Experiment 4: Asymmetric (Public) Key

Name: Deep Nayak **UID**: 2019130045 TE COMPS

AIM : The aim of this lab is to provide a practical introduction to public key encryption, and with a focus on RSA and Elliptic Curve methods. This includes the creation of key pairs and in the signing process. As a part of this objective first you perform section c which is given below.

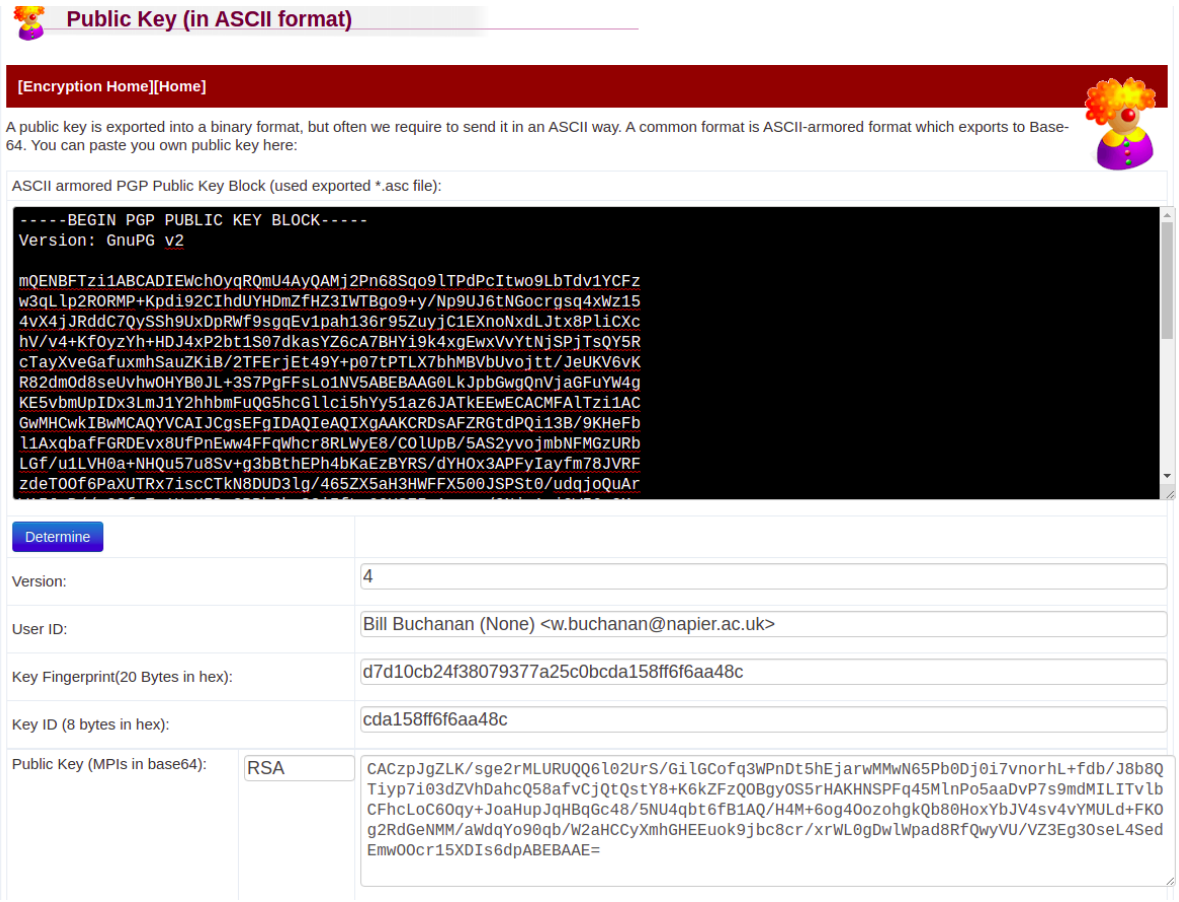
& **Web link (Weekly activities):** Unit 4: Public Key

& **Video demo:** Lab 4: Asymmetric Encryption

PROBLEM STATEMENT

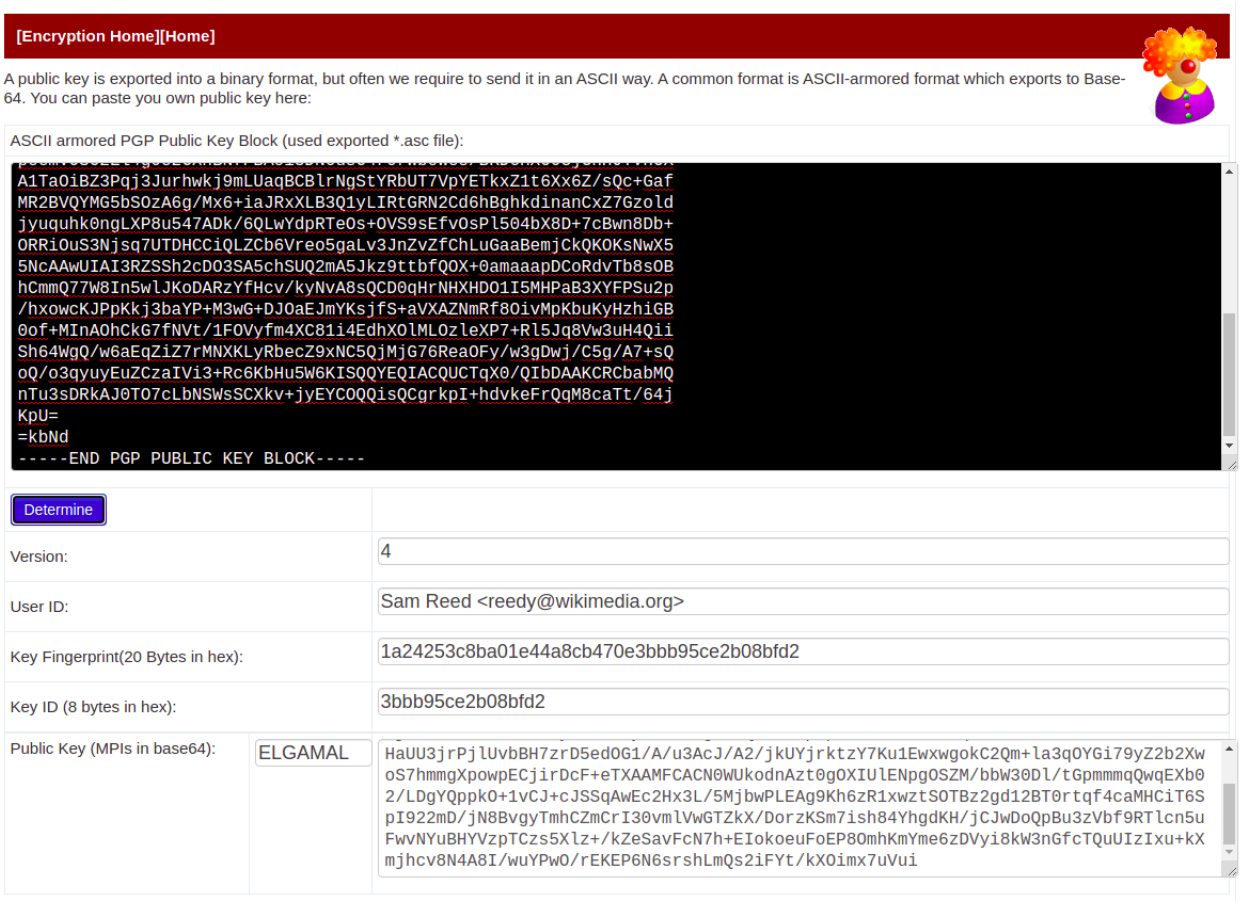
A) RSA Encryption

A.1) OUTPUT :



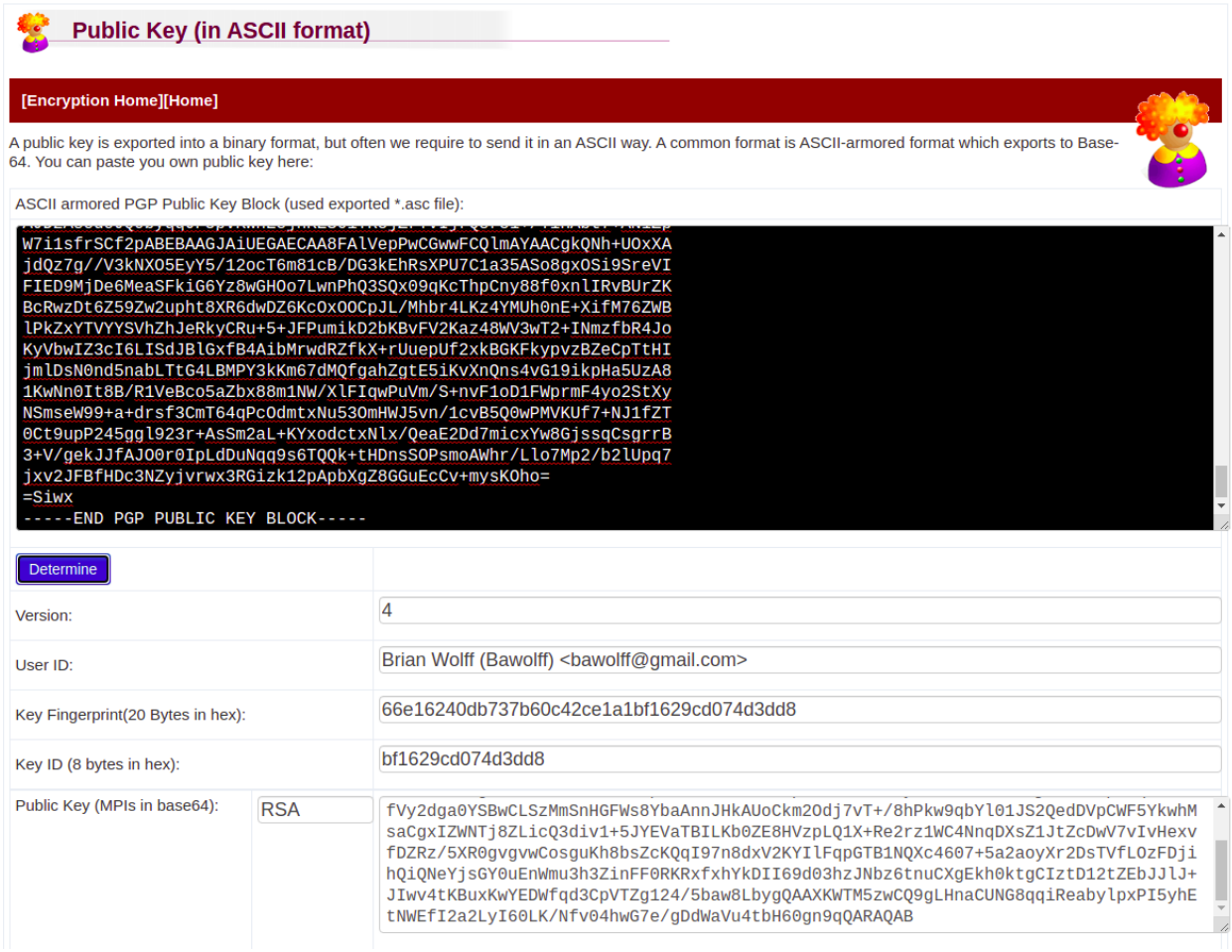
Public Key of three famous people :

- **Sam Reed**



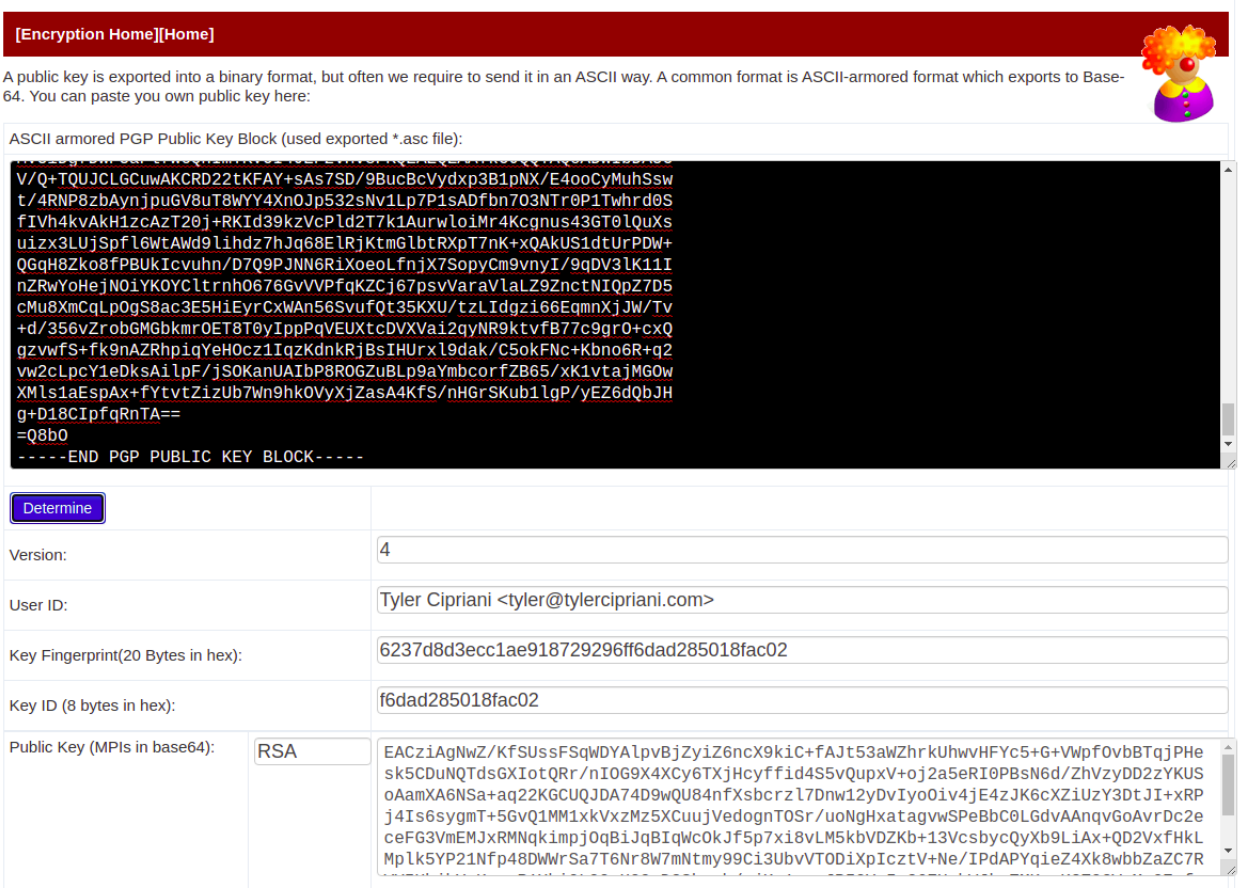
Key Encryption Algorithm : ELGAMAL

- **BRIAN WOLFF**



Key Encryption Algorithm : RSA

- **TYLER CIPRIANI**



Key Encryption Algorithm : RSA

What is ASCII armored Message ?

ASCII armour is a binary-to-text conversion tool. It is the feature of a type of encryption known as pretty good privacy (PGP). Encrypted messages are encased in ASCII armour so that they can be transmitted in a normal messaging medium like email.The reasoning behind ASCII armor for PGP is that the original PGP format is binary, which is not considered very readable by some of the most common messaging formats. Making the file into American Standard Code for Information Interchange (ASCII) format converts the binary to a printable character representation. Handling file volume can be accomplished through compressing the file.

B) Openssl RSA

No	Description	Result
	First we need to generate a key pair with: openssl genrsa -out private.pem 1024 This file contains both the public and the private key	What is the type of public key method used: **RSA key generation algorithm** How long is the default key: **1024 bits** How long did it take to generate a 1,024 bit key? : **Less than a second, immediately**
B.2	Use following command to view the output file: cat private.pem	What can be observed at the start and end of the file: **----BEGIN RSA PRIVATE KEY----** **And** **-END RSA PRIVATE KEY-** Hyphens and two words BEGIN and END are always present at the start and end of the file.
B.3	Next we view the RSA key pair: openssl rsa -in private.pem -text	Which are the attributes of the key shown: ● Modulus ● Public Exponent ● Private Exponent

		• Prime1 • Prime2 • Exponent1 • Exponent2 • Coefficient Which number format is used to display the information on the attributes: **Hexadecimal**
B.4	Let's now secure the encrypted key with 3-DES: openssl rsa -in private.pem -des3 -out key3des.pem	Why should you have a password on the usage of your private key? **The use of a passcode on the private key adds another degree of protection to the key. If the file is stolen and it does not have a passcode, anyone who has access to it can log into anything we have access to. The passcode I used is deepcss@123**
B.5	Next we will export the public key: openssl rsa -in private.pem -out public.pem -outform PEM -pubout	What does the header and footer of the file identify? **It represents that the key stored in this file is the public key.**
B.6	Now create a file named "myfile.txt" and put a message into it. Next encrypt it with your public key: openssl rsautl -encrypt -inkey public.pem -pubin -in myfile.txt -out file.bin	
B.7	And then decrypt with your private key: openssl rsautl -decrypt -inkey private.pem -in file.bin -out decrypted.txt	What are the contents of decrypted.txt ? Can be seen in the screenshot below.

B1

```
deep@deep-HP-Pavilion-Gaming-Laptop-15-dk0xxx: ~/spit/css/CSS-Lab-Deep-Nayak/Experiment 4
deep@deep-HP-Pavilion-Gaming-Laptop-15-dk0xxx:~/spit/css/CSS-Lab-Deep-Nayak/Experiment 4$ openssl genrsa -out private.pem 1024
Generating RSA private key, 1024 bit long modulus (2 primes)
...........+++++
..............+++++
e is 65537 (0x010001)
```

B2

```
deep@deep-HP-Pavilion-Gaming-Laptop-15-dk0xxx:~/spit/css/CSS-Lab-Deep-Nayak/Experiment 4$ cat private.pem
-----BEGIN RSA PRIVATE KEY-----
MIICXgIBAAKBgQDUKRLrU04Dtzxdzc3WeXefhgqXurWUlp53aZk/18+//469seR1
FjfLSprh/txk9tmmWaIojYWlMQgPDW5w7nInWCD9Q5X4sGihd7vDtd33tyIuyHld
OfUHT0go94SYyFLx1iw0EVE9O/v4TeusQwVrLZLV0dWiChq+Up8UugfaXwIDAQAB
AoGBALgAs4bjJgs09ScJ4ha8HFJ/R1qWtPscsm0waZgkTOrg8uR0j4Y48cTffHq6
tlzedp++p1PygdFmN+4OIRXSx67gF6TcKhyh72C6CRBKdB02jWTf0JSUZE+qJkEd
COKSzIgV2cIK5OF6maWPHNOzPzIX5wucvDpVYnyKgdqSDLbZAkEA/dW+RpPPx4my
U0FeyeZVQ0cnPRxFzdF3Pm7jwKh2n8TqCuc615hat5xd4eOPL9DXtk3NC9Xfe6ej
qObyucQztQJBANX4VULejEjavhRAVjtm0zkLdojGvSZeZKHUytFy77qnSNxhcYg0
MTnXbhmGz36A4WNm8TNkR2Ex2xOD6L9MSkMCQQDz/0NhKnKfVxBQkVThdQReoIXq
a0P6PTCHliZdqlIk8C8dffoFBhGlz6w7F/gquXCxcAX7K4Igphten2mRAjXFAkAJ
l5pq5wMu+O9whPnYq+FoO0VTGGW+ZJjqUFnPTfHFUl6H0q5+Rqi2SRwcdNi0eR6h
kCnb/fDdQJjrpQw4Fzl1AkEAgnWMklSJidE/r+ZeDwYrmx024qBv30JnaEXzAmrI
wdCJD1nDpwm0FTR2c6CSzRgb78hBl3qVVVlNdw4SaQU8Sg==
-----END RSA PRIVATE KEY-----
```

B3

```
deep@deep-HP-Pavilion-Gaming-Laptop-15-dk0xxx:~/spit/css/CSS-Lab-Deep-Nayak/Experiment 4$ openssl rsa -in private.pem -text
RSA Private-Key: (1024 bit, 2 primes)
modulus:
    00:d4:29:12:eb:53:4e:03:b7:3c:5d:cd:cd:d6:79:
    77:9f:86:0a:97:ba:b5:94:96:9e:77:69:99:3f:d7:
    cf:bf:ff:8e:bd:b1:e4:75:16:37:cb:4a:9a:e1:fe:
    dc:64:f6:d9:a6:59:a2:28:8d:85:a5:31:08:0f:0d:
    6e:70:ee:72:27:58:20:fd:43:95:f8:b0:68:a1:77:
    bb:c3:b5:dd:f7:b7:22:2e:c8:79:5d:39:f5:07:4f:
    48:28:f7:84:98:c8:52:f1:d6:2c:34:11:51:3d:3b:
    fb:f8:4d:eb:ac:43:05:6b:2d:92:d5:d1:d5:a2:0a:
    1a:be:52:9f:14:ba:07:da:5f
publicExponent: 65537 (0x10001)
privateExponent:
    00:b8:00:b3:86:e3:26:0b:34:f5:27:09:e2:16:bc:
    1c:52:7f:47:5a:96:b4:fb:1c:b2:6d:30:69:98:24:
    4c:ea:e0:f2:e4:74:8f:86:38:f1:c4:df:7c:7a:ba:
    b6:5c:de:76:9f:be:a7:53:f2:81:d1:66:37:ee:0e:
    21:15:d2:c7:ae:e0:17:a4:dc:2a:1c:a1:ef:60:ba:
    09:10:4a:74:1d:36:8d:64:df:d0:94:94:64:4f:aa:
    26:41:1d:08:e2:92:cc:88:15:d9:c2:0a:e4:e1:7a:
    99:a5:8f:1c:d3:b3:3f:32:17:e7:0b:9c:bc:3a:55:
    62:7c:8a:81:da:92:0c:b6:d9
prime1:
    00:fd:d5:be:46:93:cf:c7:89:b2:53:41:5e:c9:e6:
    55:43:47:27:3d:1c:45:cd:d1:77:3e:6e:e3:c0:a8:
    76:9f:c4:ea:0a:e7:3a:d7:98:5a:b7:9c:5d:e1:e3:
    8f:2f:d0:d7:b6:4d:cd:0b:d5:df:7b:a7:a3:a8:e6:
    f2:b9:c4:33:b5
prime2:
    00:d5:f8:55:42:de:8c:48:da:be:14:40:56:3b:66:
    d3:39:0b:76:88:c6:bd:26:5e:64:a1:d4:ca:d1:72:
    ef:ba:a7:48:dc:61:71:88:34:31:39:d7:6e:19:86:
    cf:7e:80:e1:63:66:f1:33:64:47:61:31:db:13:83:
    e8:bf:4c:4a:43
```

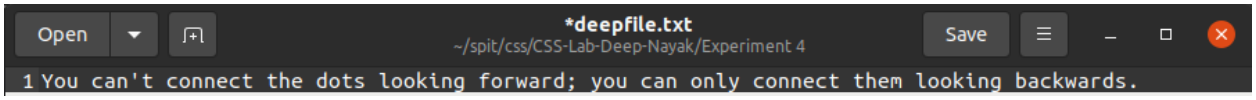
B4

```
deep@deep-HP-Pavilion-Gaming-Laptop-15-dk0xxx:~/spit/css/CSS-Lab-Deep-Nayak/Experiment 4$ openssl rsa -in
 private.pem -des3 -out key3des.pem
writing RSA key
Enter PEM pass phrase:
Verifying - Enter PEM pass phrase:
```

B5

```
deep@deep-HP-Pavilion-Gaming-Laptop-15-dk0xxx:~/spit/css/CSS-Lab-Deep-Nayak/Experiment 4$ openssl rsa -in
 private.pem -out public.pem -outform PEM -pubout
writing RSA key
deep@deep-HP-Pavilion-Gaming-Laptop-15-dk0xxx:~/spit/css/CSS-Lab-Deep-Nayak/Experiment 4$ cat public.pem
-----BEGIN PUBLIC KEY-----
MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQDUKRLrU04Dtzxdzc3WeXefhgqX
urWUlp53aZk/18+//469seR1FjfLSprh/txk9tmmWaIojYWlMQgPDW5w7nInWCD9
Q5X4sGihd7vDtd33tyIuyHldOfUHT0go94SYyFLx1iw0EVE9O/v4TeusQwVrLZLV
0dWiChq+Up8UugfaXwIDAQAB
-----END PUBLIC KEY-----
```

B6

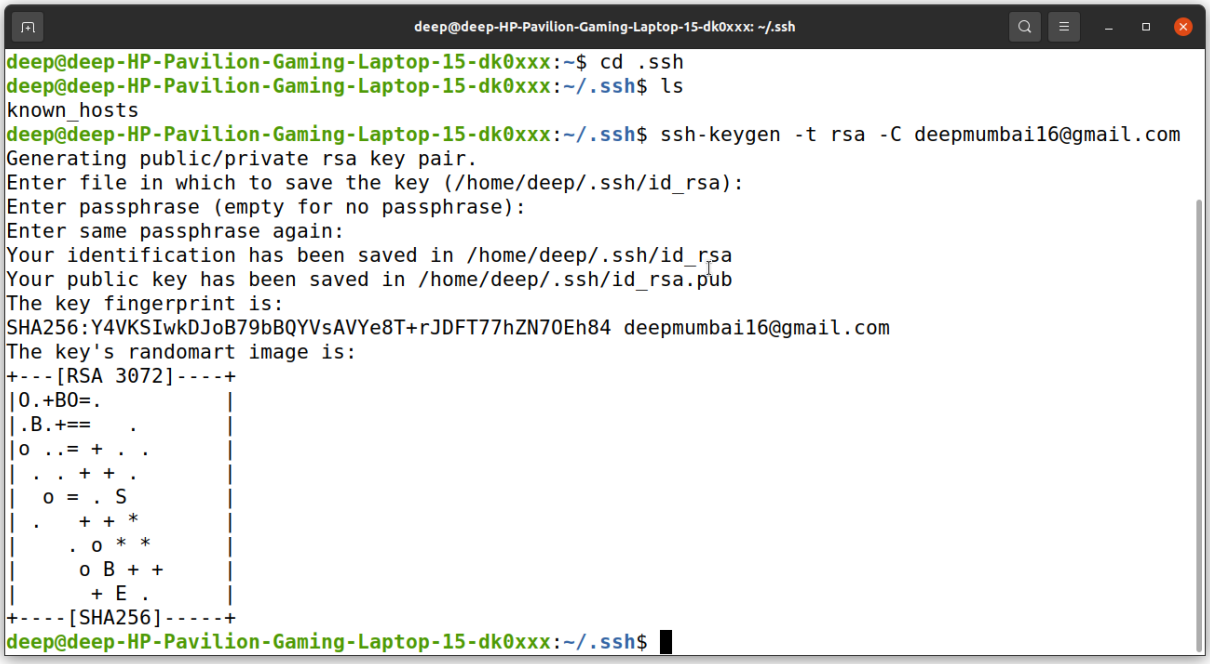


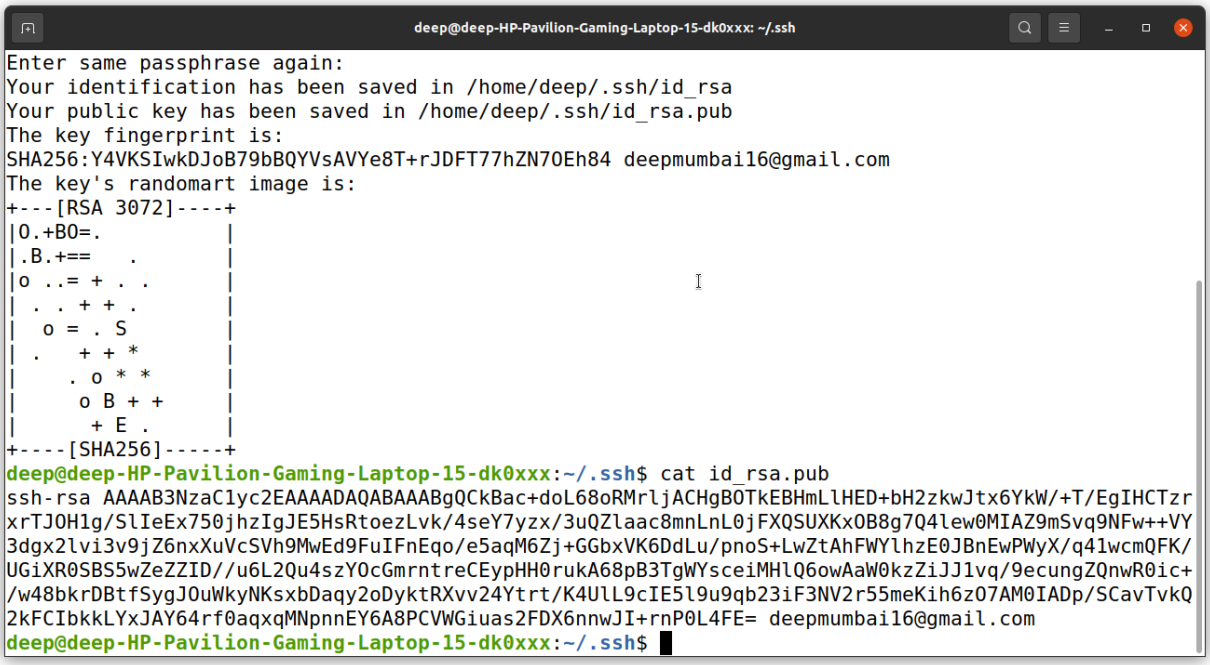

```
deep@deep-HP-Pavilion-Gaming-Laptop-15-dk0xxx:~/spit/css/CSS-Lab-Deep-Nayak/Experiment 4$ touch deepfile.
txt
deep@deep-HP-Pavilion-Gaming-Laptop-15-dk0xxx:~/spit/css/CSS-Lab-Deep-Nayak/Experiment 4$ gedit deepfile.
txt
deep@deep-HP-Pavilion-Gaming-Laptop-15-dk0xxx:~/spit/css/CSS-Lab-Deep-Nayak/Experiment 4$ openssl rsautl
-encrypt -inkey public.pem -pubin -in deepfile.txt -out file.bin
deep@deep-HP-Pavilion-Gaming-Laptop-15-dk0xxx:~/spit/css/CSS-Lab-Deep-Nayak/Experiment 4$ cat file.bin
;g@h@7@D@@
          @@@DPf@@x<@@h\y@9@_@@'N@@}@@@@4v@@\F@D@@\-D{@@@g0Q@@@ɾ!@        U@-]@ag@@@@@Sw\Ѝ@P@@@@<@@@@@+@@P
X@5@@ O*@deep@deep-HP-Pavilion-Gaming-Laptop-15-dk0xxx:~/spit/css/CSS-Lab-Deep-Nayak/Experiment 4$
```

B7

```
deep@deep-HP-Pavilion-Gaming-Laptop-15-dk0xxx:~/spit/css/CSS-Lab-Deep-Nayak/Experiment 4$ openssl rsautl -d
ecrypt -inkey private.pem -in file.bin -out decrypted.txt
deep@deep-HP-Pavilion-Gaming-Laptop-15-dk0xxx:~/spit/css/CSS-Lab-Deep-Nayak/Experiment 4$ cat decrypted.txt
You can't connect the dots looking forward; you can only connect them looking backwards.
deep@deep-HP-Pavilion-Gaming-Laptop-15-dk0xxx:~/spit/css/CSS-Lab-Deep-Nayak/Experiment 4$
```

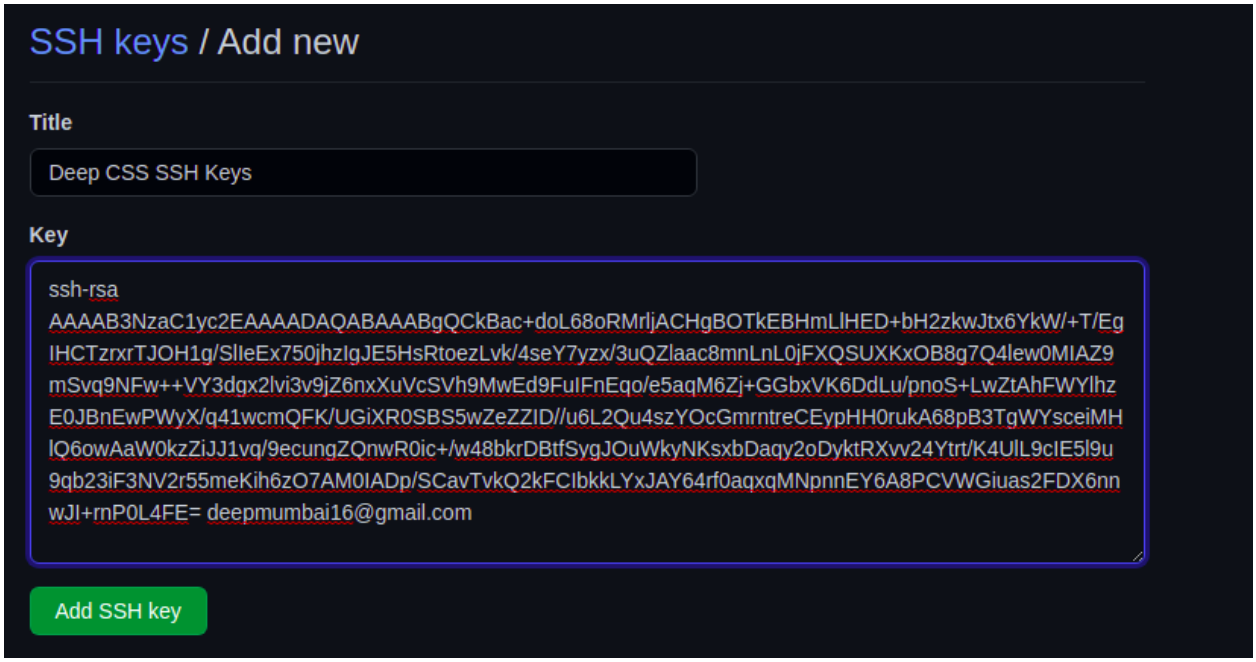
On your VM, go into the ~/.ssh folder. Now generate your SSH keys:






```
deep@deep-HP-Pavilion-Gaming-Laptop-15-dk0xxx:~/.ssh$ ssh-add ~/.ssh/id_rsa
Enter passphrase for /home/deep/.ssh/id_rsa:
Identity added: /home/deep/.ssh/id_rsa (deepmumbai16@gmail.com)
deep@deep-HP-Pavilion-Gaming-Laptop-15-dk0xxx:~/.ssh$
```

Uploading Public Key to Github Account



```
deep@deep-HP-Pavilion-Gaming-Laptop-15-dk0xxx:~/test$ git clone git@github.com:deepnayak/CSS-Lab-D
eep-Nayak.git
Cloning into 'CSS-Lab-Deep-Nayak'...
The authenticity of host 'github.com (13.234.210.38)' can't be established.
ECDSA key fingerprint is SHA256:p2QAMXNIC1TJYWeIOttrVc98/R1BUFWu3/LiyKgUfQM.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added 'github.com,13.234.210.38' (ECDSA) to the list of known hosts.
remote: Enumerating objects: 65, done.
remote: Counting objects: 100% (65/65), done.
remote: Compressing objects: 100% (36/36), done.
remote: Total 65 (delta 13), reused 62 (delta 10), pack-reused 0
Receiving objects: 100% (65/65), 8.12 MiB | 5.00 MiB/s, done.
Resolving deltas: 100% (13/13), done.
deep@deep-HP-Pavilion-Gaming-Laptop-15-dk0xxx:~/test$
```

I was able to successfully clone the repository using the newly generated SSH keys.

C) Openssl ECC

No	Description	Result
C.1	First we need to generate a private key with: openssl ecparam -name secp256k1 -genkey -out priv.pem The file will only contain the private key (and should have 256 bits).Now use “cat priv.pem” to view your key.	Can you view your key? **Yes**
C.2	We can view the details of the ECC parameters used with: openssl ecparam -in priv.pem -text -param_enc explicit -noout	Outline these values: Prime (last two bytes): **fc:2f** A: **0** B: **7** Generator (last two bytes): **d4:b8** Order (last two bytes): **41:41**
C.3	Now generate your public key based on your private key with: openssl ec -in priv.pem -text -noout	How many bits and bytes does your private key have: **256 bits or 32 Bytes** How many bits and bytes does your public key have (Note the 04 is not part of the elliptic curve point): **64 Bytes and 512 bits** What is the ECC method that you have used? **secp256k1**

C1

```
deep@deep-HP-Pavilion-Gaming-Laptop-15-dk0xxx:~/spit/css/CSS-Lab-Deep-Nayak/Experiment 4$ openssl
ecparam -name secp256k1 -genkey -out priv.pem
deep@deep-HP-Pavilion-Gaming-Laptop-15-dk0xxx:~/spit/css/CSS-Lab-Deep-Nayak/Experiment 4$ cat priv
.pem
-----BEGIN EC PARAMETERS-----
BgUrgQQACg==
-----END EC PARAMETERS-----
-----BEGIN EC PRIVATE KEY-----
MHQCAQEEIAxpIilr08CkXmzW7Ideo8ZnMBKxu3VYAWXifM7mxD0VoAcGBSuBBAAK
oUQDQgAEb9nLx2IzajeTxH5bTpIB0dIlSIts195y0EcuYW5+pMEjbPiIky2fwD4E
8JYLHZwtMrXpXcStHAwexS1C/arFLA==
-----END EC PRIVATE KEY-----
```

C2

```
deep@deep-HP-Pavilion-Gaming-Laptop-15-dk0xxx:~/spit/css/CSS-Lab-Deep-Nayak/Experiment 4$ openssl
ecparam -in priv.pem -text -param_enc explicit -noout
Field Type: prime-field
Prime:
    00:ff:ff:ff:ff:ff:ff:ff:ff:ff:ff:ff:ff:ff:ff:
    ff:ff:ff:ff:ff:ff:ff:ff:ff:ff:ff:ff:fe:ff:
    ff:fc:2f
A:    0
B:    7 (0x7)
Generator (uncompressed):
    04:79:be:66:7e:f9:dc:bb:ac:55:a0:62:95:ce:87:
    0b:07:02:9b:fc:db:2d:ce:28:d9:59:f2:81:5b:16:
    f8:17:98:48:3a:da:77:26:a3:c4:65:5d:a4:fb:fc:
    0e:11:08:a8:fd:17:b4:48:a6:85:54:19:9c:47:d0:
    8f:fb:10:d4:b8
Order:
    00:ff:ff:ff:ff:ff:ff:ff:ff:ff:ff:ff:ff:ff:ff:
    ff:fe:ba:ae:dc:e6:af:48:a0:3b:bf:d2:5e:8c:d0:
    36:41:41
Cofactor:  1 (0x1)
deep@deep-HP-Pavilion-Gaming-Laptop-15-dk0xxx:~/spit/css/CSS-Lab-Deep-Nayak/Experiment 4$
```

C3

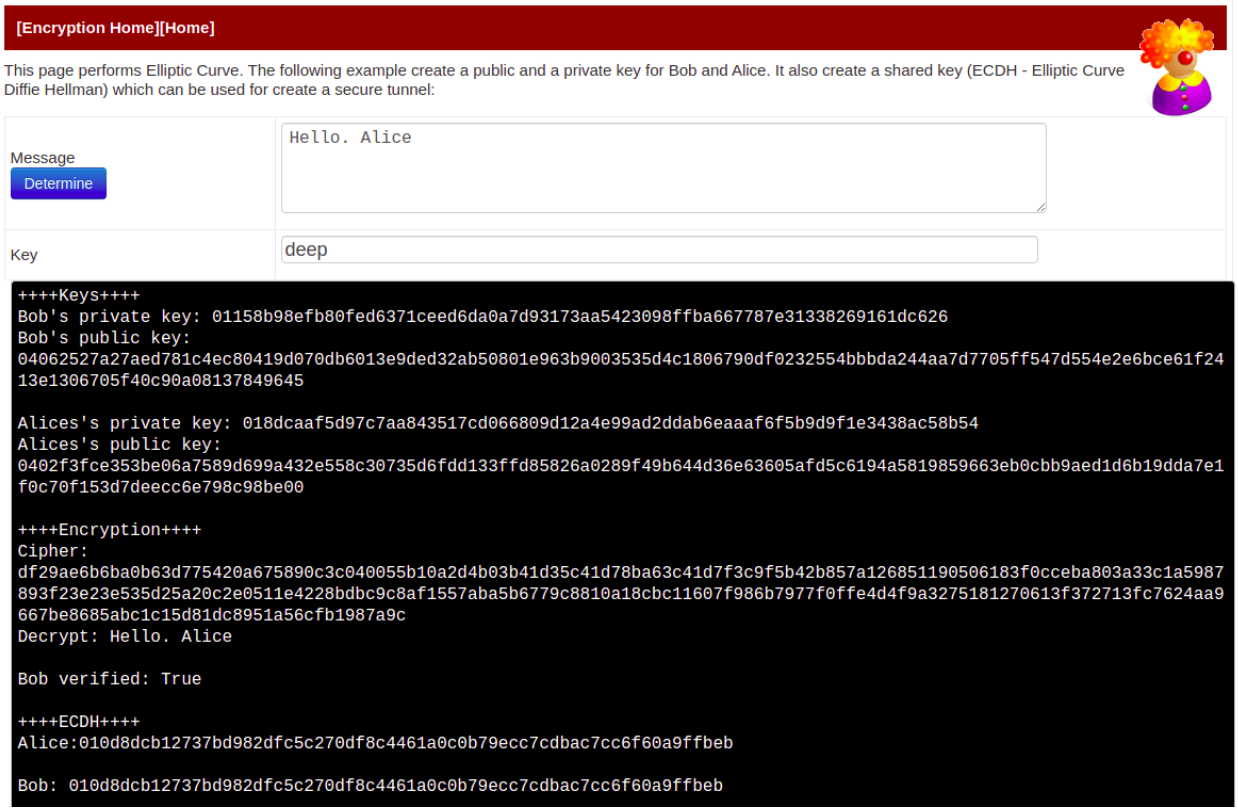
```
deep@deep-HP-Pavilion-Gaming-Laptop-15-dk0xxx:~/spit/css/CSS-Lab-Deep-Nayak/Experiment 4$ openssl
ec -in priv.pem -text -noout
read EC key
Private-Key: (256 bit)
priv:
    0c:69:22:29:6b:d3:c0:a4:5e:6c:d6:ec:87:5e:a3:
    c6:67:30:12:b1:bb:75:58:01:65:e2:7c:ce:e6:c4:
    3d:15
pub:
    04:6f:d9:cb:c7:62:33:6a:37:93:c4:7e:5b:4e:92:
    01:d1:d2:25:48:8b:6c:d7:de:72:d0:47:2e:61:6e:
    7e:a4:c1:23:6c:f8:88:93:2d:9f:c0:3e:04:f0:96:
    0b:1d:9c:2d:32:b5:e9:5d:c4:ad:1c:0c:1e:c5:2d:
    42:fd:aa:c5:2c
ASN1 OID: secp256k1
```

Elliptic Curve Encryption

D.1) OUTPUT :

For a message of “Hello. Alice”, what is the ciphertext sent (just include the first four characters):

The first four characters are df29

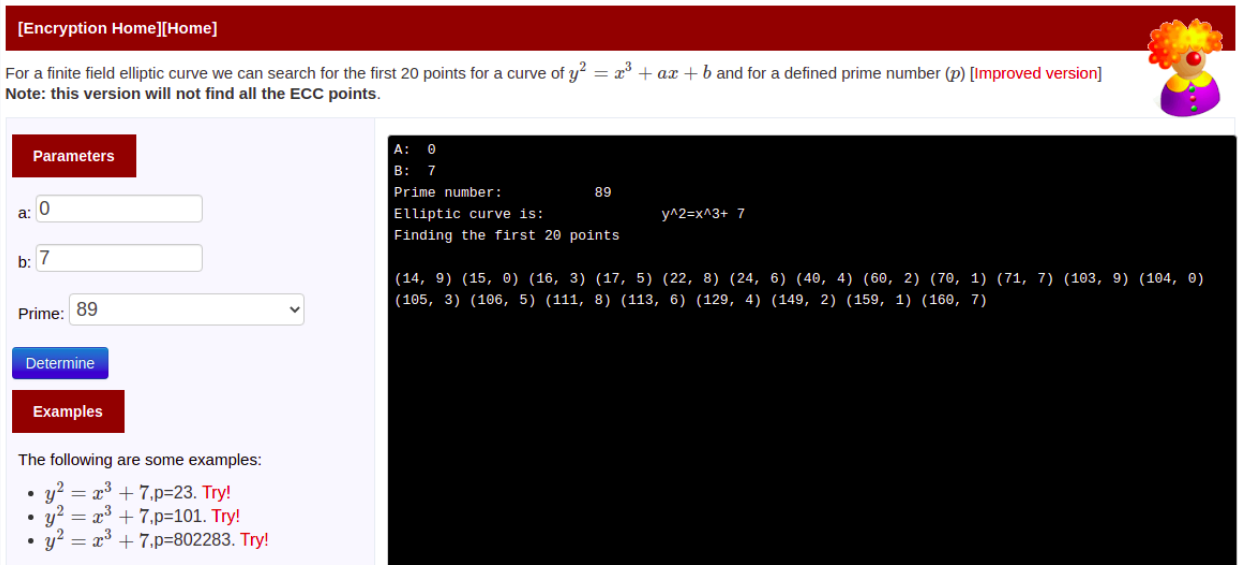


How is the signature used in this example?

Because Bob is encrypting the message and delivering it to Alice, she must first verify that the message originated from Bob. To do so, she uses Bob's signature together with his public key to validate his authenticity.

D.2) OUTPUT :

Let's say we create an elliptic curve with y2 = x3 + 7, and with a prime number of 89, generate the first five (x,y) points for the finite field elliptic curve, find the first five points



First Five Points are : (14,9) , (15,0) , (16,3) , (17,5) , (22,8)

D.3) OUTPUT :

Elliptic curve methods are often used to sign messages, and where Bob will sign a message with his private key, and where Alice can prove that he has signed it by using his public key. With ECC, we can use ECDSA, and which was used in the first version of Bitcoin

NIST 192p:

ecc.py 1, U

ecc.py > ...

```python
from ecdsa import SigningKey,NIST192p,NIST224p,NIST256p,NIST384p,NIST521p,SECP256k1
import base64
import sys

msg=b"Bob"
type = 1
cur=NIST192p

sk = SigningKey.generate(curve=cur)
vk = sk.get_verifying_key()
signature = sk.sign(msg)

print("Message:\t",msg)
print("Type:\t\t",cur.name)
print("==========================")
print("Signature:\t",base64.b64encode(signature))
print("==========================")
print("Signatures match:\t",vk.verify(signature, msg))

```

PROBLEMS 1 OUTPUT TERMINAL DEBUG CONSOLE

```
deep@deep-HP-Pavilion-Gaming-Laptop-15-dk0xxx:~/spit/css/CSS-Lab-Deep-Nayak/Experiment 4$ /bin/python3
ment 4/ecc.py"
Message:        b'Bob'
Type:           NIST192p
==========================
Signature:      b'PwIGk6TY8n8sWKZJaHtdADkkqER9vnhIw5ztOsuXymCT8loxtFCzfSMb9yurXwis'
==========================
Signatures match:       True
deep@deep-HP-Pavilion-Gaming-Laptop-15-dk0xxx:~/spit/css/CSS-Lab-Deep-Nayak/Experiment 4$
```

NIST 521p:

ecc.py 1, U

ecc.py > ...

```python
from ecdsa import SigningKey,NIST192p,NIST224p,NIST256p,NIST384p,NIST521p,SECP256k1
import base64
import sys

msg=b"Bob"
type = 1
cur=NIST521p

sk = SigningKey.generate(curve=cur)
vk = sk.get_verifying_key()
signature = sk.sign(msg)

print("Message:\t",msg)
print("Type:\t\t",cur.name)
print("========================")
print("Signature:\t",base64.b64encode(signature))
print("========================")
print("Signatures match:\t",vk.verify(signature, msg))

```

PROBLEMS 1 OUTPUT TERMINAL DEBUG CONSOLE

```
deep@deep-HP-Pavilion-Gaming-Laptop-15-dk0xxx:~/spit/css/CSS-Lab-Deep-Nayak/Experiment 4$ /bin/python3 "/home/deep/spit/css/CSS-Lab-Deep-Nayak/Experi
ment 4/ecc.py"
Message:        b'Bob'
Type:           NIST521p
========================
Signature:      b'ANu89odN2nq9vb7ffiZBPcNGT34sLkZRJcHF+pEq00XjwtQV/pK1Ytxcr1g0LSu5C1jzSB2xQiGBtJBY6dJRYU1kAQcftL2s3Qcw2cH7RtD0uT3hv2BAx0jVgGmb0ba++T
ydEC6Fz1pUY9I2bnI3iddoh6YvXPioJyAfmO2hRJgDFK9+'
========================
Signatures match:       True
deep@deep-HP-Pavilion-Gaming-Laptop-15-dk0xxx:~/spit/css/CSS-Lab-Deep-Nayak/Experiment 4$
```

SECP256k1:

ecc.py 1, U

ecc.py > ...

```
1  from ecdsa import SigningKey,NIST192p,NIST224p,NIST256p,NIST384p,NIST521p,SECP256k1
2  import base64
3  import sys
4
5  msg=b"Bob"
6  type = 1
7  cur=SECP256k1
8
9  sk = SigningKey.generate(curve=cur)
10 vk = sk.get_verifying_key()
11 signature = sk.sign(msg)
12
13 print("Message:\t",msg)
14 print("Type:\t\t",cur.name)
15 print("=========================")
16 print("Signature:\t",base64.b64encode(signature))
17 print("=========================")
18 print("Signatures match:\t",vk.verify(signature, msg))
19
```

PROBLEMS 1 OUTPUT TERMINAL DEBUG CONSOLE

```
deep@deep-HP-Pavilion-Gaming-Laptop-15-dk0xxx:~/spit/css/CSS-Lab-Deep-Nayak/Experiment 4$ /bin/python3 "/home/deep
ment 4/ecc.py"
Message:        b'Bob'
Type:           SECP256k1
=========================
Signature:      b'3ozE5xtRVp5JFKN0FWAfsXAsUx8e0Vse0Nhxr9W39AOVt5HyxajpD77T3tyufTNWmvH/KcOYFkIDzDNPx1NYtw=='
=========================
Signatures match:       True
deep@deep-HP-Pavilion-Gaming-Laptop-15-dk0xxx:~/spit/css/CSS-Lab-Deep-Nayak/Experiment 4$
```

Application Areas of SECP256k1 :

Secp256k1 is used for all of Bitcoin's public-key cryptography. At least one secp256k1 signature and at least one secp256k1 public key or public key hash are present in every reasonable transaction (address). The only technological issue that might bring Bitcoin down is a full failure of ECDSA/secp256k1. However, this is extremely unlikely. Other programmes are likely to use Secp256k1 (though Bitcoin is certainly the largest). It's a Certicom Research standard curve that's included in OpenSSL and other crypto libraries since 2000. The RLPx transport protocol in Ethereum makes use of an elliptic curve integrated encryption technique based on the SECP256k1 curve. These applications are appropriate for our attack scenario, in which the attacker chooses the starting point. We employ SECP256k1 as the concrete curve parameters to execute the aforementioned attacks in order to increase their application.

Difference between Hashes :

The difference between the hash signatures that I noticed was their size; while NIST192p employs a 192-bit elliptic curve, the associated hash signature is 64-bit, and when the elliptic curve size grows, as it does in NIST512 and SECP256, the size increases as well.

RSA

E.1) OUTPUT :

Picking the prime numbers
P = 29
Q = 31

N = 899
PHI = 840
e = 11
d = 99

Now if M = 4 then cipher is calculated as
C = 4^e (mod N) = 469

After decrypting the above ciphertext
411 = C^d (mod N) = 469^99 (mod 22)

rsa.py U

rsa.py > ...

```
1   p=29
2   q=31
3   N=p*q
4
5   PHI=(p-1)*(q-1)
6   e=11
7   for d in range(1,100):
8       if ((e*d % PHI)==1): break
9   print (e,N)
10
11  print (d,N)
12  M=4
13  cipher = M**e % N
14
15  print (cipher)
16  message = cipher**d % N
17  print (message)
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

```
deep@deep-HP-Pavilion-Gaming-Laptop-15-dk0xxx:~/spit/css/CSS-Lab-Deep-Nayak/Experiment 4$ /bin/python3
ment 4/rsa.py"
11 899
99 899
469
411
deep@deep-HP-Pavilion-Gaming-Laptop-15-dk0xxx:~/spit/css/CSS-Lab-Deep-Nayak/Experiment 4$
```

3 more examples :

rsa.py U

```python
p=17
q=23
N=p*q

PHI=(p-1)*(q-1)
e=3
for d in range(1,100):
    if ((e*d % PHI)==1): break
print (e,N)

print (d,N)
M=4
cipher = M**e % N

print (cipher)
message = cipher**d % N
print (message)
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

```
deep@deep-HP-Pavilion-Gaming-Laptop-15-dk0xxx:~/spit/css/CSS-Lab-Deep-Nayak/Experiment 4$ /bin/python3
ment 4/rsa.py"
3 391
99 391
64
208
deep@deep-HP-Pavilion-Gaming-Laptop-15-dk0xxx:~/spit/css/CSS-Lab-Deep-Nayak/Experiment 4$
```

rsa.py U

```python
p=7
q=23
N=p*q

PHI=(p-1)*(q-1)
e=5
for d in range(1,100):
    if ((e*d % PHI)==1): break
print (e,N)

print (d,N)
M=4
cipher = M**e % N

print (cipher)
message = cipher**d % N
print (message)
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

```
deep@deep-HP-Pavilion-Gaming-Laptop-15-dk0xxx:~/spit/css/CSS-Lab-Deep-Nayak/Experiment 4$ /bin/python3
ment 4/rsa.py"
5 161
53 161
58
4
deep@deep-HP-Pavilion-Gaming-Laptop-15-dk0xxx:~/spit/css/CSS-Lab-Deep-Nayak/Experiment 4$
```

rsa.py U

```python
p=19
q=13
N=p*q

PHI=(p-1)*(q-1)
e=5
for d in range(1,100):
    if ((e*d % PHI)==1): break
print (e,N)

print (d,N)
M=4
cipher = M**e % N

print (cipher)
message = cipher**d % N
print (message)
```

```
PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE
deep@deep-HP-Pavilion-Gaming-Laptop-15-dk0xxx:~/spit/css/CSS-Lab-Deep-Nayak/Experiment 4$ /bin/python3
ment 4/rsa.py"
5 247
99 247
36
77
deep@deep-HP-Pavilion-Gaming-Laptop-15-dk0xxx:~/spit/css/CSS-Lab-Deep-Nayak/Experiment 4$
```

E.2) OUTPUT :

In the RSA method, we have a value of e, and then determine d from (d.e) (mod PHI)=1. But how do we use code to determine d

inverse.py U

```python
        # Either n is 0, or p is not a prime number.
        raise ValueError(
            '{} has no multiplicative inverse '
            'modulo {}'.format(n, p))
    else:
        return x % p

val1=53
val2=120

print("Inverse of ",val1," mod ",val2)
print("Result:\t:",inverse_of(val1,val2))
```

```
PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE
deep@deep-HP-Pavilion-Gaming-Laptop-15-dk0xxx:~/spit/css/CSS-Lab-Deep-Nayak/Experiment 4$ /bin/python3
ment 4/inverse.py"
Inverse of  53  mod  120
Result: : 77
deep@deep-HP-Pavilion-Gaming-Laptop-15-dk0xxx:~/spit/css/CSS-Lab-Deep-Nayak/Experiment 4$
```

inverse.py U

```
        # Either n is 0, or p is not a prime number.
        raise ValueError(
            '{} has no multiplicative inverse '
            'modulo {}'.format(n, p))
    else:
        return x % p

val1=65537
val2=103477685183741822601240611393312008O
print("Inverse of ",val1," mod ",val2)
print("Result:\t:",inverse_of(val1,val2))
```

```
PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE

deep@deep-HP-Pavilion-Gaming-Laptop-15-dk0xxx:~/spit/css/CSS-Lab-Deep-Nayak/Experiment 4$ /bin/python3
ment 4/inverse.py"
Inverse of  65537  mod  1034776851837418226012406113933120080
Result: : 568411228254986589811047501435713
deep@deep-HP-Pavilion-Gaming-Laptop-15-dk0xxx:~/spit/css/CSS-Lab-Deep-Nayak/Experiment 4$
```

E.3) OUTPUT :

Run the following code and observe the output of the keys. If you now change the key generation key from ‘PEM’ to ‘DER’, how does the output change:

e3.py U

```
from Crypto.PublicKey import RSA

key = RSA.generate(2048)

binPrivKey = key.exportKey('PEM')

binPubKey = key.publickey().exportKey('PEM')

print(binPrivKey)

print(binPubKey)
```

```
PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE

deep@deep-HP-Pavilion-Gaming-Laptop-15-dk0xxx:~/spit/css/CSS-Lab-Deep-Nayak/Experiment 4$ /bin/python3 "/home/deep/spit/css/CSS-Lab-Deep-Nayak/Experiment 4/e3.py"
b'-----BEGIN RSA PRIVATE KEY-----\nMIIEpQIBAAKCAQEA57FwUcq0u42AdpVvFQ/s+SACd0FYPyGC9ZjfOmT9sXkgk89s\nJwpo2wp0N9IhmU5924uj8Firgxia2p7f5P+vsVp5hiJ4/FXMRYI17ndv+2MoafVj\nLXk9x4+XsuPmk+iKaeKrM7APT+H2Ph9lFwjZ2htlxhDTqziHydTRxe0MsnbHePHV\nlWqLJ3X0oUckvbJOpznHFdFzm1i4WOA7XiwMgBvJuYWrHQs1zxhiWEHhSMxiUYD+\ntML0GnQkGrN/jfV+TIBeI1W+K4ozrO4XQFI/NneE4W0p+9qatWQoyzZvNxtGxLkO\nTifNBpDHJaT1CYXVkzWqn1A1B2MnlTbqsDdIXQIDAQABAoIBAA7RsNEuca5gT9Pl\n3W+/r8TEaRv3FD2fd1idkUOTI43XJUAG/5AksI24BrRuToNXe6HP9emgxkGRkJAE\nouXq+7IQix7K9CmRlEnq2Oe6gHFYB4T1csU9ELhDpSyGOqCBul6o5pHdBDG0DnOt\nPuJBCOwUHgeT2nIEYN0GLW+nKI1+f7yWom8rKKYYxFg2vdFE/Mbrhu1qCbGYW3Nn\n4Zr9YKIZ7juAM0E4pelPO1qRfrg6Qx60qtQR1XFTMsQtd4xe7YVgdl6KyLRalT0M\nPtWEdksrAbaNFaXgX3ft2kTxdfWRxYJwmwvzWNVmzAqeoFL9Wc8fG8D48ToxTVrR\nhKT8BWECgYEA7iTAiMVn7dfps3EZ+ylsY7aETUUMQ81D9pMqpJEiCqicGxpbusn2\nh0lxtb9teg0CJ81+mmTztGOw9kGHNlRUwFzSRG4FdSOlcTN9eLcLdXT4cISKejK4\nXPzrs5duamSMyIA7q9TTnp2/hU0vItG88wETyDii8jKAqPSMCAdJ4WECgYEA+RDe\nQWtXj9/d8Ua09HIRuoP7RoyCPq7qcj1zsPpyik8qhuISm9nbrYhAz5K3aCa8gwqA\n7oEGIwvIIrm4xOgRNpi6tkwnJzvwsZRzPMI1WSVQ6oZV30l9xHk5zP2wDcxb49VM\n8JnjCu0vCYq8RYPyTewkjHO7+4lh263h1mZBvH0CgYEA5EkG6BWUk0OvG/0H0uTa\nqpuEKC5Qm45h4DUjY65c26rWOoUB+gUw9m9JOdjPsXhW4KD1BDxSAlKuiOI4I7Sj\nqeNRDZgKU0hgiCu708Rkj6wZnILx3EJSyZEKwZYgmozZCDsk+IuEcUD3H2nmDZLR\ndoFJ3V2tlk8jVtNWiTuzbwECgYEAuzfpyYUVscmWvJnoxeLE3s384OAXu2SiSLNW\nto7b8IlnrzJayJovnjbQePeRNBICHAKWPjHwWgGuNCQk9qZ24fJWT1fZCo3s9J9h\nLCfUECmTtGa8P/avOjPhtexwHT8jR5R7nFQgrYLym5qUwTBYwCgkpQsOIBr52zbO\njuoYucUCgYEAmx7BbrE4+zw0e+9ZmQUyPOih6j0JoOtiCyfH8yi9k4n00nfl+QiQ\n6q+Pn3t2LaCKo1ycjZ2TsP5T6LLFh+kAPx4t3x49h+9KW63Eu85fEUoH6ZHnV1WJ\n5ljSKqP+OV1VfpWo5VPjptDob2n9dB2HZznJjFPfnVPsJ3TMZiR5aQI=\n-----END RSA PRIVATE KEY-----'
b'-----BEGIN PUBLIC KEY-----\nMIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEA57FwUcq0u42AdpVvFQ/s\n+SACd0FYPyGC9ZjfOmT9sXkgk89sJwpo2wp0N9IhmU5924uj8Firgxia2p7f5P+v\nsVp5hiJ4/FXMRYI17ndv+2MoafVjLXk9x4+XsuPmk+iKaeKrM7APT+H2Ph9lFwjZ\n2htlxhDTqziHydTRxe0MsnbHePHVlWqLJ3X0oUckvbJOpznHFdFzm1i4WOA7XiwM\ngBvJuYWrHQs1zxhiWEHhSMxiUYD+tML0GnQkGrN/jfV+TIBeI1W+K4ozrO4XQFI/\nNneE4W0p+9qatWQoyzZvNxtGxLkOTifNBpDHJaT1CYXVkzWqn1A1B2MnlTbqsDdI\nXQIDAQAB\n-----END PUBLIC KEY-----'
deep@deep-HP-Pavilion-Gaming-Laptop-15-dk0xxx:~/spit/css/CSS-Lab-Deep-Nayak/Experiment 4$
```

e3.py U

```python
from Crypto.PublicKey import RSA

key = RSA.generate(2048)

binPrivKey = key.exportKey('DER')

binPubKey = key.publickey().exportKey('DER')

print(binPrivKey)

print(binPubKey)
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

```
deep@deep-HP-Pavilion-Gaming-Laptop-15-dk0xxx:~/spit/css/CSS-Lab-Deep-Nayak/Experiment 4$ /bin/python3 "/home/deep/spit/css/CSS-Lab-Deep-Nayak/Experiment 4/e3.py"
b'0\x82\x04\xa2\x02\x01\x00\x02\x82\x01\x01\x00\xa99\xc8R\x971\x8fj\xcc[D\xfb)Bq\x017\x07bli\x8d\xa6\x83;\x16I\x1dJU\xe2\xaf\xfc\xa8\xa1\x99P\xdb3\x17U\xaa\xfa\x08\x0c\xfbLr\xdb\x8e\xed\xc6\x03\x8eai\xe2^\x0c4\xf1\xec\x96V\xb5.\x91\x8c\xf655{\xe5\x1e\xf8.\x08~\xb8\xa1\x8f\xde\x80u\xba\xa7\xef\xf86z\x04\\\x93\xc8\xd5#\xca\x87:\x17\xa2\xa3\xd61\x99\xc7nb\xcaK\x1e\x1f\x08\x13\x18\x93c[\xdf\xc0\x8d\xd8\x89[U|\xea\xbcl\xcc\xf8x*\x924<\x1a6\xa9\x92\x8b\xb9\xb1#\x1b\x110\xbaF\xc2M\x8d\xa29L\x90\xfc,\x82\x8f\xb0\'O\xfb\xb1\xc7H\xca\x19 \n\xd31\xf6\xbc\x07\xecqb\xea\xb8\x88\xce\xe3\x8c\xc0.#\x83\x1a\xb6\xf5\xff\x11K\x89j\xb9X\xac\xfc\xbb\xc7\xff\x14\xa3C\xd3\xa0\x05\'\x8c\xaf\xe2\x01<\x86\xf9\xf9/\x8f\xd5\x1c\xf0\n\x1e\xa4\x1c\xcfC\xae\x1eud\xe4\xd1\xaf\xd6O\'\xfe\x8d,\x95_\xc7E\xbf\xdc\xe2\xfe.\xd7\x1b\x9fS\x02\x03\x01\x00\x01\x02\x82\x01\x00N\x13+1\xb3\xb0\xf9n\xb8\'\x19\xcd\xd9,\xde\xcfE0\xc2\x7f\xab9_\xcc\'\x0c2\xa2qc\xeb\xa8\xefdk\xde<\xf8L\xfb\xa4\x9a4.W\xb2\xef\x84\x04yN\x89o\xa5\x96\x9b\x10\x05\xfd\x12\xc6U5\xcd~]$i\tD\xe6N\xa9\x98\x9e\xbf:K \xea\xe5b.\x19c\x0bMCR\xfa\xfb\x9d,\xcf\xd9J`\x8c\x00H\xdcPC\xa7\xf1\xbf\x99\xd5\xb9hk\xd6\xbcVw\x1d\xb5\x173\xc5\xf6\x8b\xa3_\xc0\xb2\xd7\xe7d\';g\x91\x1b\x03hzO\xf80be\xb6\xe8\xc4\x1fc\x14$\xd6\xbc\xb0;\x1bJe\xbeD8\xc8G\xd2\xearb\t5\xc2\xfcJ\x1e\xad;\xfc\xea\x87\xc5^\xa9\xc5\xdc\x8dxO\x83\x9f\x1f\xf8(\xa9\xd7\x10\xb0\xd4\xb71\x80o\xd0\x87\xa1\xc4\xe4\xba\xc47\xf1\xb5\xc0IA\\D\xe4\xdc\xd5\xd1L|\x8d\xf5\xb7\x81ot\x8b\x9c\x8dI\x0b|\xb3\xbb\xaf\xf0x\xf7\xeb:[\x12\xb1\xfc\xec\xba\x81\xfa<\x9c\xb9\xb3\x1e\t\xb4LQ\x02\x81\x81\x00\xcd\xe8k[\xf4\'\x99\x97%\xffA\x90F\x84%wP\t\xa2N\xe0\xf6!o\xf4$\x96I8\xb7{\x8e\xf7\xce\x99\xf6\x80A5YW\xa4\x04\xa1\xa5\xb2[\xf6\xe6P\x8bp\xf2\x92\xedXP\x8e\xec\xc6\x9f\xdc\x95\xbc\x01\x13\xa6\xe8\xb4\xdd\xeb=\x80\x88D\xee\xdf\xa4\xa9\x88\xa8~\xf5f\x9a\x88iY\x7fhoD\x91\x9d\x1b\xbf\xb0:\x9a\x8f\xfd\xee\x7f\xe91:\xb6%F\xf3\xc3k\xfe\x80,5\r\xd2\x1b$\x86e/\xfa%\xffe\xe3\x02\x81\x81\x00\xd2d\xdc\xa3y\x85\x9c\xc5\x00\x9b\x94&\x91\xf3\x11L\xa51\xfd9\xc9\xd2\xa5\x11\xe4[\x15\x01\x15A\xe3\x84\xf26\xc8y\x94]\xba\xe9\x82\x94\xc0Kl\xb5\xf9\xbd\xf10\xe3\x8f\xbf\xf2F\x82\x12\x14\x13\x0f\x97@7\x01\xae\x87F\x96h\xe0Q5U\x87\xa4\xfb\xbc8\x9a\x8a{\xb7\x9c" \x99\xd8$y\xa3\xf8\xc4g~\x1cf\x9c\x03\xa6\x95\xc1\xf2V\xe6\x9e\xfe-\x05\xf5\x1e\x9fW\'{\xc0|Lnv\x00At\x1b\xbc\x91\xc3\x9b\xd1\x02\x81\x80\x07s\x08\'L~\xfeD\x84\xa0f{\xd4\xc1E%\xba\xe4\x8eU\xb27\xd5_\xda\x82TY\x93\xce\xb7\x8e\xa5\x92x\xe9}\x1c^\xd4\xbb\xa1\xf8t\xec\xac\xb6=\tz\x12\x81d\x95B6\xf6\xbb$\xff\xce\x9a\xc9d\x02\xd9\xf6(\xe7\x1a<?~]\xef\xb8\x80\xa8\xd1e[K4\x8f\xa0\xb9\x9c\xae\x1b8\xe9\x05\x9f\x0b\x03\x0c\xfdP1uR\'\x85\xc3&\xff\xcb\xc5/z\xf4/\xfd\x06 \x13OK\xb1\xa5\xb1\xa3\t\x88PE\x9c\'\x02\x81\x808\xe2*AL\x9e\xd2V\xf5&\x1b5\x1ej\xc0w\x9f\x86Ww\xa3X\x7ff\xc9\xfd\xd2<\xfb\xb9W\xc1_\x17n\xb6he\xbb\x1e]>\xc8\x8a\xdb\xf6\x8brN^\xb0w\xf2&\x97\xe7\x9c\xe9\xae\x89\x9d\x940\x0b\x15[\xdf\xba\xd4\xda):\xa9HFT\x05|\xacS\xda\xc56\xc2\x87[\xf0\xf3S\xac\x00\x9ba{\xd8C\xe9\xd4\x96\xda\x90\x98k\xe4h\x82B<\x05\x1c\xe5i\xd3\xf4\x02\x83\xd8\xcaX\xbf\x12\xf3Hx\x80T\x89\xb1\x02\x81\x80gc\xc2\xf9R\xe4\xa2\xbd\x9eT\xfe\xac\xf9\x1d\x8c@}\x15\x8a\x19Fs\xf0E[!$\x93\x9c\xa2\x9f:G\xad\x14C\x03_$\x84\x00\x7f-\xcd\xf5so\x94\xd4\t\x95U\x11\xa3\x9e4$\xd9\xad\xa2\x85\xb0\x9e\xf3\xb9\xae\xeb\x9b\x06\xb8b=_\xe6\xd4\'r\xdcOHj\x1e\xaa\x1f\xb41#\xcb1\x1e\x03\x84\x0b7\xc6e\xb3\xfb\xff\xc2\xa4\xb3)\xb7\xb0\x80\xb6\x08\x94o\xc7\xab\x02\x89\xc8\x8b\x19d\x9c\xfc)\x93G\x8f\xf4\xf0\x7fh'
```

The data that makes up the certificate is encoded using the DER technique. PEM is a way of encoding binary data as a string. DER can represent any type of data, although it commonly refers to an encoded certificate or a CMS container (ASCII armor). It has a header and a footer line (which indicate the kind of data encoded and show where the data begins and ends if the data is chained together), and the data in the centre is base 64 data. PEM stands for Privacy Enhanced Mail, which means that mail cannot contain unencoded binary values like DER.

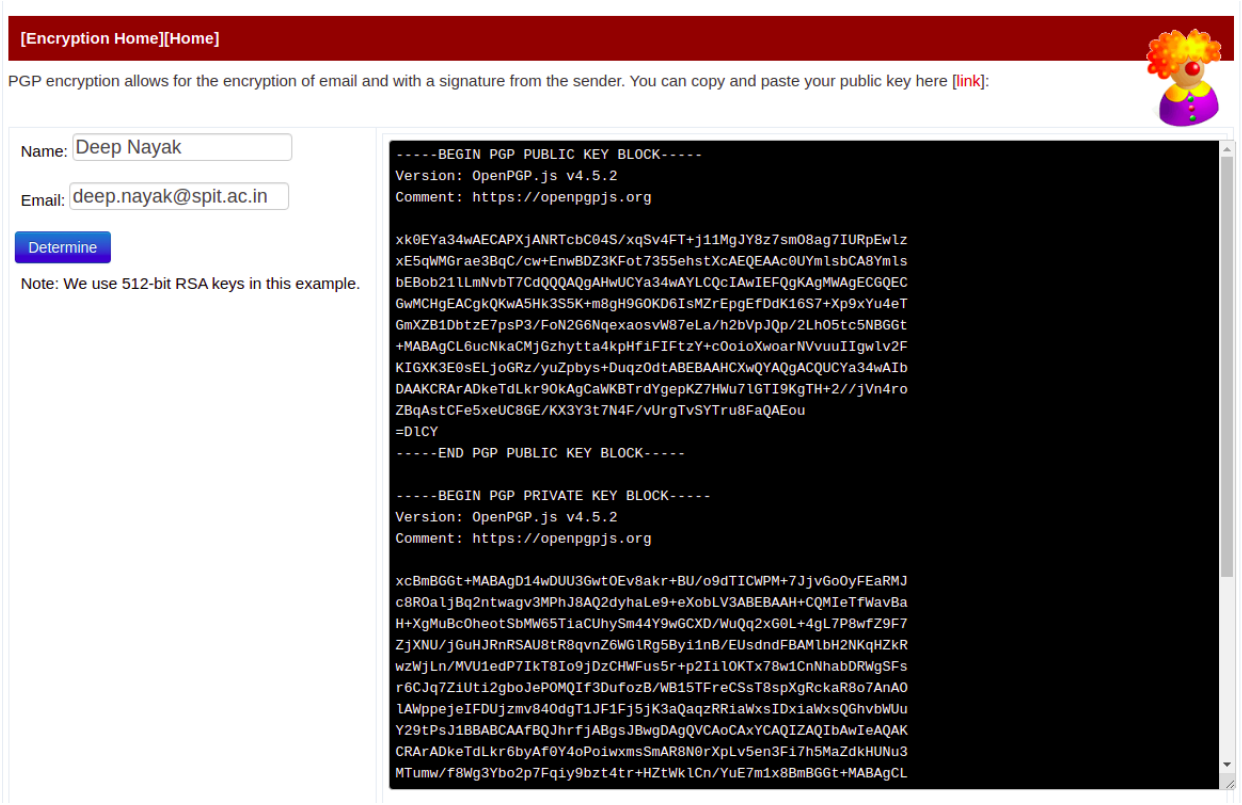
PGP

F.1) OUTPUT :

Not possible to implement as the web tool provided does not give the required output

F.2) OUTPUT :

Generating a key for Deep Nayak and deep.nayak@spit.ac.in



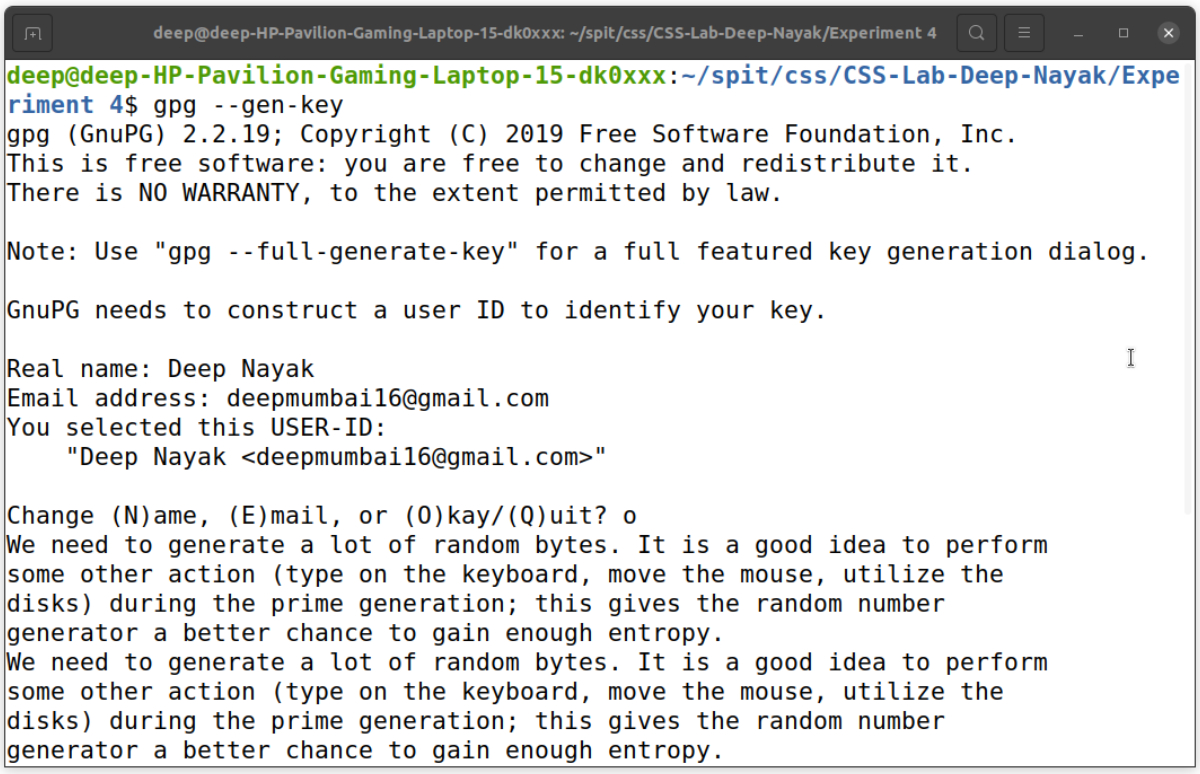
F.3) OUTPUT :

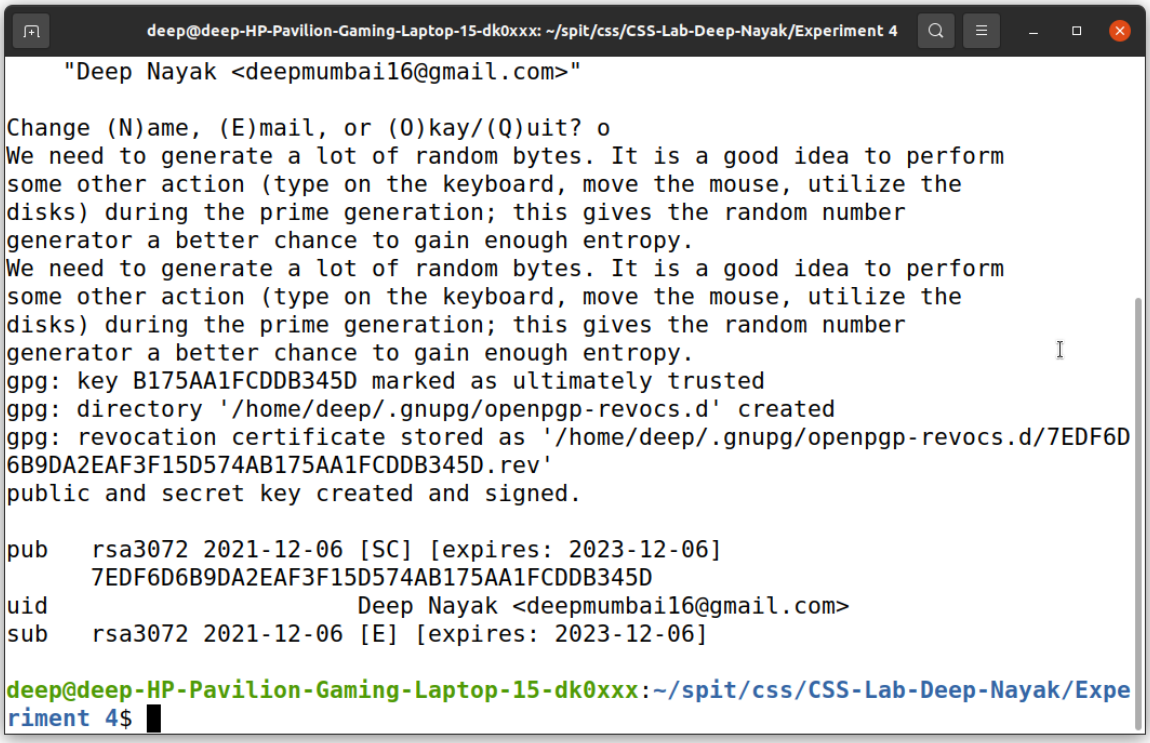
No	Description	Result
1	Create a key pair with (RSA and 2,048-bit keys): gpg --gen-key Now export your public key using the form of: gpg --export -a "Your name" > mypub.key Now export your private key using the form of: gpg--export-secret-key -a "Your name" > mypriv.key	How is the randomness generated? **PGP creates a session key, which is a secret key that can only be created once. The movement of our cursor and the keystrokes we enter produce a random number when we press this key. The plaintext is encrypted with this session key using an extremely safe and fast**

		symmetric encryption method, and the result is ciphertext. Outline the contents of your key file: **Both files include a header and footer that indicate whether they are PGP Public or Private key blocks, and the text in between is the actual key.**
2	Now send your lab partner your public key in the contents of an email, and ask them to import it onto their key ring (if you are doing this on your own, create another set of keys to simulate another user, or use Bill's public key – which is defined at http://asecuritysite.com/public.txt and send the email to him): gpg --import their publickey.key Now list your keys with: gpg --list-keys	Which keys are stored on your key ring and what details do they have: **After establishing another Dummy User (Bhushan Patil) and importing their key, I saw that the list included both my personal key and the key of the user I had just made. The user's public key encryption algorithm (RSA), uid, which includes the user's name and email address, and the pgp key's expiration date were also shown.**
3	Create a text file, and save it. Next encrypt the file with their public key: gpg -e -a -u "Your Name" -r "Your Lab Partner Name" hello.txt	What does the –a option do: **Create ASCII armored output. The default is to create the binary OpenPGP format** What does the –r option do:

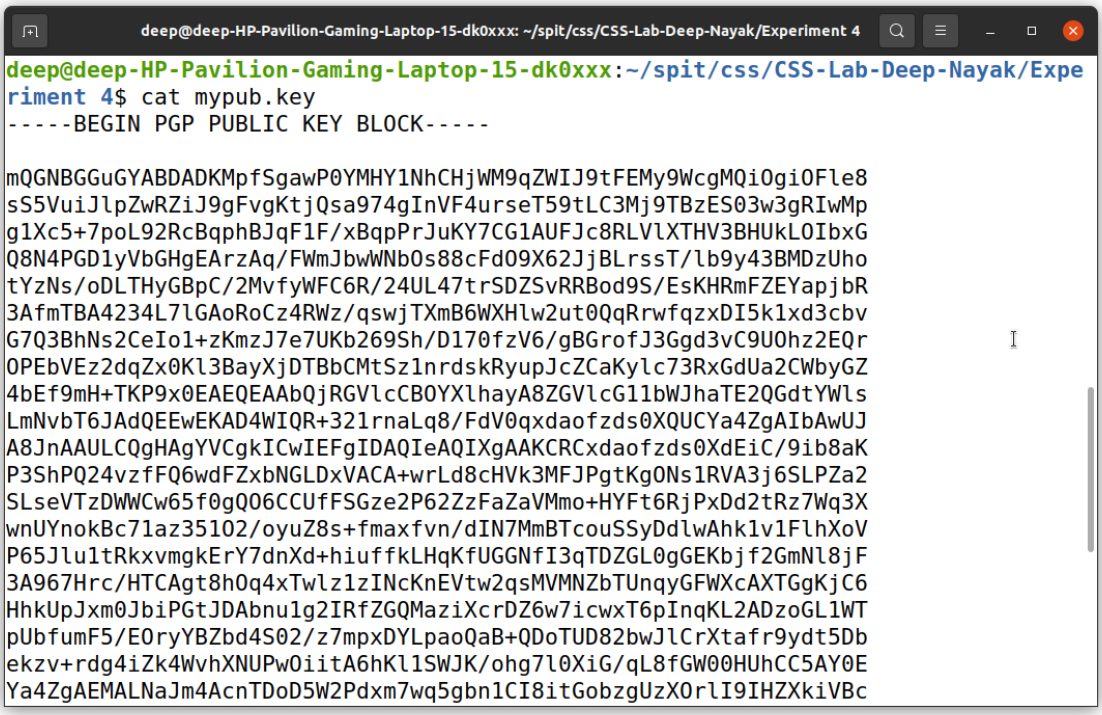
		Encrypt the text of the user id. Unless '—default-recipient' is set, GnuPG will ask for the user-id if this option or '—hidden-recipient' is not defined. What does the –u option do: **Use name as the key to sign with. Note that this option overrides '--default-key'.** Which file does it produce and outline the format of its contents: **It generates an .asc (ascii armoured file) in which the header and footer indicate the start and end of the PGP message, respectively, and the actual encrypted message is stored between them.**
4	Send your encrypted file in an email to your lab partner, and get one back from them. Now create a file (such as myfile.asc) and decrypt the email using the public key received from them with: gpg –d myfile.asc > myfile.txt	Can you decrypt the message: Yes

1

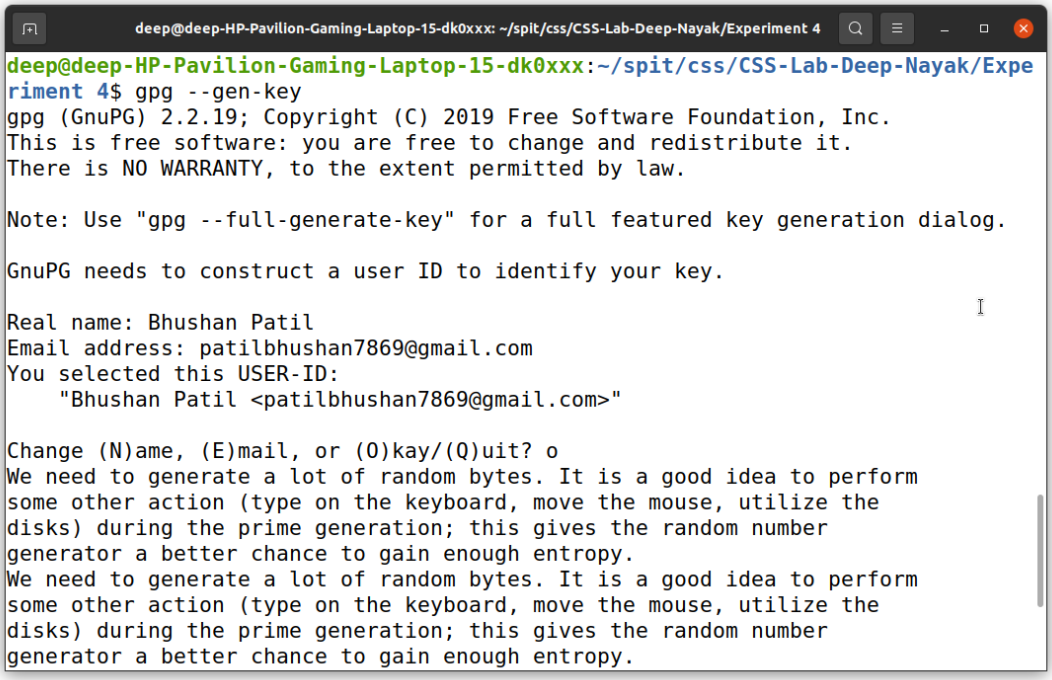

deep@deep-HP-Pavilion-Gaming-Laptop-15-dk0xxx: ~/spit/css/CSS-Lab-Deep-Nayak/Experiment 4
deep@deep-HP-Pavilion-Gaming-Laptop-15-dk0xxx:~/spit/css/CSS-Lab-Deep-Nayak/Experiment 4$ gpg --gen-key
gpg (GnuPG) 2.2.19; Copyright (C) 2019 Free Software Foundation, Inc.
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Note: Use "gpg --full-generate-key" for a full featured key generation dialog.
GnuPG needs to construct a user ID to identify your key.
Real name: Deep Nayak
Email address: deepmumbai16@gmail.com
You selected this USER-ID:
"Deep Nayak <deepmumbai16@gmail.com>"
Change (N)ame, (E)mail, or (O)kay/(Q)uit? o
We need to generate a lot of random bytes. It is a good idea to perform
some other action (type on the keyboard, move the mouse, utilize the
disks) during the prime generation; this gives the random number
generator a better chance to gain enough entropy.
We need to generate a lot of random bytes. It is a good idea to perform
some other action (type on the keyboard, move the mouse, utilize the
disks) during the prime generation; this gives the random number
generator a better chance to gain enough entropy.

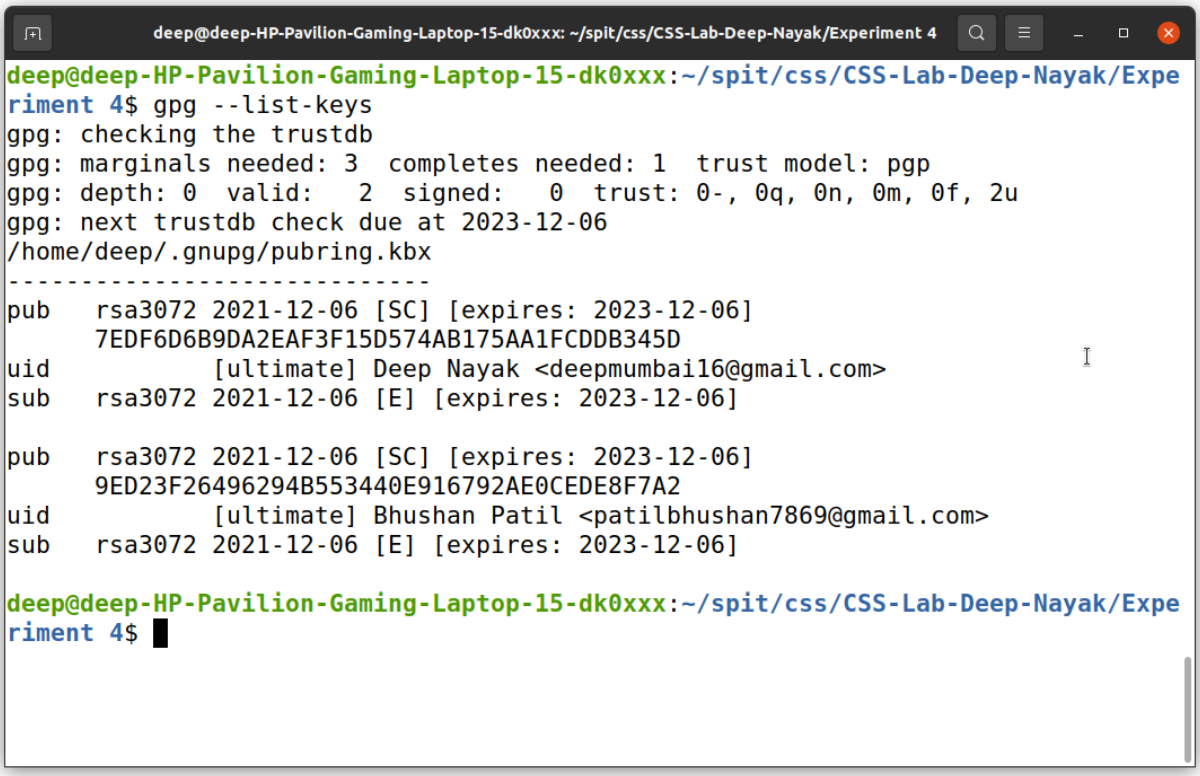

deep@deep-HP-Pavilion-Gaming-Laptop-15-dk0xxx: ~/spit/css/CSS-Lab-Deep-Nayak/Experiment 4
"Deep Nayak <deepmumbai16@gmail.com>"
Change (N)ame, (E)mail, or (O)kay/(Q)uit? o
We need to generate a lot of random bytes. It is a good idea to perform
some other action (type on the keyboard, move the mouse, utilize the
disks) during the prime generation; this gives the random number
generator a better chance to gain enough entropy.
We need to generate a lot of random bytes. It is a good idea to perform
some other action (type on the keyboard, move the mouse, utilize the
disks) during the prime generation; this gives the random number
generator a better chance to gain enough entropy.
gpg: key B175AA1FCDDB345D marked as ultimately trusted
gpg: directory '/home/deep/.gnupg/openpgp-revocs.d' created
gpg: revocation certificate stored as '/home/deep/.gnupg/openpgp-revocs.d/7EDF6D6B9DA2EAF3F15D574AB175AA1FCDDB345D.rev'
public and secret key created and signed.
pub rsa3072 2021-12-06 [SC] [expires: 2023-12-06]
7EDF6D6B9DA2EAF3F15D574AB175AA1FCDDB345D
uid Deep Nayak <deepmumbai16@gmail.com>
sub rsa3072 2021-12-06 [E] [expires: 2023-12-06]
deep@deep-HP-Pavilion-Gaming-Laptop-15-dk0xxx:~/spit/css/CSS-Lab-Deep-Nayak/Experiment 4$

```
deep@deep-HP-Pavilion-Gaming-Laptop-15-dk0xxx:~/spit/css/CSS-Lab-Deep-Nayak/Expe
riment 4$ gpg --export -a "Deep Nayak" > mypub.key
deep@deep-HP-Pavilion-Gaming-Laptop-15-dk0xxx:~/spit/css/CSS-Lab-Deep-Nayak/Expe
riment 4$ gpg --export-secret-key -a "Deep Nayak" > mypriv.key
```

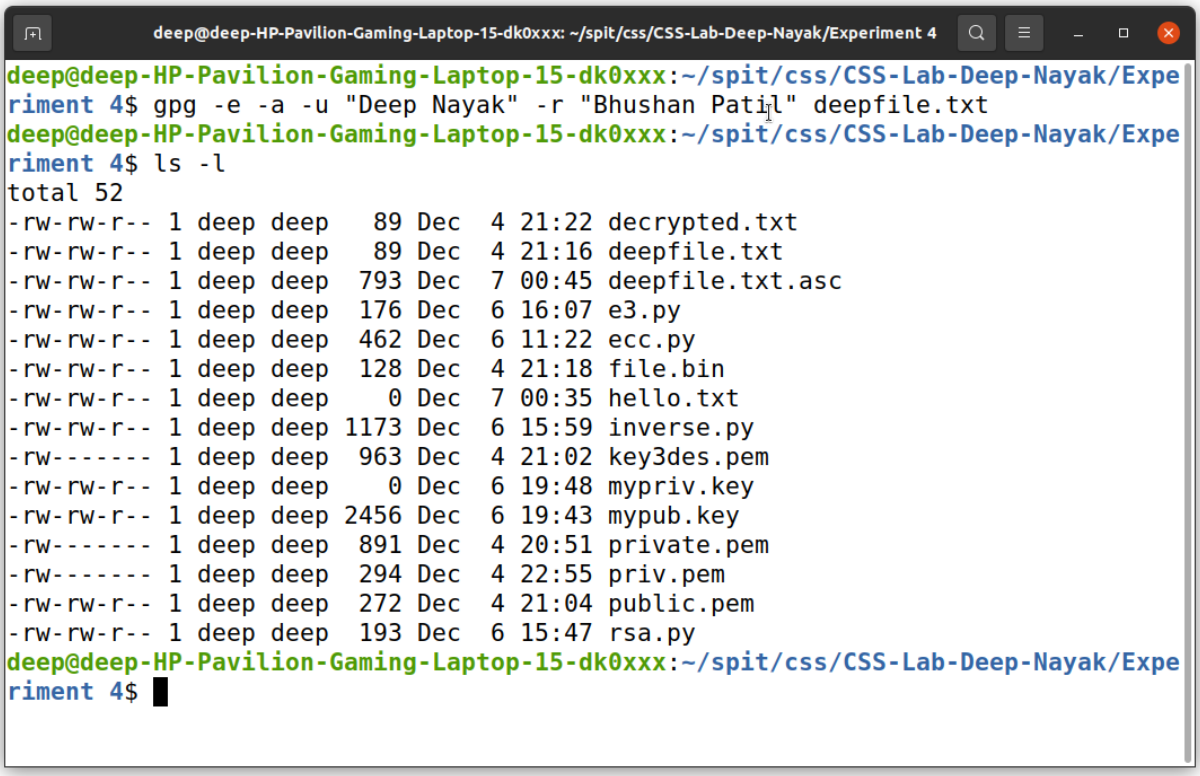


2




deep@deep-HP-Pavilion-Gaming-Laptop-15-dk0xxx: ~/spit/css/CSS-Lab-Deep-Nayak/Experiment 4
deep@deep-HP-Pavilion-Gaming-Laptop-15-dk0xxx:~/spit/css/CSS-Lab-Deep-Nayak/Experiment 4$ gpg --list-keys
gpg: checking the trustdb
gpg: marginals needed: 3 completes needed: 1 trust model: pgp
gpg: depth: 0 valid: 2 signed: 0 trust: 0-, 0q, 0n, 0m, 0f, 2u
gpg: next trustdb check due at 2023-12-06
/home/deep/.gnupg/pubring.kbx
----
pub   rsa3072 2021-12-06 [SC] [expires: 2023-12-06]
      7EDF6D6B9DA2EAF3F15D574AB175AA1FCDDB345D
uid   [ultimate] Deep Nayak <deepmumbai16@gmail.com>
sub   rsa3072 2021-12-06 [E] [expires: 2023-12-06]
pub   rsa3072 2021-12-06 [SC] [expires: 2023-12-06]
      9ED23F26496294B553440E916792AE0CEDE8F7A2
uid   [ultimate] Bhushan Patil <patilbhushan7869@gmail.com>
sub   rsa3072 2021-12-06 [E] [expires: 2023-12-06]
deep@deep-HP-Pavilion-Gaming-Laptop-15-dk0xxx:~/spit/css/CSS-Lab-Deep-Nayak/Experiment 4$

3


deep@deep-HP-Pavilion-Gaming-Laptop-15-dk0xxx: ~/spit/css/CSS-Lab-Deep-Nayak/Experiment 4
deep@deep-HP-Pavilion-Gaming-Laptop-15-dk0xxx:~/spit/css/CSS-Lab-Deep-Nayak/Experiment 4$ gpg -e -a -u "Deep Nayak" -r "Bhushan Patil" deepfile.txt
deep@deep-HP-Pavilion-Gaming-Laptop-15-dk0xxx:~/spit/css/CSS-Lab-Deep-Nayak/Experiment 4$ ls -l
total 52
-rw-rw-r-- 1 deep deep   89 Dec 4 21:22 decrypted.txt
-rw-rw-r-- 1 deep deep   89 Dec 4 21:16 deepfile.txt
-rw-rw-r-- 1 deep deep 793 Dec 7 00:45 deepfile.txt.asc
-rw-rw-r-- 1 deep deep 176 Dec 6 16:07 e3.py
-rw-rw-r-- 1 deep deep 462 Dec 6 11:22 ecc.py
-rw-rw-r-- 1 deep deep 128 Dec 4 21:18 file.bin
-rw-rw-r-- 1 deep deep   0 Dec 7 00:35 hello.txt
-rw-rw-r-- 1 deep deep 1173 Dec 6 15:59 inverse.py
-rw------- 1 deep deep 963 Dec 4 21:02 key3des.pem
-rw-rw-r-- 1 deep deep   0 Dec 6 19:48 mypriv.key
-rw-rw-r-- 1 deep deep 2456 Dec 6 19:43 mypub.key
-rw------- 1 deep deep 891 Dec 4 20:51 private.pem
-rw------- 1 deep deep 294 Dec 4 22:55 priv.pem
-rw-rw-r-- 1 deep deep 272 Dec 4 21:04 public.pem
-rw-rw-r-- 1 deep deep 193 Dec 6 15:47 rsa.py
deep@deep-HP-Pavilion-Gaming-Laptop-15-dk0xxx:~/spit/css/CSS-Lab-Deep-Nayak/Experiment 4$

4

```
deep@deep-HP-Pavilion-Gaming-Laptop-15-dk0xxx:~/spit/css/CSS-Lab-Deep-Nayak/Expe
riment 4$ gpg -d deepfile.txt.asc > decoded.txt
gpg: encrypted with 3072-bit RSA key, ID 1090F15029022726, created 2021-12-06
      "Bhushan Patil <patilbhushan7869@gmail.com>"
deep@deep-HP-Pavilion-Gaming-Laptop-15-dk0xxx:~/spit/css/CSS-Lab-Deep-Nayak/Expe
riment 4$ ls -l
total 56
-rw-rw-r-- 1 deep deep   89 Dec  7 11:11 decoded.txt
-rw-rw-r-- 1 deep deep   89 Dec  4 21:22 decrypted.txt
-rw-rw-r-- 1 deep deep   89 Dec  4 21:16 deepfile.txt
-rw-rw-r-- 1 deep deep  793 Dec  7 00:45 deepfile.txt.asc
-rw-rw-r-- 1 deep deep  176 Dec  6 16:07 e3.py
-rw-rw-r-- 1 deep deep  462 Dec  6 11:22 ecc.py
-rw-rw-r-- 1 deep deep  128 Dec  4 21:18 file.bin
-rw-rw-r-- 1 deep deep    0 Dec  7 00:35 hello.txt
-rw-rw-r-- 1 deep deep 1173 Dec  6 15:59 inverse.py
-rw------- 1 deep deep  963 Dec  4 21:02 key3des.pem
-rw-rw-r-- 1 deep deep    0 Dec  6 19:48 mypriv.key
-rw-rw-r-- 1 deep deep 2456 Dec  6 19:43 mypub.key
-rw------- 1 deep deep  891 Dec  4 20:51 private.pem
-rw------- 1 deep deep  294 Dec  4 22:55 priv.pem
```

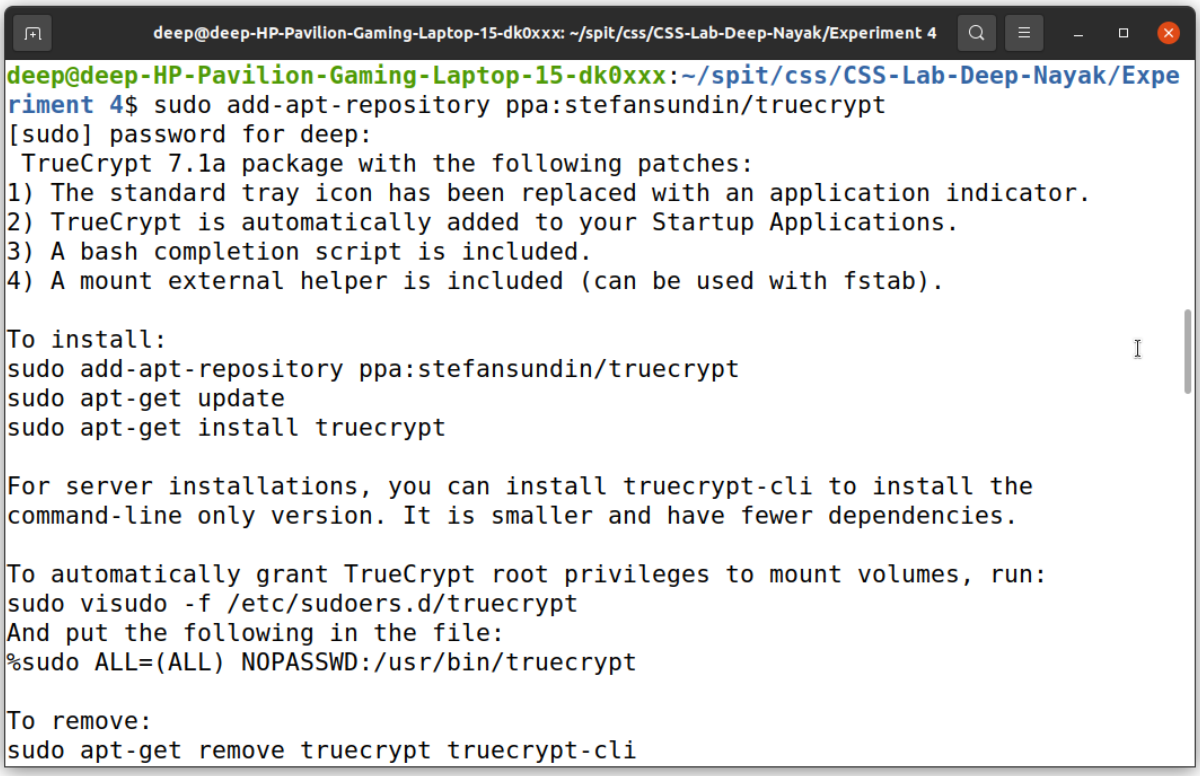
```
deep@deep-HP-Pavilion-Gaming-Laptop-15-dk0xxx:~/spit/css/CSS-Lab-Deep-Nayak/Expe
riment 4$ cat decoded.txt
You can't connect the dots looking forward; you can only connect them looking ba
ckwards.
deep@deep-HP-Pavilion-Gaming-Laptop-15-dk0xxx:~/spit/css/CSS-Lab-Deep-Nayak/Expe
riment 4$ cat deepfile.txt
You can't connect the dots looking forward; you can only connect them looking ba
ckwards.
```

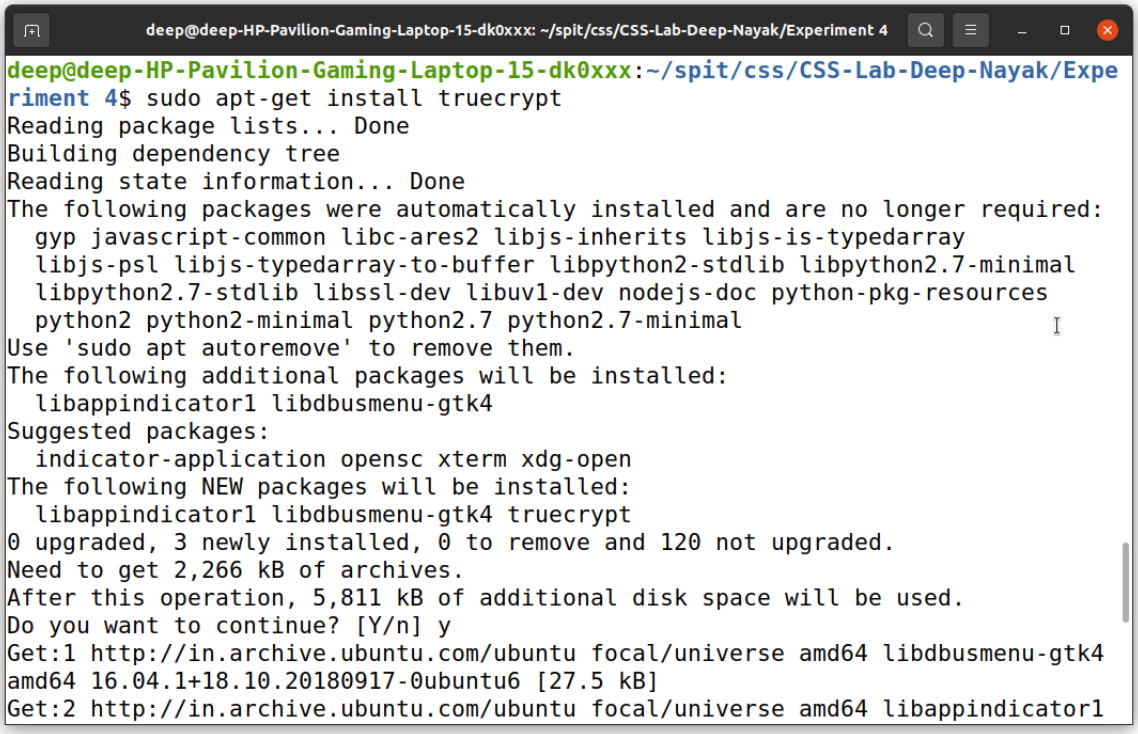
G) TrueCrypt

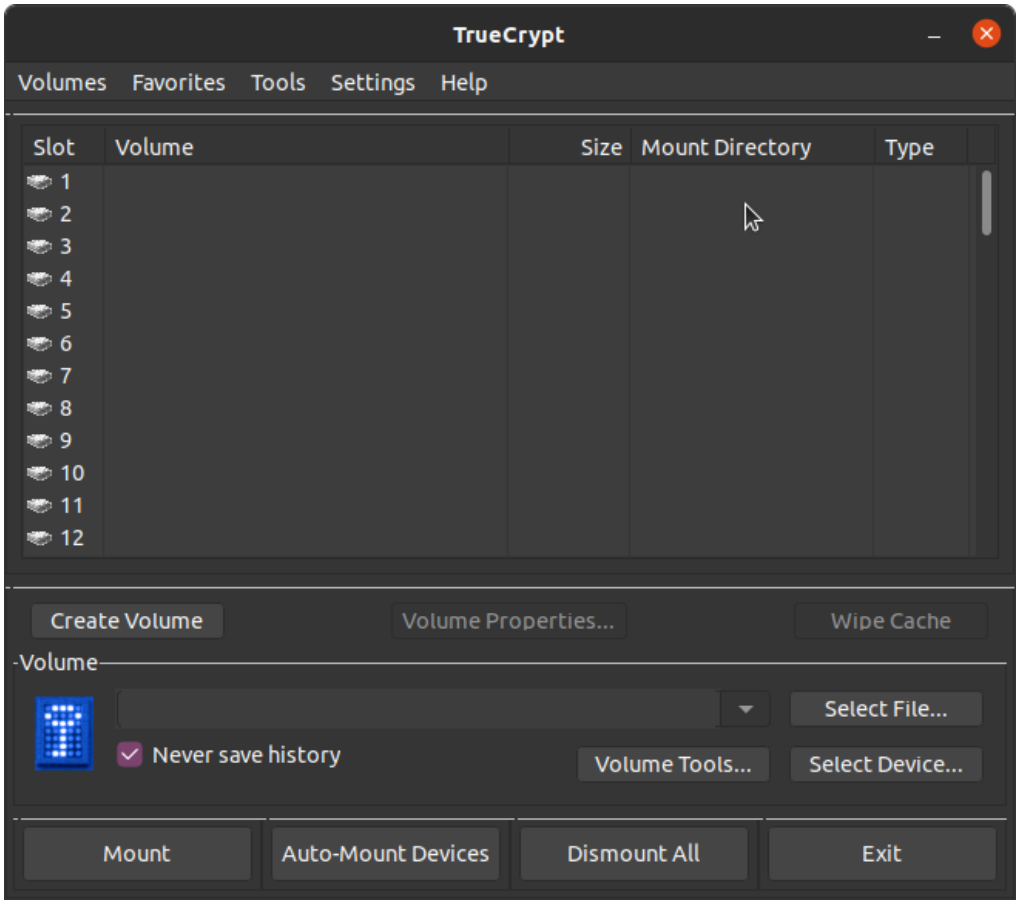
No	Description	Result
1	Go to your Kali instance (User: root, Password:toor). Now Create a new volume and use an CPU (Mean) encrypted file container (use tc_yourname) with a Standard TrueCrypt volume. When you get to the Encryption Options, run the AES-Two-Seperate benchmark tests and outline the results:	CPU (Mean) AES: **4.4 GB/s** AES-Twofish: **889 MB/s** AES-Two-Seperent Serpent -AES : **526 MB/s** Serpent: **391 MB/s** Serpent-Twofish-AES: **322 MB/s** Twofish: **966 MB/s** Twofish-Serpent: **341 MB/s**

		Which is the fastest: **AES** Which is the slowest: **AES-Twofish-Serpent**
2	Select AES and RIPEMD-160 and create a 100MB file. Finally select your password and use FAT for the file system.	What does the random pool generation do, and what does it use to generate the random key? **The random pool simply collects the user's mouse movements indefinitely since it is also notified on the screen that the user must move the mouse within the window as randomly as possible, which helps to increase the encryption keys' cryptographic strength.**
3	Now mount the file as a drive.	Can you view the drive on the file viewer and from the console? **Yes**
4	Create some files on your TrueCrypt drive and save them.	**I was able to perform any operations on the files I created and saved, but when I demounted the volume and tried to access it, only a file named 'tc_deepnayak' was displayed. When I tried to mount the volume again, I was prompted to enter the password that I had set earlier, and only after entering the correct password was I able to access all of my files saved in that volume.**

G1

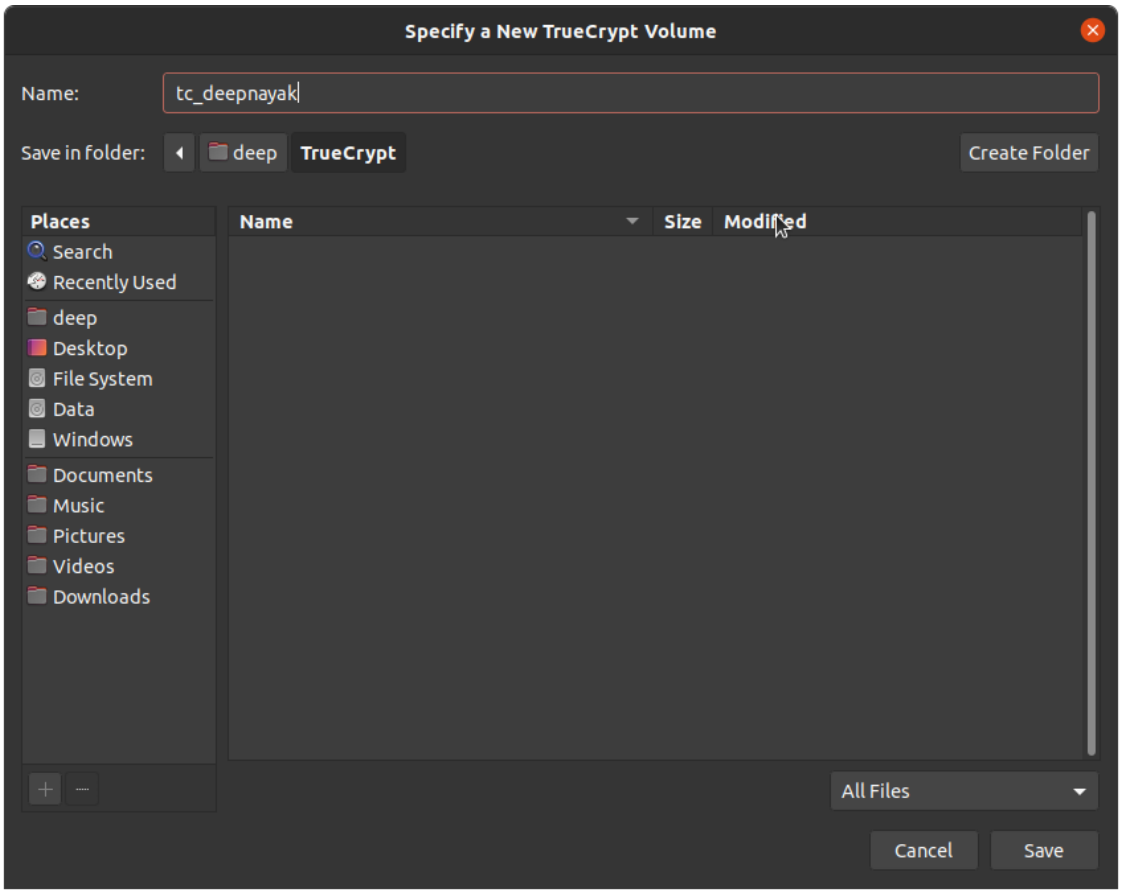

deep@deep-HP-Pavilion-Gaming-Laptop-15-dk0xxx: ~/spit/css/CSS-Lab-Deep-Nayak/Experiment 4
deep@deep-HP-Pavilion-Gaming-Laptop-15-dk0xxx:~/spit/css/CSS-Lab-Deep-Nayak/Experiment 4$ sudo add-apt-repository ppa:stefansundin/truecrypt
[sudo] password for deep:
TrueCrypt 7.1a package with the following patches:
1) The standard tray icon has been replaced with an application indicator.
2) TrueCrypt is automatically added to your Startup Applications.
3) A bash completion script is included.
4) A mount external helper is included (can be used with fstab).
To install:
sudo add-apt-repository ppa:stefansundin/truecrypt
sudo apt-get update
sudo apt-get install truecrypt
For server installations, you can install truecrypt-cli to install the
command-line only version. It is smaller and have fewer dependencies.
To automatically grant TrueCrypt root privileges to mount volumes, run:
sudo visudo -f /etc/sudoers.d/truecrypt
And put the following in the file:
%sudo ALL=(ALL) NOPASSWD:/usr/bin/truecrypt
To remove:
sudo apt-get remove truecrypt truecrypt-cli

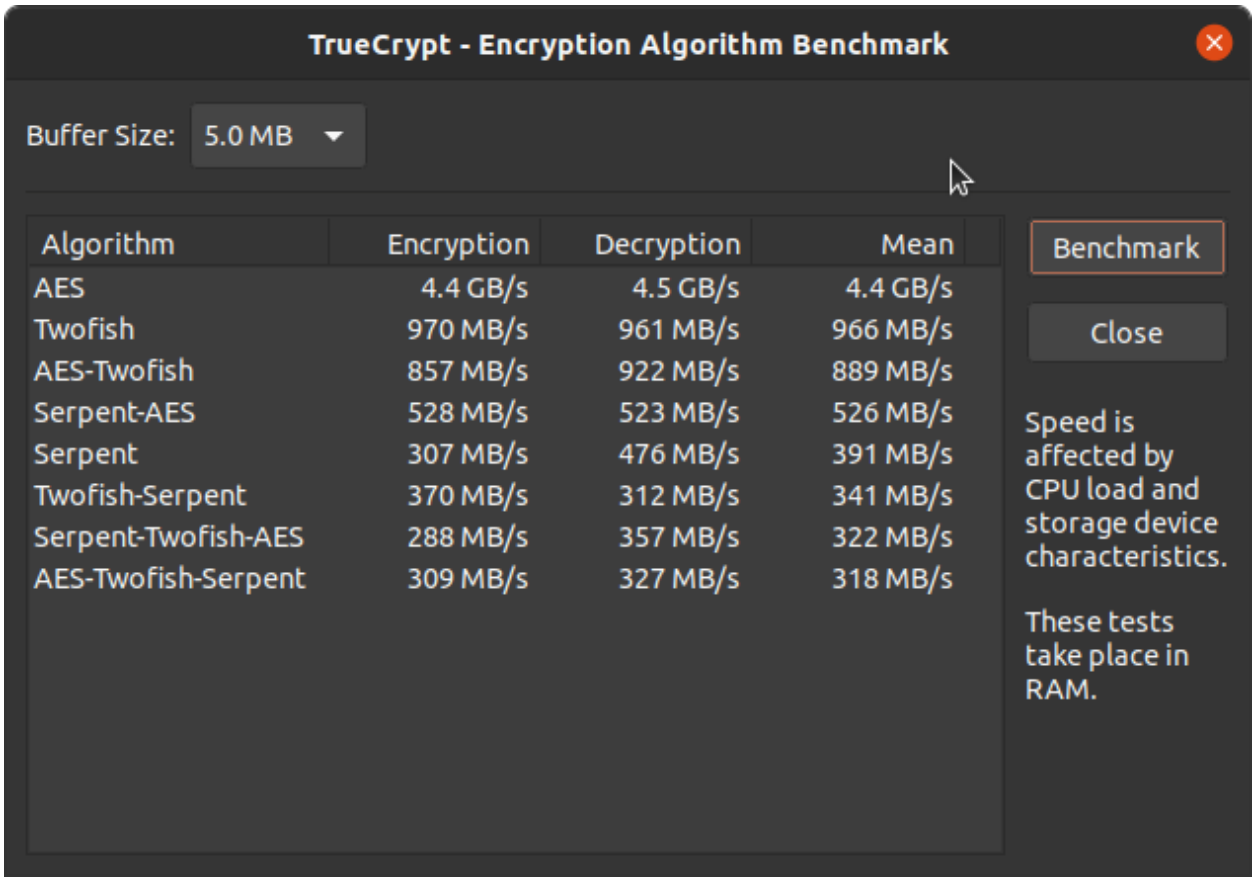

deep@deep-HP-Pavilion-Gaming-Laptop-15-dk0xxx: ~/spit/css/CSS-Lab-Deep-Nayak/Experiment 4
deep@deep-HP-Pavilion-Gaming-Laptop-15-dk0xxx:~/spit/css/CSS-Lab-Deep-Nayak/Experiment 4$ sudo apt-get install truecrypt
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following packages were automatically installed and are no longer required:
gyp javascript-common libc-ares2 libjs-inherits libjs-is-typedarray
libjs-psl libjs-typedarray-to-buffer libpython2-stdlib libpython2.7-minimal
libpython2.7-stdlib libssl-dev libuv1-dev nodejs-doc python-pkg-resources
python2 python2-minimal python2.7 python2.7-minimal
Use 'sudo apt autoremove' to remove them.
The following additional packages will be installed:
libappindicator1 libdbusmenu-gtk4
Suggested packages:
indicator-application opensc xterm xdg-open
The following NEW packages will be installed:
libappindicator1 libdbusmenu-gtk4 truecrypt
0 upgraded, 3 newly installed, 0 to remove and 120 not upgraded.
Need to get 2,266 kB of archives.
After this operation, 5,811 kB of additional disk space will be used.
Do you want to continue? [Y/n] y
Get:1 http://in.archive.ubuntu.com/ubuntu focal/universe amd64 libdbusmenu-gtk4
amd64 16.04.1+18.10.20180917-0ubuntu6 [27.5 kB]
Get:2 http://in.archive.ubuntu.com/ubuntu focal/universe amd64 libappindicator1


TrueCrypt
Volumes
Favorites
Tools
Settings
Help
Slot
Volume
Size
Mount Directory
Type
1
2
3
4
5
6
7
8
9
10
11
12
Create Volume
Volume Properties...
Wipe Cache
Volume
Select File...
Never save history
Volume Tools...
Select Device...
Mount
Auto-Mount Devices
Dismount All
Exit


TrueCrypt Volume Creation Wizard
TRUECRYPT
TrueCrypt Volume Creation Wizard
Create an encrypted file container
Creates a virtual encrypted disk within a file. Recommended for inexperienced users.
More information
Create a volume within a partition/drive
Formats and encrypts a non-system partition, entire external or secondary drive, entire USB stick, etc.
Help
< Prev
Next >
Cancel


TrueCrypt Volume Creation Wizard
Volume Type
Standard TrueCrypt volume
Select this option if you want to create a normal TrueCrypt volume.
Hidden TrueCrypt volume
It may happen that you are forced by somebody to reveal the password to an encrypted volume. There are many situations where you cannot refuse to reveal the password (for example, due to extortion). Using a so-called hidden volume allows you to solve such situations without revealing the password to your volume.
More information about hidden volumes
Help
< Prev
Next >
Cancel

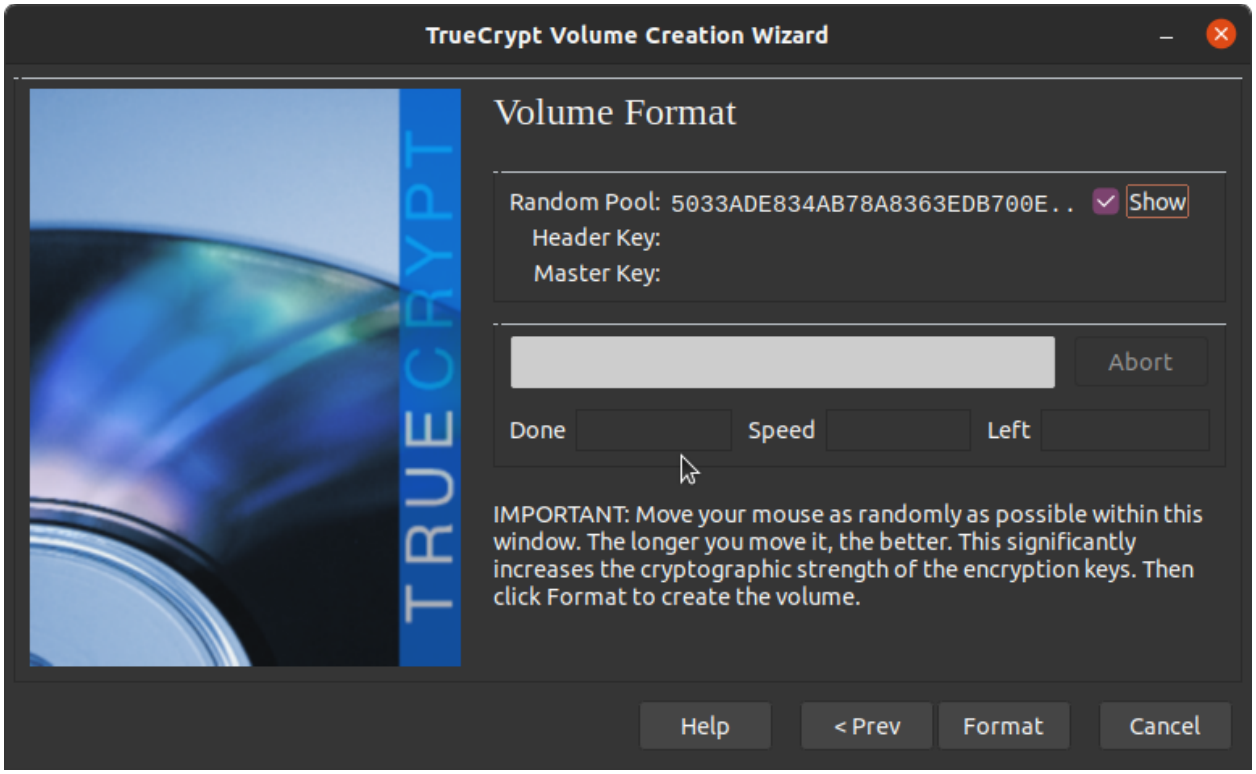

Specify a New TrueCrypt Volume
Name:
tc_deepnayak
Save in folder:
deep
TrueCrypt
Create Folder
Places
Search
Recently Used
deep
Desktop
File System
Data
Windows
Documents
Music
Pictures
Videos
Downloads
Name
Size
Modified
All Files
Cancel
Save



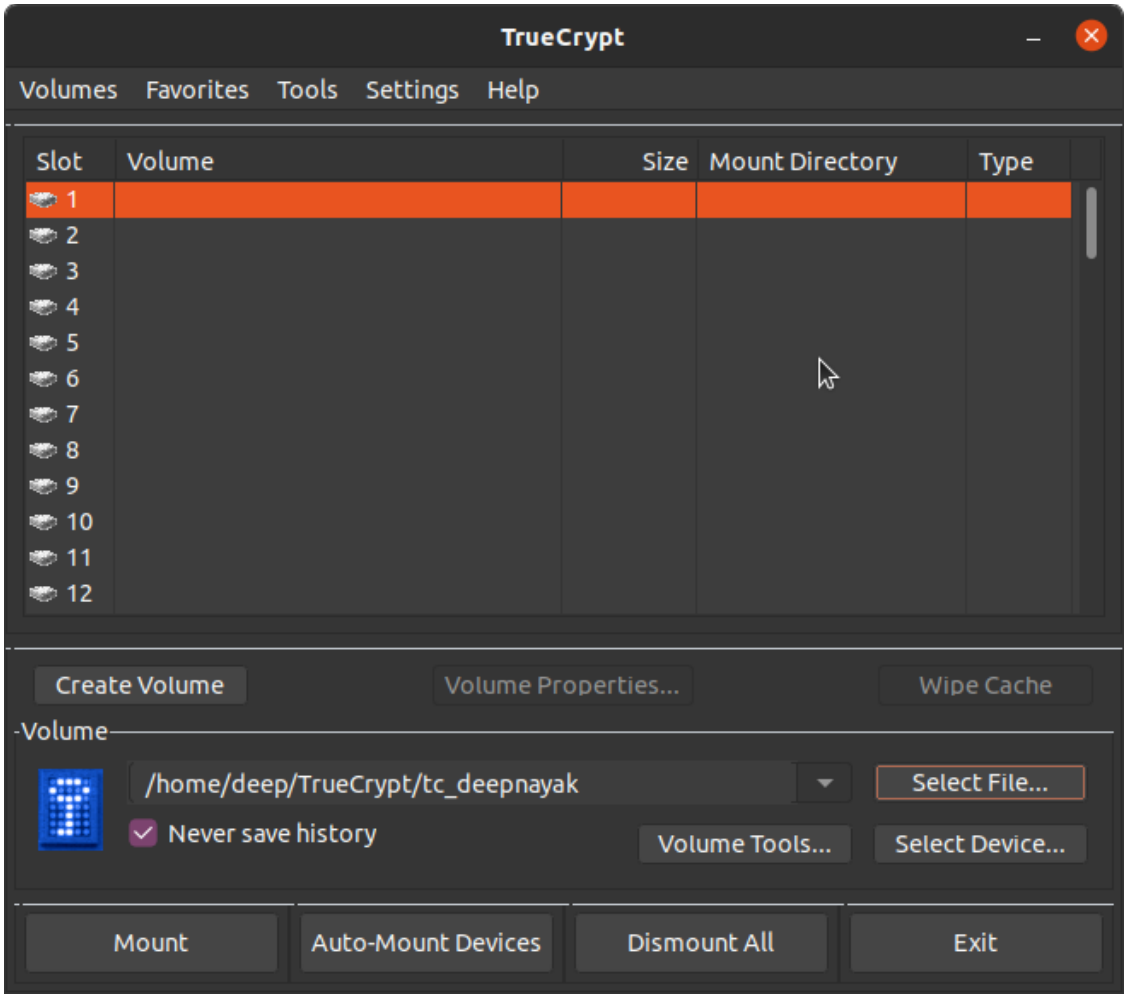
Algorithm	Encryption	Decryption	Mean
AES	4.4 GB/s	4.5 GB/s	4.4 GB/s
Twofish	970 MB/s	961 MB/s	966 MB/s
AES-Twofish	857 MB/s	922 MB/s	889 MB/s
Serpent-AES	528 MB/s	523 MB/s	526 MB/s
Serpent	307 MB/s	476 MB/s	391 MB/s
Twofish-Serpent	370 MB/s	312 MB/s	341 MB/s
Serpent-Twofish-AES	288 MB/s	357 MB/s	322 MB/s
AES-Twofish-Serpent	309 MB/s	327 MB/s	318 MB/s

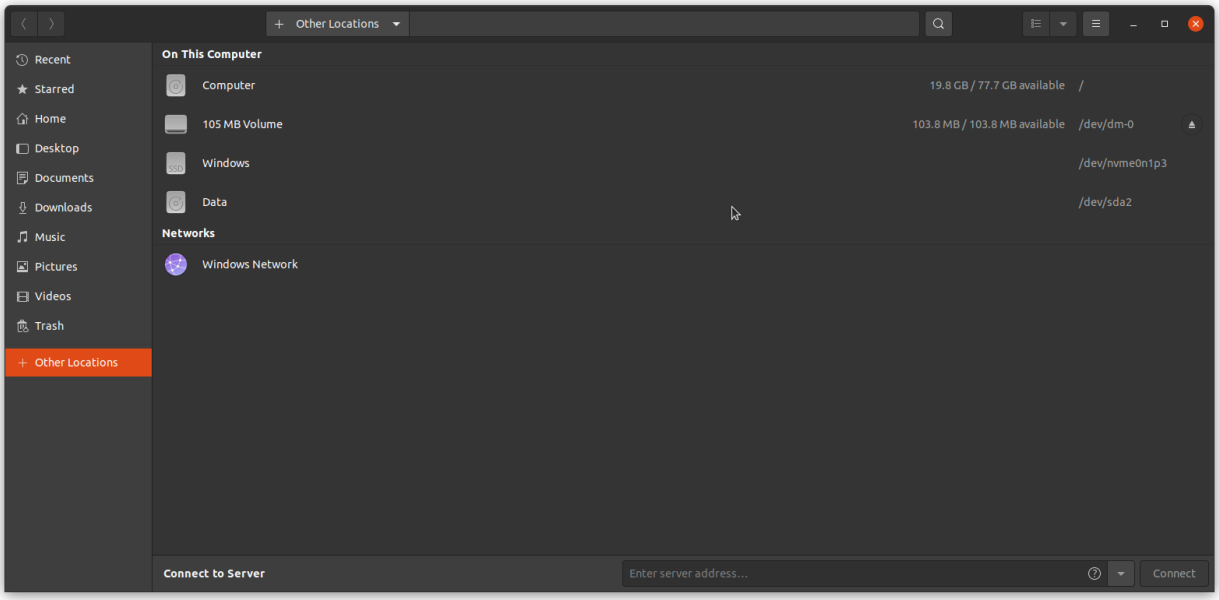

TrueCrypt Volume Creation Wizard
Volume Size
100
M
Free space available: 98.2 GB
Please specify the size of the container to create. Note that the minimum possible size of a volume is 292 KB.
TRUECRYPT
Help
< Prev
Next >
Cancel


TrueCrypt Volume Creation Wizard
Volume Password
Password:
Confirm password:
Display password
Use keyfiles
Keyfiles...
It is very important that you choose a good password. You should avoid choosing one that contains only a single word that can be found in a dictionary (or a combination of 2, 3, or 4 such words). It should not contain any names or dates of birth. It should not be easy to guess. A good password is a random combination of upper and lower case letters, numbers, and special characters, such as @ ^ = $ * + etc. We recommend choosing a password consisting of more than 20 characters (the longer, the better). The maximum possible length is 64 characters.
TRUECRYPT
Help
< Prev
Next >
Cancel


TrueCrypt Volume Creation Wizard
TRUECRYPT
Volume Format
Random Pool: 5033ADE834AB78A8363EDB700E..
Show
Header Key:
Master Key:
Abort
Done
Speed
Left
IMPORTANT: Move your mouse as randomly as possible within this window. The longer you move it, the better. This significantly increases the cryptographic strength of the encryption keys. Then click Format to create the volume.
Help
< Prev
Format
Cancel


TrueCrypt Volume Creation Wizard
TRUECRYPT
Volume Created
The TrueCrypt volume has been created and is ready for use. If you wish to create another TrueCrypt volume, click Next. Otherwise, click Exit.
Help
< Prev
Next >
Exit


TrueCrypt
Volumes
Favorites
Tools
Settings
Help
Slot
Volume
Size
Mount Directory
Type
1
2
3
4
5
6
7
8
9
10
11
12
Create Volume
Volume Properties...
Wipe Cache
Volume
/home/deep/TrueCrypt/tc_deepnayak
Select File...
Never save history
Volume Tools...
Select Device...
Mount
Auto-Mount Devices
Dismount All
Exit


Other Locations
Recent
Starred
Home
Desktop
Documents
Downloads
Music
Pictures
Videos
Trash
Other Locations
On This Computer
Computer
19.8 GB / 77.7 GB available
/
105 MB Volume
103.8 MB / 103.8 MB available
/dev/dm-0
Windows
/dev/nvme0n1p3
Data
/dev/sda2
Networks
Windows Network
Connect to Server
Enter server address...
Connect

H) Reflective Statements

In ECC, we use a 256-bit private key. This is used to generate the key for signing Bitcoin transactions. Do you think that a 256-bit key is large enough? If we use a cracker that performs 1 Tera keys per second, will someone be able to determine our private key?

Before Bitcoin became famous, Secp256k1 (which uses a 256-bit private key) was nearly never used, but it is currently gaining popularity owing to its various useful qualities. The majority of regularly used curves have a random structure, while secp256k1 was built in a non-random method that allows for extremely fast computing. As a consequence, if the implementation is suitably tuned, it can be up to 30% quicker than other curves. In addition, unlike popular NIST curves, secp256k1's constants were chosen in a predictable manner, reducing the chance that the curve's developer included a backdoor into the curve.

1 TB = 8e+12 keys, therefore if the cracker examines these many keys in one second and the private key size possibilities become 2^256 (about 1.2e+77 keys), breaking the 256 bit private key is not a difficult process; the key can be broken in under eight seconds.

Github Link:
https://github.com/deepnayak/CSS-Lab-Deep-Nayak/tree/master/Experiment%204