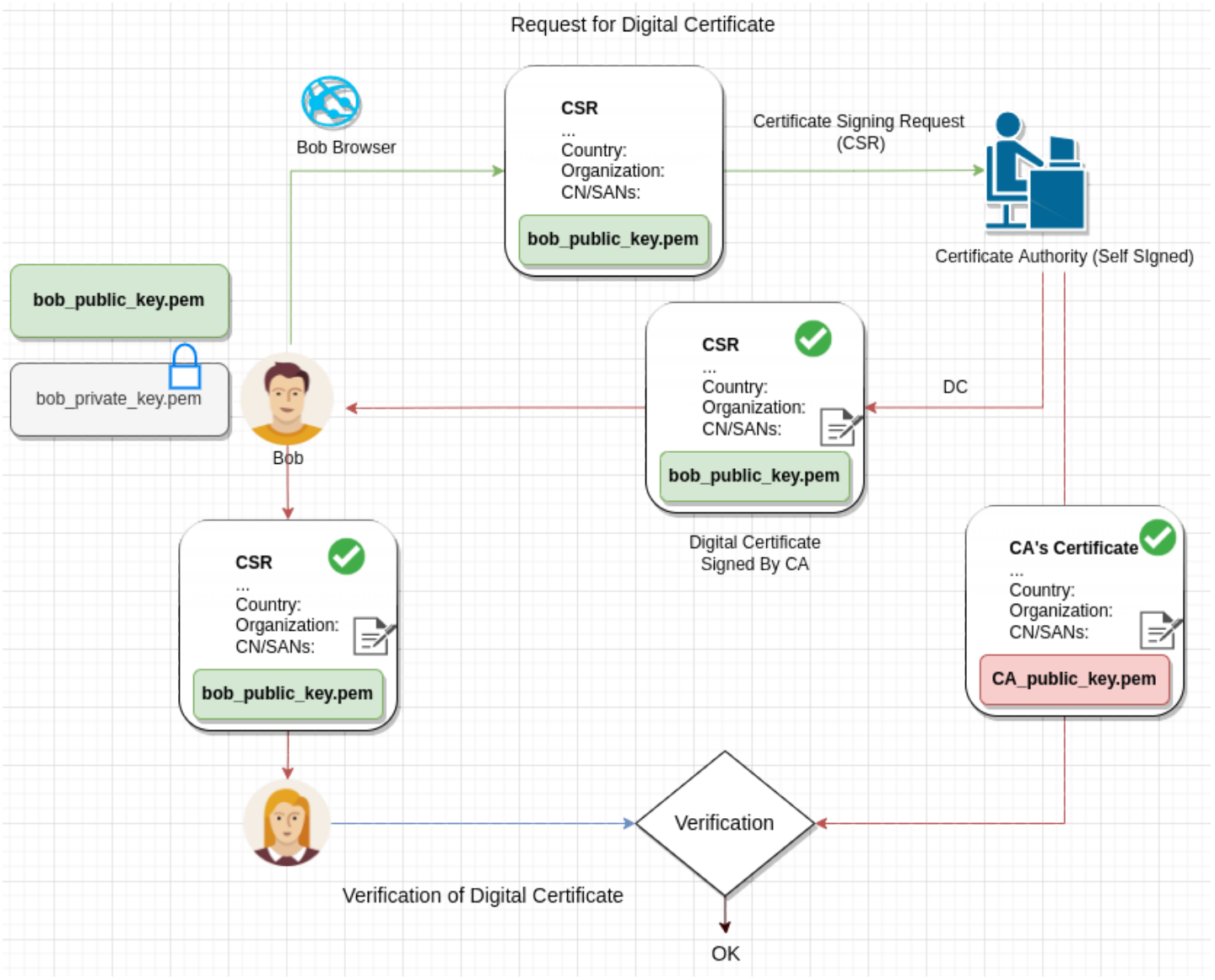
```
6d:23:87:12:5a:29:1d:63:f0:07:ae:48:ed:eb:4d:51:d0:be:8b:6c
```

and key usages and the constraints associated are:

```
        X509v3 extensions:
            X509v3 Extended Key Usage:
                TLS Web Server Authentication, TLS Web Client Authentication, Code Signing, E-mail Protection
            X509v3 Basic Constraints:
                CA:FALSE
            X509v3 Key Usage:
                Digital Signature, Non Repudiation, Key Encipherment
```

The entire process of requesting for a certificate and its verification is shown in the figure below:

The complete workflow for PART B can be found here: PART B


Request for Digital Certificate
Bob Browser
CSR
...
Country:
Organization:
CN/SANs:
bob_public_key.pem
Certificate Signing Request
(CSR)
Certificate Authority (Self SIgned)
bob_public_key.pem
bob_private_key.pem
Bob
CSR
...
Country:
Organization:
CN/SANs:
bob_public_key.pem
DC
Digital Certificate
Signed By CA
CSR
...
Country:
Organization:
CN/SANs:
bob_public_key.pem
CA's Certificate
...
Country:
Organization:
CN/SANs:
CA_public_key.pem
Verification
Verification of Digital Certificate
OK

PLAGIARISM STATEMENT

I certify that this assignment/report is my own work, based on my personal study and/or research and that I have acknowledged all material and sources used in its preparation, whether they be books, articles, reports, lecture notes, and any other kind of document, electronic or personal communication. I also certify that this assignment/report has not previously been submitted for assessment in any other course, except where specific permission has been granted from all course instructors involved, or at any other time in this course, and that I have not copied in part or whole or otherwise plagiarised the work of other students and/or persons. I pledge to uphold the principles of honesty and responsibility at CSE@IITH. In addition, I understand my responsibility to report honor violations by other students if I become aware of them.

Name: Kamal Shrestha, Srivathsa L. Rao
Date: Feb 6, 2022
Signature: K.S., S. L. R.