

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 (in ASCII format)**

[Encryption Home][Home]

A public key is exported into a binary format, but often we require to send it in an ASCII way. A common format is ASCII-armored format which exports to Base-64. You can paste your own public key here:

ASCII armored PGP Public Key Block (used exported *.asc file):

```
-----BEGIN PGP PUBLIC KEY BLOCK-----  
Version: GnuPG v2  
  
mQENBFTzi1ABCADIEWchOyqRQmU4AyQAMj2Pn68Sqd9lTPdPcItwo9LbTdv1YCFz  
w3qlp2R0RMp+Kpd192CIhduYHdmZfHZ3IWTBgo9+/Np9UJ6tNGocrsgq4xWz15  
4vX4jJRddC7QySSh9UxDpRWf9sgqEv1pah136r95ZuyjC1ExnoNxdlJtx8PlxCc  
hV/v4+kf0yzYh+HD34xP2bt1S07dkasYZ6ca7BHYi9k4xgEwxVvYtNjsPjTsQY5R  
cTayXveGafuxmhSauZK1B/2TFerijEt49Y+p07tPTLX7bhMBvbUvojtt/JeUKV6VK  
R82dm0d8seUhwOHYB0JL+S57PgFFsLo1NV5ABEBAAg0LkJpbGwgQnVjaGfuYW4g  
KE5vbmlUpIDx3lMj1Y2hhbmFuQG5hcGllci5hy51az6JATKEEwECACMFALTzi1AC  
GwMHcwkIBwMCAQYVCAIJCgsEFgIDAQIEAQIXgAAKCRDsAFZR6tdPQi13B/9KHeFb  
l1AxqbaFGRDExx8ufPnEww4FFqWhcr8RLwE8/C0lupB/5AS2zyvojmNFMGzUrB  
LGF/u1LVH0a+NHQu57u8Sv+g3bBthEPPh4bKaEzByRS/dYH0x3APFyIayfm78JVRF  
zdeT00f6PaxUTRx7iscCTkN8DUd3lq/465ZX5aH3HWFxFX500JSPr0/udqjoQuAr
```

Determine

Version:	4
User ID:	Bill Buchanan (None) <w.buchanan@napier.ac.uk>
Key Fingerprint(20 Bytes in hex):	d7d10cb24f38079377a25c0bcda158ff6f6aa48c
Key ID (8 bytes in hex):	cda158ff6f6aa48c
Public Key (MPIs in base64):	RSA CACzpJgZLK/sge2rMLURUQQ6l02UrS/GilGcofq3WPnDt5hEjarwMMwN65Pb0Dj0i7vnorhL+fdb/J8b8Q Tiyp7i03dZvhDahcQ58afvcjQtQstY8+K6kZFzQ0BgyOS5rHAKHNSPFq45MlnPo5aaDvP7s9mdMILITvLb CFhcLoC60qy+JoaHupJqHBqGc48/5NU4qb6fB1AQ/H4M+6og40ozohgkqb80HoxYbJV4sv4VYMLd+FK0 g2RdGeNMM/aWdqYo90qb/W2aHCCyXmhGHEEuok9jbc8cr/xrWL0gdwlwpad8RfQwyVU/VZ3Eg30seL4Sed Emw00cr15XDI6dpABEBAAE=

Public Key of three famous people :

● Sam Reed

[Encryption Home][Home] 

A public key is exported into a binary format, but often we require to send it in an ASCII way. A common format is ASCII-armored format which exports to Base-64. You can paste your own public key here:

ASCII armored PGP Public Key Block (used exported *.asc file):

```
A1ta01BZ3Pqj3Juruhwkj9mLUaqBCBlrNgStYrbUT7VpYETKxZ1t6Xx6Z/s0c+Gaf  
MR2BVQYM65bS0zA6g/Mx6+iaJRxLB3Q1yLIRTGRN2Cd6hBghkdnancxZ7Gzold  
juquuhk0ngLP8u547ADk/6QLWYdpRTeoS+OVS9sEfvoSpL504bX8D+7cBwn8Db+  
ORRi0uS3Njsq7UTDHCCiQLZCb6Vre05gaLv3JnZvZfChLuGaaBemjCkQKOKsNwX5  
5NCAAwUIAt3RZSSh2cD03SA5chSUQ2mAJkz9ttbfQ0X+0amaaaapDCoRdvTb8sOB  
hCmmQ77W8In5wlJKoDARzYfHcv/kyNvA8s0CD0qHrNHXHD01I5MHPaB3XYFPSu2p  
/hxowcKJPpkj3baYP+M3wG+DJOaEJmYksjfS+aVXAZNmrF80ivMpKbuKyHzhigB  
0of+MInAOhCkG7fNVt/1FOVfyfm4XC81i4EdhX01ML0zleXP7+Rl5Jq8Vw3uH4Qii  
Sh64Wq0/w6aEqZ1Z7rMNKLYRbecz9xNC5Q1MjG76ReaOFy/w3gDwJ/C5g/A7+sQ  
oQ/o3qyuyEuZCzaIVi3+Rc6KbHu5W6KISQQYEQIAQCQUTqX0/QIBDAAKCRCbabMQ  
nTu3sDRkJ0T07cLbNSwsscXkv+jyEYC0QQIsQCgrkPI+hdvkeFrQqM8caTt/64j  
KpU=  
=kbNd  
-----END PGP PUBLIC KEY BLOCK-----
```

Determine

Version:	4
User ID:	Sam Reed <reedy@wikimedia.org>
Key Fingerprint(20 Bytes in hex):	1a24253c8ba01e44a8cb470e3bbb95ce2b08bfd2
Key ID (8 bytes in hex):	3bbb95ce2b08bfd2
Public Key (MPIs in base64):	ELGAMAL HaUU3jrPjlUvbBH7zrD5ed0G1/A/u3AcJ/A2/jkUYjrk7zY7Ku1EwxwgokC2Qm+la3q0YGi79yZ2b2Xw os7hmmgXpowpEcjirDcF+=TXAAMFCACNGWUkodnAzt0g0XIulENpgOSZM/bbW30DL/tGpmmmqQwqExb0 2/LDgYQppk0+1vCJ+cJSsqAwEc2Hx3L/5MjbwPLEAg9Kh6zR1xwtsOTBz2gd12BT0rtqf4caMHciT6S pi922mD/jN8BvgTmhCZmCrI30vmLwGTZkX/DorzkSm7ish84YhgdKH/jCJwDoQpbu3zbfb9RTlcnu5u FwvNYuBHYVzpTCzs5Xlz+/KZeSavFcN7h+EiokoeuFoEP80mhKmYme6zDvyi8kw3nfcTQuUIzIxu+kX mjhcv8N4A8I/wuYPw0/rEKEP6N6srshLmQs2iFYt/kX0imx7uVui

Key Encryption Algorithm : ELGAMAL

● BRIAN WOLFF


Public Key (in ASCII format)

 [Encryption Home] [Home]

A public key is exported into a binary format, but often we require to send it in an ASCII way. A common format is ASCII-armored format which exports to Base-64. You can paste your own public key here:

ASCII armored PGP Public Key Block (used exported *.asc file):

```
-----BEGIN PGP PUBLIC KEY BLOCK-----
W7iisfrScf2pABEBAAGJA1UEGAECAA8FA1VepPwCGwwFCQ1mAYAACgkQNh+U0xA
jdQz7g//V3kN05EyY5/12ocT6m81cB/DG3kEhRsXPU7C1a35ASo8gx0S19SreV
FIED9MjDe6MeaSFkiG6Yz8wGH0o7LwnPhQ3SQx09qKcThpCny88f0xn1RvBUrZK
BcRwzDt6Z59Zw2upht8XR6dwDZ6Kc0x00CpJL/Mhbr4LKz4YMUh0nE+XifM76ZWB
LPKZxYTYYSVhZhJeRkyCRu+5+JFPumikD2bKBvFV2Kaz48WV3wT2+INMzfbR4Jo
KyVbwIZ3cI6LISdJBlgxfB4AibMrwdRZfkX+rUuepUf2xkBGFkypvzBZecpTtH
jmlDsN0nd5nablTtG4LBMPY3Kkm67dMQfgahZgtE5iKvnQns4v0191kpHa5UzA8
1KwNn0It8B/R1VeBco5aZbx88m1NW/XlfIqwpUvM/S+nvF1oD1FwprmF4y02StXy
NSmseW99+a+drsf3CmT64qPc0dmtxNu530mHWJ5vn/1cvB5Q0wPMVKUF7+NJ1fZT
0ct9upP245gg1923r+AsSm2al+KYxodctxNlx/QeaE2d7micxyw8GjssqCsgrRB
3+v/gekJJfAJ00r0IpLdDuNqq9s6TQQK+tHDnsSOPsmoAWhr/Llo7Mp2/b2lUpq7
jxv2JFBfHDc3Nzyjvrwx3Rgizk12pApbXgZ8GguEcCy+mysKoho=
=Siwx
-----END PGP PUBLIC KEY BLOCK-----
```

<input type="button" value="Determine"/>	
Version:	4
User ID:	Brian Wolff (Bawolff) <bawolff@gmail.com>
Key Fingerprint(20 Bytes in hex):	66e16240db737b60c42ce1a1bf1629cd074d3dd8
Key ID (8 bytes in hex):	bf1629cd074d3dd8
Public Key (MPIs in base64):	<input type="radio" value="RSA"/> RSA fVy2dga0YSBwCLSzMmSnHGFWsYbAnnJhkAUoCkm20dj7vT+/8hPkw9qbYl01JS2QedDVpCWF5YkwhM saCgxIZWNTj8ZLlicQ3div1+5JYEVaTBILkb0ZE8HVzpLQ1X+Re2rz1WC4NnqDXsZ1JtZcDwV7vIvHexv fdZRz/5Xr0gvgvwCosguKh8bsZcKQqI97n8dxV2KYIlFppGTB1NQXc4607+5a2aoYxr2dsTVfLozFDji hQiQNeYjsGY0uEnWmu3h3ZinFF0RKRxfxhYkDII69d03hzJNbZ6tnuCXgEkh0ktgcIztD12tZEbjJlJ+ JIw4tKBuxKwYEDWfqd3CpVTZg124/5baw8LbygQAAXKwTM5zwC99gLhnaCUNG8qq1ReabylpxPI5yhEtNWEfI2a2LyI60LK/Nfv04hwG7e/gDdWaVu4tbH60gn9qQARAQAB

Key Encryption Algorithm : RSA

● TYLER CIPRIANI

[Encryption Home][Home]



A public key is exported into a binary format, but often we require to send it in an ASCII way. A common format is ASCII-armored format which exports to Base-64. You can paste your own public key here:

ASCII armored PGP Public Key Block (used exported *.asc file):

```
V/Q+TQUJCLGCuwAKCRD22tKFAY+sAs7SD/9BucBcVwdx3B1pNX/E4ooCyMuhSsw
t/4RNP8zbAynjpuGV8uT8WY4Xn0Jp532sNv1Lp7P1sADFbn703Ntr0P1Twhrd0S
fIVh4kvAkH1zcAzt20j+RKId39kzVcPld2T7k1Aurwloimr4Kcgnus43Gt0lQuxs
uizx3LUjSpfl6wtAwd9lihdz7hJq68ElRjkTMGlbtrXpT7Nk+xQAkUS1dtUrPDW+
QGgH8Zko8fPBukIcvuhn/D7Q9PJNN6RixoeoLfnjX7SopyCm9vnyI/9qdV3lK1I
nZRWYoHejNOiYKOYC1trnh0676GvvVPfqKZCj67psvVaraVlaLZ9ZnctNIQopZ7D5
cMu8XmcLp0gs8ac3E5HiEyrCxwAn56SvufQt35KXU/tzLIdgz166EommnXjJW/Tv
+d/356vZrobGMGbkmr0ET8T0yIppPqvEUxtCDVXvai2qyNR9ktvfb77c9gr0+cXQ
gzvwfs+fk9nAZRhpiqYeHocziIqzKdnkRjBsIHUr xl9dak/C5okFNc+Kbn06R+q2
v2cLpcY1eDksAilp/jSOkanuAIbP8ROGZuBLp9aYmbcorfZB65/x1vtajMGow
XMls1aEspAx+fYtvZizUb7Wn9hk0VyxjZasA4KfS/nHgrSKub1lgP/yEZ6dQbjH
g+D18C1pfqRnTA==
=Q8bo
-----END PGP PUBLIC KEY BLOCK-----
```

Determine	
Version:	4
User ID:	Tyler Cipriani <tyler@tylercipriani.com>
Key Fingerprint(20 Bytes in hex):	6237d8d3ecc1ae918729296ff6dad285018fac02
Key ID (8 bytes in hex):	f6dad285018fac02
Public Key (MPIs in base64):	RSA EACziAgNwZ/KfSuSSFSqWDYAlpvBjZyiZ6nc9kiC+fAJt53awZhrkUhvwHFYc5+G+VwpfOvbBTqjPHe sk5CDuNQTdsGXIoQRr/nIOG9X4XCy6TXjHcyffid4S5vQupxV+oj2a5eRI0PBsN6d/ZhVzyDD2zYKUs oAamXA6Ns+aq22KGCUQJDA74D9wQU84nfXsbcrz17Dnw12yDvIyo0iv4je4zJK6cZziUzY3dtJI+xRP j4Is6sygmt+5GvQ1MM1xkVxzMz5XcuujVedognTOSr/uoNgHxatagvwSPeBbC0LgdvAAnqvGoAvrDc2e cefG3VmEMJXRMNqkimpj0gBiJaBIqwccOKjf5p7x18vLM5kbVDZKb+13VcsbycQyXb9LiAx+QD2VxfhKL Mplk5YP21Nfp48DWWrSa7T6Nr8W7mNtmy99Ci3UbvVTODiXpIcztV+Ne/IPdAPYqieZ4Xk8wbbZaZC7R

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: <ul style="list-style-type: none">• Modulus• Public Exponent• Private Exponent

		<ul style="list-style-type: none"> • Prime1 • Prime2 • Exponent1 • Exponent2 • Coefficient <p>Which number format is used to display the information on the attributes: Hexadecimal</p>
B.4	Let's now secure the encrypted key with 3-DES: openssl rsa -in private.pem -des3 -out key3des.pem	<p>Why should you have a password on the usage of your private key?</p> <p>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</p>
B.5	Next we will export the public key: openssl rsa -in private.pem -out public.pem -outform PEM -pubout	<p>What does the header and footer of the file identify?</p> <p>It represents that the key stored in this file is the public key.</p>
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	<p>What are the contents of decrypted.txt ?</p> <p>Can be seen in the screenshot below.</p>

B1

```
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-----
MIICXgIBAAKBgQDUKRLrU04Dttxdzc3WeXefhgqXurWUlP53aZk/18//+469seR1
FjfLSprh/txk9tmmWaIojYWLMQgPDW5w7nInWCD9Q5X4sGihd7vDtd33tyIuyHld
OfuHT0go94SYyFLxliw0EVE90/v4TeusQwVrLZLV0dWlChq+Up8UugfaXwIDAQAB
AoGBALgAs4bjJgs095cJ4ha8HFJ/R1qWtPscsm0waZgkT0rg8uR0j4Y48cTffHq6
tlzedp++p1PygdFmN+40IRXSx67gF6TcKhyh72C6CRBKdB02jWTf0JSUZE+qJKEd
COKSzIgV2cIK50F6maWPHN0zPzIX5wucvDpVYnyKgdqSDLbZAKEA/dW+RpPPx4my
U0FeyeZVQ0cnPRxFzdF3Pm7jwKh2n8TqCuc615hat5xd4e0PL9DXtk3NC9Xfe6ej
q0byucQztQJBANX4VULEjEjavhRAVjtmozkLdojGvSZeZKHUytFy77qnSNxhcYg0
MTnXbhmGz36A4WNm8TNkR2Ex2x0D6L9MSkMCQQDz/0NhKnKfVxBQkVThdQReoIXq
a0P6PTCHlizdqlIk8C8dffoFBhGLz6w7F/gquXCxcAX7K4Igphen2mRAjXFAKAJ
l5pq5wMu+09whPnYq+Fo00VTGGW+ZJjqUFnPTfHFUL6H0q5+Rqi2SRwcdNi0eR6h
kCnb/fDdQJjrPw4Fz1AkEAgnWMkLSJidE/r+ZeDwYrmx024qbv30JhaExzAmrI
wdCJD1nPwm0FTR2c6CSzRgb78hB13qvVVVlNdw4SaQU8Sg==
-----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
```

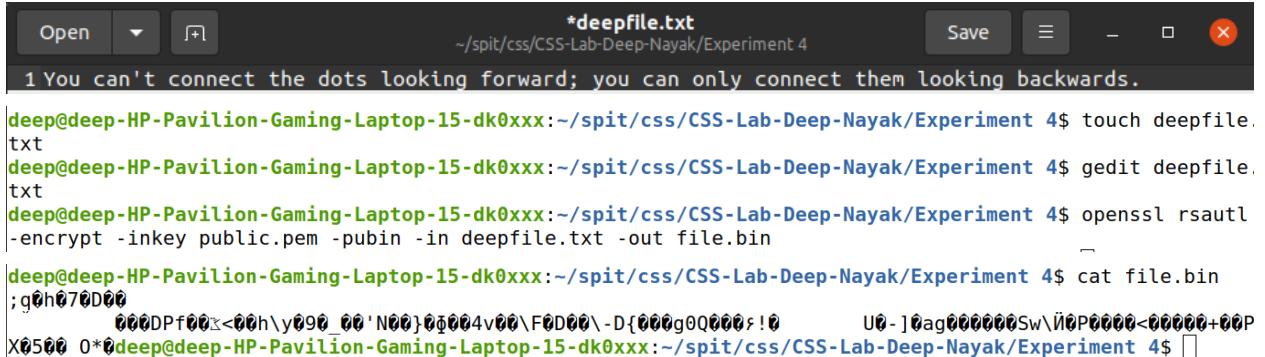
B4

```
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
-----BEGIN PUBLIC KEY-----
MIIFMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQDUKRLrU04DtZxdzc3WeXefhgqX
urWUlP53aZk/18+/469seR1FjfLSprh/tXk9tmmWaIojYWLmQgPDW5w7nInWCd9
Q5X4sGihd7vtd33tyIuyHldofUHT0go94SYyFLx1iw0EVE90/v4TeusQwVrLzLV
0dWiChq+Up8UugfaXwIDAQAB
-----END PUBLIC KEY-----
```

B6



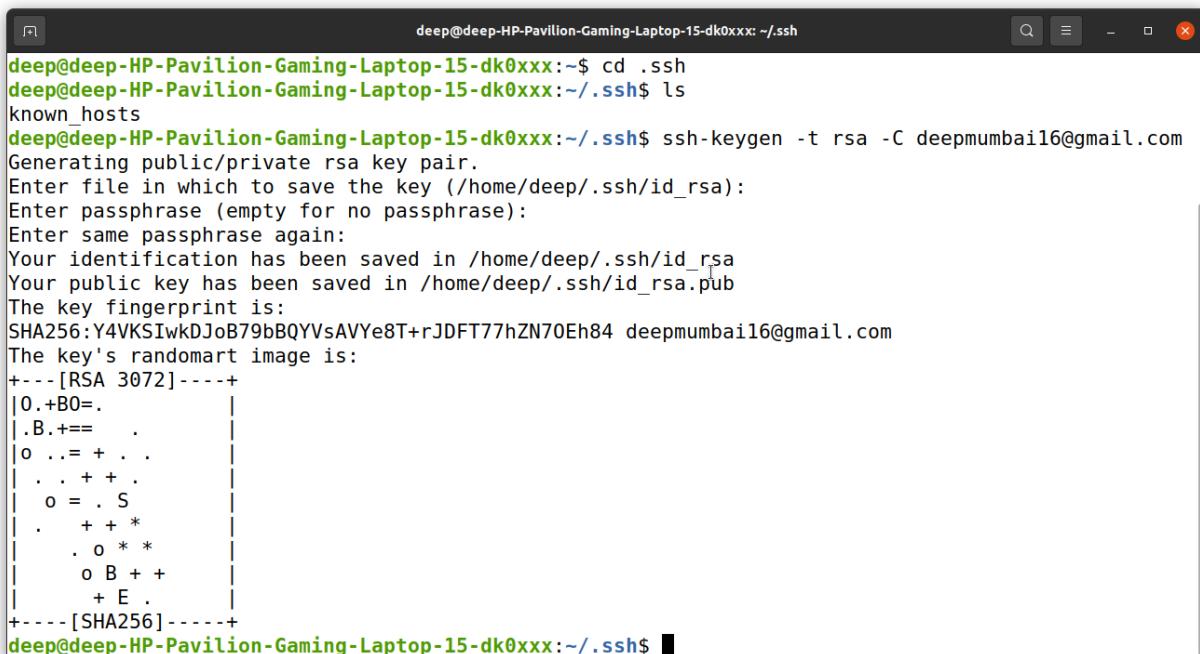
```
*deepfile.txt
~/spit/css/CSS-Lab-Deep-Nayak/Experiment 4
1 You can't connect the dots looking forward; you can only connect them looking backwards.

deep@deep-HP-Pavilion-Gaming-Laptop-15-dk0xxx:~/spit/css/CSS-Lab-Deep-Nayak/Experiment 4$ 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
;gfhf70D00
    000DPf00<00h\y090_00'N00}0f004v00\f0D00\.-D{000g00000!0      U0-]0ag000000Sw\00P0000<00000+00P
X0500 0*0deep@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 encrypt -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
deep@deep-HP-Pavilion-Gaming-Laptop-15-dk0xxx:~$ cd .ssh
deep@deep-HP-Pavilion-Gaming-Laptop-15-dk0xxx:~/.ssh$ ls
known_hosts
deep@deep-HP-Pavilion-Gaming-Laptop-15-dk0xxx:~/.ssh$ ssh-keygen -t rsa -C deepmumbai16@gmail.com
Generating public/private rsa key pair.
Enter file in which to save the key (/home/deep/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/deep/.ssh/id_rsa
Your public key has been saved in /home/deep/.ssh/id_rsa.pub
The key fingerprint is:
SHA256:Y4VKS1wkDjoB79bBQYVsAVYe8T+rJDFT77hZN70Eh84 deepmumbai16@gmail.com
The key's randomart image is:
-----[RSA 3072]----+
|0.+B0=.
|.B.+==
|o ..= + ..
| .. + +
| o = . S
| . + + *
| . o * *
| o B + +
| + E .
-----[SHA256]----+
deep@deep-HP-Pavilion-Gaming-Laptop-15-dk0xxx:~/.ssh$
```

```
deep@deep-HP-Pavilion-Gaming-Laptop-15-dk0xxx: ~/ssh
Enter same passphrase again:
Your identification has been saved in /home/deep/.ssh/id_rsa
Your public key has been saved in /home/deep/.ssh/id_rsa.pub
The key fingerprint is:
SHA256:Y4VKS1wkDJoB79bBQYVsAVYe8T+rJDFT77hZN70Eh84 deepmumbai16@gmail.com
The key's randomart image is:
+---[RSA 3072]---+
|0.+B0=.
|.B.== .
|o .. = + ..
| . . + +
| o = . S
| . + + *
| . o * *
| o B +
| + E .
+---[SHA256]---+
deep@deep-HP-Pavilion-Gaming-Laptop-15-dk0xxx:~/ssh$ cat id_rsa.pub
ssh-rsa AAAAB3NzaC1yc2EAAAQABAAQgQCKBac+doL68oRMrljACHgBOTkEBHmLlHED+bH2zkwJtx6YkW/+T/EgIHCTzr
xrTJOH1g/SlieEx750jhzIgJE5HsRtoezLvk/4seY7yzx/3uQZlaac8mnLnL0jFXQSUXKxOB8g7Q4lew0MIAZ9mSvq9NFw++VY
3dgx2lvi3v9jZ6nxXuVcSVh9MwEd9FuIFnEqo/e5aqM6Zj+GGbxVK6DdLu/pnoS+LwZtAhFWYlhxE0JBnEwPWyX/q4lwcmQFK/
UGiXR0SBS5wZeZZID//u6L20u4szY0cGmrntreCEypHH0rukA68pB3TgWYsceiMHlQ6owAaW0kzZiJJ1vq/9ecungZQnwR0ic+
/w48bkrDBtfSygJ0uWkyNKsxbDaqy2oDyktRXvv24Ytrt/K4UIL9cIE5l9u9qb23iF3NV2r55meKih6zO7AM0IADp/SCavTvkQ
2kFCIBkkLYxJAY64rf0aqxqMNpnnEY6A8PCVWGiuas2FDX6nnwJI+rnP0L4FE= deepmumbai16@gmail.com
deep@deep-HP-Pavilion-Gaming-Laptop-15-dk0xxx:~/ssh$
```

```
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

SSH keys / Add new

Title

Key

```
ssh-rsa
AAAAB3NzaC1yc2EAAAQABAAQgQCKBac+doL68oRMrljACHgBOTkEBHmLlHED+bH2zkwJtx6YkW/+T/Eg
IHCTzrTJOH1g/SlieEx750jhzIgJE5HsRtoezLvk/4seY7yzx/3uQZlaac8mnLnL0jFXQSUXKxOB8g7Q4lew0MIAZ9
mSvq9NFw++VY3dgx2lvi3v9jZ6nxXuVcSVh9MwEd9FuIFnEqo/e5aqM6Zj+GGbxVK6DdLu/pnoS+LwZtAhFWYlh
E0JBnEwPWyX/q4lwcmQFK/UGiXR0SBS5wZeZZID//u6L20u4szY0cGmrntreCEypHH0rukA68pB3TgWYsceiMH
lQ6owAaW0kzZiJJ1vq/9ecungZQnwR0ic+/w48bkrDBtfSygJ0uWkyNKsxbDaqy2oDyktRXvv24Ytrt/K4UIL9cIE5l9u
9qb23iF3NV2r55meKih6zO7AM0IADp/SCavTvkQ2kFCIBkkLYxJAY64rf0aqxqMNpnnEY6A8PCVWGiuas2FDX6nn
wJI+rnP0L4FE= deepmumbai16@gmail.com
```

Add SSH key

```

deep@deep-HP-Pavilion-Gaming-Laptop-15-dk0xxx:~/test$ git clone git@github.com:deepnayak/CSS-Lab-Deep-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:p2QAMXNIC1TJYWeI0ttrVc98/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: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

[Encryption Home][Home]

This page performs Elliptic Curve. The following example create a public and a private key for Bob and Alice. It also create a shared key (ECDH - Elliptic Curve Diffie Hellman) which can be used for create a secure tunnel:



Message	Hello. Alice
Determine	
Key	deep

```
++++Keys++++
Bob's private key: 01158b98efb80fed6371ceed6da0a7d93173aa5423098ffba667787e31338269161dc626
Bob's public key:
04062527a27aed781c4ec80419d070db6013e9ded32ab50801e963b9003535d4c1806790df0232554bbbda244aa7d7705ff547d554e2e6bce61f24
13e1306705f40c90a08137849645

Alices's private key: 018dcaaf5d97c7aa843517cd066809d12a4e99ad2ddab6eaaaf6f5b9d9f1e3438ac58b54
Alices's public key:
0402f3fce353be06a7589d699a432e558c30735d6fdd133ffd85826a0289f49b644d36e63605af5c6194a5819859663eb0cbb9aed1d6b19dda7e1
f0c70f153d7deccc6e798c98be00

++++Encryption++++
Cipher:
df29ae6b6ba0b63d775420a675890c3c040055b10a2d4b03b41d35c41d7f3c9f5b42b857a126851190506183f0cceba803a33c1a5987
893f23e23e535d25a20c2e0511e4228bdbc9c8af1557aba5b6779c8810a18cbc11607f986b7977f0ffe4d4f9a3275181270613f372713fc7624aa9
667be8685abc1c15d81dc8951a56cfb1987a9c
Decrypt: Hello. Alice

Bob verified: True

++++ECDH++++
Alice:010d8dcb12737bd982dfc5c270df8c4461a0c0b79ecc7cdbac7cc6f60a9ffbeb
Bob: 010d8dcb12737bd982dfc5c270df8c4461a0c0b79ecc7cdbac7cc6f60a9ffbeb
```

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 $y^2 = x^3 + 7$, 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

[Encryption Home][Home]



For a finite field elliptic curve we can search for the first 20 points for a curve of $y^2 = x^3 + ax + b$ and for a defined prime number (p) [Improved version]
Note: this version will not find all the ECC points.

Parameters

a: 0 A: 0
b: 7 B: 7
Prime: 89 Prime number: 89
Elliptic curve is: $y^2=x^3+7$

Finding the first 20 points

(14, 9) (15, 0) (16, 3) (17, 5) (22, 8) (24, 6) (40, 4) (60, 2) (70, 1) (71, 7) (103, 9) (104, 0)
(105, 3) (106, 5) (111, 8) (113, 6) (129, 4) (149, 2) (159, 1) (160, 7)

Determine

Examples

The following are some examples:

- $y^2 = x^3 + 7, p=23$. Try!
- $y^2 = x^3 + 7, p=101$. Try!
- $y^2 = x^3 + 7, p=802283$. Try!

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 > ...
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=NIST192p
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
ment 4/ecc.py"
Message:      b'Bob'
Type:        NIST192p
=====
Signature:    b'PwIGk6TY8n8sWKZJaHtdADkkqER9vnhIw5zt0suXymCT8loxtFCzfSMb9yurXwis'
=====
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 > ...
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=NIST521p
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/spit/css/CSS-Lab-Deep-Nayak/Experi
ment 4/ecc.py"
Message:      b'Bob'
Type:        NIST521p
=====
Signature:    b'ANu89odN2nq9vb7ffizBzPcNGT34sLkZRJchF+pEq00Xjw7QV/pK1Ytxcr1g0LSu5C1jzSB2xQigBtJBY6dJRYU1kAQcfL2s30cw2ch7RtD0uT3hv2B
Ax0jVgGmb0ba++TydEc6Fz1pUY9I2bnI3iddoh6YvXPioJyAfm02hRJgDFK9+'
=====
Signatures match:      True
deep@deep-HP-Pavilion-Gaming-Laptop-15-dk0xxx:~/spit/css/CSS-Lab-Deep-Nayak/Experiment 4$
```

SECP256k1:

The screenshot shows a Jupyter Notebook interface. The code cell contains Python code for generating a digital signature using the SECP256k1 curve. The code imports ecdsa, base64, and sys, then defines a message 'msg' as 'Bob', sets 'type' to 1, and uses 'cur=SECP256k1'. It generates a signing key 'sk', gets its verifying key 'vk', and signs the message 'msg'. The output cell shows the command run in a terminal: `/bin/python3 "/home/deepment 4/ecc.py"`. The output itself is as follows:

```
deep@deep-HP-Pavilion-Gaming-Laptop-15-dk0xxx:~/spit/css/CSS-Lab-Deep-Nayak/Experiment 4$ /bin/python3 "/home/deepment 4/ecc.py"
Message:      b'Bob'
Type:        SECP256k1
=====
Signature:   b'3ozE5xtRVp5JFKN0FWAfXAsUx8e0Vse0Nhxr9W39A0Vt5HyxajpD77T3tyufTNWmVH/Kc0YFkIDzDNPx1NYtw=='
=====
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

$$C = 4^e \pmod{N} = 469$$

After decrypting the above ciphertext

$$411 = C^d \pmod{N} = 469^{99} \pmod{22}$$

```
rsa.py  U  X
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 ×
rsa.py > ...
1  p=17
2  q=23
3  N=p*q
4
5  PHI=(p-1)*(q-1)
6  e=3
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"
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 ×

```
rsa.py > ...
1  p=7
2  q=23
3  N=p*q
4
5  PHI=(p-1)*(q-1)
6  e=5
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"
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 ×
rsa.py > ...
1  p=19
2  q=13
3  N=p*q
4
5  PHI=(p-1)*(q-1)
6  e=5
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"
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 $(d \cdot e) \pmod{\text{PHI}} = 1$. But how do we use code to determine d

```

inverse.py  U ×
inverse.py > ...
35     # Either n is 0, or p is not a prime number.
36     raise ValueError(
37     '{} has no multiplicative inverse '
38     'modulo {}'.format(n, p))
39 else:
40     return x % p
41
42 val1=53
43 val2=120
44
45 print("Inverse of ",val1," mod ",val2)
46 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 X
inverse.py > ...
35     # Either n is 0, or p is not a prime number.
36     raise ValueError(
37         '{} has no multiplicative inverse'.format(n, p))
38     else:
39         return x % p
40
41
42 val1=65537
43 val2=1034776851837418226012406113933120080
44
45 print("Inverse of ",val1," mod ",val2)
46 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 X
e3.py > ...
1 from Crypto.PublicKey import RSA
2
3 key = RSA.generate(2048)
4
5 binPrivKey = key.exportKey('PEM')
6
7 binPubKey = key.publickey().exportKey('PEM')
8
9 print(binPrivKey)
10 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"
-----BEGIN RSA PRIVATE KEY-----\nMIIEpQIBAAQCAQE57FwUcq0u42AdpVvFQ/s+5ACd0FYPyGC9Zjfm0T9sXkgk89s\nJwpo2wp0N9IhmU5924uj8Firgxia2p7f5P+vsVp5hiJ4/FXMRYI17ndv+2MoafVj\nlNLXk9x4+XsuPmk+iKaeKrM7APT+H2Ph91FwJz2htlxhDTqziHydTRxe0MsnbHePHV\nnlWqLJ3X0oUckvbJ0pznHFdzm1i4W0A7XiwMgBvJuYWrHqs1zxhiWEHhSMxiUYD+\nntML0GnQkGrN/jfv+TIBE1W+K4o2r04XQFI/\nneE4W0p+9qatwOoyzzVnxT6xLk0\nnTifNbpHJa1CYXVkwqn1A1B2Mn1TbgsDdIXQIDAQABAoIBAA7RsNEuc5g79P\\n3W+/r8TeArV3FD2f
d1dkUOTI43XJUAG/5AksI24BrRuToNx6HP9emgxkGRkJAE\\nouXq+7I0ix7K9Cmr1Enq20e6ghFYB4T1csU9LhdpsyG0qCbul6o5pHdBG0Dn0t\\nPuJBC0wUhgeT2nIEYNGLW+nK1I+f7yw0
m8rKKYYxFg2v2FE/Mbrhu1qCbGYW3Nn\\n4Zr9YK1Z7juAM0E4pelP01qRfrg6Qx60qtQR1XFTMsQtd4xe7YVgdl6KyLra1T0M\\nPtwEdksrAbaNFaXgX3ft2kTxdfrWrxYJwmwv2WNVmzAqeoFl9Wc
8fGbd48T0xTrVnhKT8BWECgYEAt7IAiMv71dfps3E2z+ylsY\\nTUMQ81D9phqjEiCqicGxpbusn2\\nh0lxrb9teg0CJ81+mmTztG0w9KGNLRLUwFsR04FdS01cTN9elLdxt4c1SKejK4\\nX
Pzr5duanSMyIA7qTTnp2/hu0vIt688wEtYdi8jkaQpSMCADj4WeCgYE+A+Rde\\n0Wtxj9/d8u09HIRuoP7RoyCp47qcj1zsPpyik8ghu1Sm9brYhA25K3aca8gwqA\\n7EGIwII\\rm4x0RNp
16tkwnJzvwsZRpM1WSV06oZ3019xHk5zP2wDcx49VM\\n8JnjCu0vCyq8RYPyTewkjH07+4lh263h1mZbvH0CgYEAE5EkG6BWUk00VG/H0uTa\\nqpuEKc50m45h4D0UjY65c26rW00UB+glw9m9
J0djPsXhW4K01BDxsAlku01475j\\nqeNRDZgKU0hgicu708RkjwZnILx3E5jyZEKwZYgmzZCDsk+IuEcUD3H2nmDZLR\\ndoF3J2v2tlk8jvtNw1TuzbwCgjYEAuzfpYUvscmW\\nnoxeLE3s38
40AXu2SiSLNW\\nt07b1lnrzJavJovnjbQePeRNBIChAKWPjHwlgGuNCQk9qZ24fJWTF1ZCo3s9J9h\\nlLcfUeCmTtg8P\\av0jPhtexHTB8R57nF0grYLymsqUwTBWycgkPQs0IBr52zb0\\njuo
YuCuCgYEAmx7BprE4+zw0e+92m0QuyPoih6j0j0o0ticyfH8y19k4n0nf1+0i\\n6q+Pn3t2LaCk01cyjZ2TsP5T6LLFh+kAPx4t3x49h+9Kw63Eu85fEUoH6ZhnV1WJ\\n5ljk5Qp+Pv1VfpWo5Vpj
ptDob2n9db2H2ZnnjFPfnVpsJ3TMzR5aQi=\\n-----END RSA PRIVATE KEY-----'
-----BEGIN PUBLIC KEY-----\\nMIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIBCgKCAQE57FwUcq0u42AdpVvFQ/s\\n+5ACd0FYPyGC9Zjfm0T9sXkgk89s\\nJwpo2wp0N9IhmU5924uj8Firgxia2p7f5P+v\\nsVp5hiJ4/FXMRYI17ndv+2MoafVj\\nLXk9x4+XsuPmk+iKaeKrM7APT+H2Ph91FwJz2htlxhDTqziHydTRxe0MsnbHePHV\\nlWqLJ3X0oUckvbJ0pznHFdzm1i4W0A7XiwM\\ngBvJuYWrHqs1zxhiWEHhSMxiUYD+t\\nML0GnQkGrN/jfv+TIBE1W+K4o2r04XQFI/\\nNneE4W0p+9qatwOoyzzVnxT6xLk0\\nTifNbpHJa1CYXVkwqn1A1B2Mn1TbgsDdI\\nXQIDAQAB\\n-----END PU
BLIC KEY-----'
deep@deep-HP-Pavilion-Gaming-Laptop-15-dk0xxx:~/spit/css/CSS-Lab-Deep-Nayak/Experiment 4$ █

```

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

[Encryption Home][Home]

PGP encryption allows for the encryption of email and with a signature from the sender. You can copy and paste your public key here [link]:



Name: Deep Nayak
Email: deep.nayak@spit.ac.in

Determine

Note: We use 512-bit RSA keys in this example.

```
-----BEGIN PGP PUBLIC KEY BLOCK-----
Version: OpenPGP.js v4.5.2
Comment: https://openpgpjs.org

xk0EYa34wAECApjANRTcbC94S/xqSv4FT+j11MgJY8z7sm08ag7TURpEwlz
xE5gWMgraeb3BqC/cw+ErnbDDZ3KFo7t355ehstExcAEQEAac0YmlsbCA8Ymls
bEBob21llmNvbT7CdQQQAQgAHwUCYy34wAYLCQcIAwIEFQgKAgMWAECGQEC
GwMCQgEACgkQKwA5Hk3S5K+m8gh9GOKD6IsM2rEpgEfDdK16S7+Xp9XYu4eT
GmXZB1DbtzE7ps3/FoN2G6NqexaosvW87eLa/h2bVpJqp/2Lh05tc5NBGGt
+MABAgCL6ucnKacMjGzhytt4kpHF1IFTzY+c0oioXwaarNVvuuiIgwlv2F
KIGXK3E0sELjoGRz/yuZpbys+Duqz0dtABEBAAHCxwQYAQgACQUCYa34wAIB
DAAKCRarADkeTdlkr90kAgCaWKBrdYgepkZ7Hu71GT19kgTH+2//jvn4ro
ZBqAstCFe5xeUC8GE/KX3Y3t7N4F/vUrgTvSYTru8FaQAEou
=D1CY
-----END PGP PUBLIC KEY BLOCK-----

-----BEGIN PGP PRIVATE KEY BLOCK-----
Version: OpenPGP.js v4.5.2
Comment: https://openpgpjs.org

xcBmBGgt+MABAgD14wDUUSGwt0Ev8akr+BU/o9dTICwPM+7jjvGoOyFEaRMJ
c8R0aljBq2ntwagv3MPhJ8AQ2dyhaLe9+eXoLbLV3ABEBAAH+CMiETTwavBa
H+XgMuBcOheotSbMW65TiaCuHySm44Y9w6CXD/WuQq2xG0L+4gL7P8wfZ9F7
ZjXNU/jGuHJRnRSAU8tR8gvnZ6wG1Rg5By1nB/EUsndFBAM1bh2NKqHzkR
wzWjLn/MVU1edP7IKT8Io9jDzCHWFus5r+p2i1lOKTx78w1CnhabDRWgSFs
r6Cjq7z1Ut12gboJePOWQIf3dufozB/WB15TfReCsT8spXgRckaR87AnAO
1AWppejeIFDUjzmv840dgT1JF1Fj5jK3aQaqzRR1aWxsIDx1aWxsQGhvbWUu
Y29tPsJ1BABCACAFBQJhrrfjABgsJBwgDagQVCaoCAXYCQIZAQIBaWTeAQAK
CRArADkeTdlkr6byAf0Y4oPoiwxmsSmAR8N0rXpLv5en3F17h5MaZdkHUNu3
MTumw/f8Wg3Ybo2p7Fqiy9bzt4tr+HZtWklCn/YuE7m1x8BmBGgt+MABAgCL
```

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

		<p>symmetric encryption method, and the result is ciphertext.</p> <p>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.</p>
2	<p>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):</p> <pre>gpg --import their publickey.key</pre> <p>Now list your keys with:</p> <pre>gpg --list-keys</pre>	<p>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.</p>
3	<p>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</p>	<p>What does the –a option do: Create ASCII armored output. The default is to create the binary OpenPGP format</p> <p>What does the –r option do:</p>

	<p>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.</p> <p>What does the –u option do:</p> <p>Use name as the key to sign with. Note that this option overrides ‘--default-key’.</p> <p>Which file does it produce and outline the format of its contents:</p> <p>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.</p>
4	<p>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</p>

1

```
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.
```

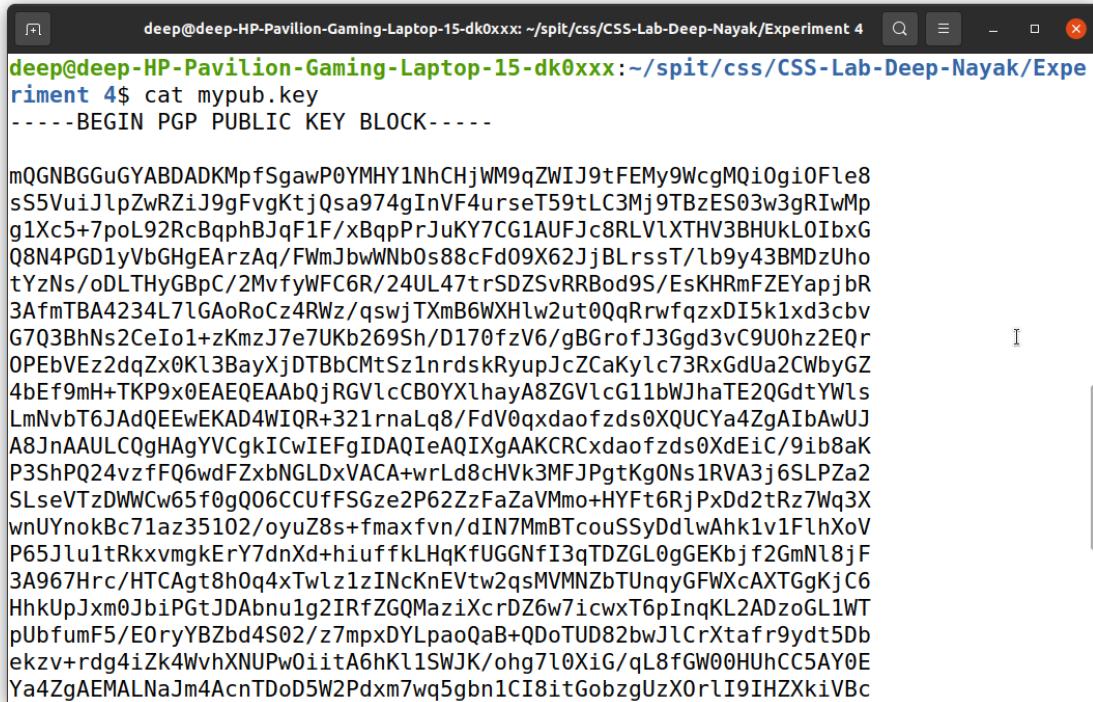
```
deep@deep-HP-Pavilion-Gaming-Laptop-15-dk0xxx:~/spit/css/CSS-Lab-Deep-Nayak/Experiment 4$ "Deep Nayak <deepmumbai16@gmail.com>

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 B175AA1FCDBB345D marked as ultimately trusted
gpg: directory '/home/deep/.gnupg/openpgp-revocs.d' created
gpg: revocation certificate stored as '/home/deep/.gnupg/openpgp-revocs.d/7EDF6D
6B9DA2EAF3F15D574AB175AA1FCDBB345D.rev'
public and secret key created and signed.

pub    rsa3072 2021-12-06 [SC] [expires: 2023-12-06]
      7EDF6D6B9DA2EAF3F15D574AB175AA1FCDBB345D
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/Experiment 4$ gpg --export -a "Deep Nayak" > mypub.key
deep@deep-HP-Pavilion-Gaming-Laptop-15-dk0xxx:~/spit/css/CSS-Lab-Deep-Nayak/Experiment 4$ gpg --export-secret-key -a "Deep Nayak" > mypriv.key
```

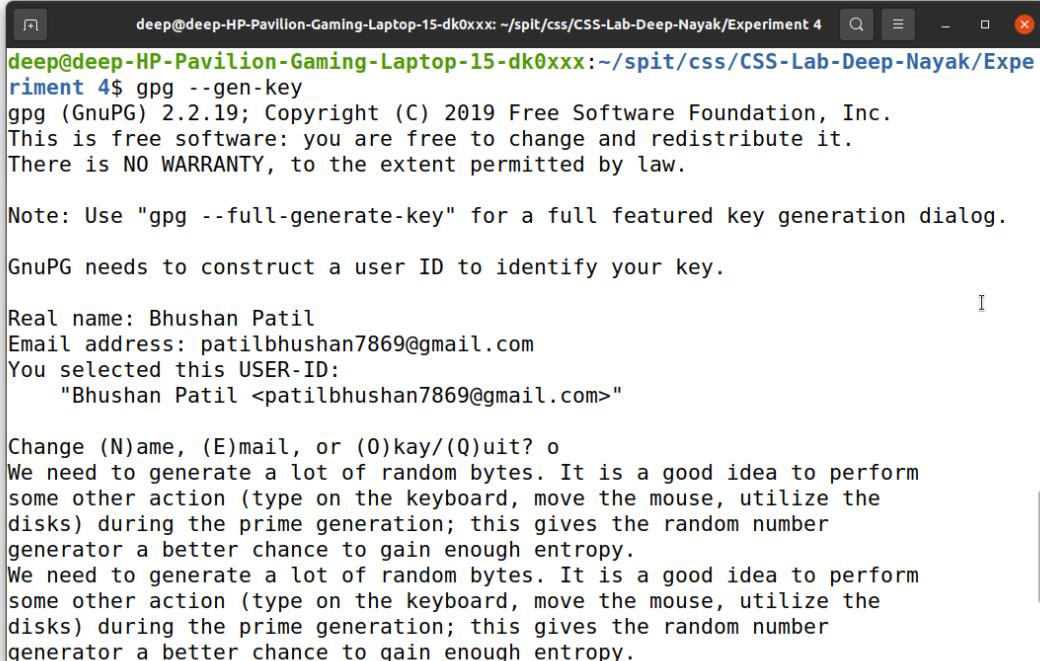


A terminal window titled "deep@deep-HP-Pavilion-Gaming-Laptop-15-dk0xxx: ~/spit/css/CSS-Lab-Deep-Nayak/Experiment 4". The command "cat mypub.key" is run, displaying a long string of characters representing a PGP public key.

```
deep@deep-HP-Pavilion-Gaming-Laptop-15-dk0xxx:~/spit/css/CSS-Lab-Deep-Nayak/Experiment 4$ cat mypub.key
-----BEGIN PGP PUBLIC KEY BLOCK-----

mQGNBGGuGYABDADKMPfSgawP0YMHY1NhCHjWM9qZWIJ9tFEMy9WcgMQi0gi0Fle8
sS5VuiJlpZwRZiJ9gFvgKtjQsa974gInVF4urseT59tLC3Mj9TBzES03w3gRIwMp
g1Xc5+7poL92RcBqphBjQf1F/xBqpPrJuKY7CG1AUJc8RLVLXTHV3BHUKL0IbxG
Q8N4PGD1yVbGHgEAzAq/FWmJbwWNbOs88cFd09X62JjBLrssT/lb9y43BDmzUho
tYzNs/oDLTHyGBpC/2MvfYWF6R/24UL47trSDZSvRRBod9S/EsKHRmFZEYapjbR
3AfTBA4234L7lGAoRoCz4RWz/qswjTXmb6WXHlw2ut0QqRrwfqzxDI5k1xd3cbv
G7Q3BhNs2CeIo1+zKmzJ7e7UKb269Sh/D170fzV6/gBGrofJ3Ggd3vC9U0hz2Eqr
0PEbVEz2dqZx0Kl3BayXjDTBbCMtSz1nrdskRyupJcZCaKylc73RxGdUa2CwbyGZ
4bEf9mh+TKP9x0EAEEAAbQjRGVlcCB0YXlhayA8ZGVlcG11bWJhaTE2QGdtYWls
LmNvbT6JAdQEEwEKAD4WIQR+321rnaLq8/FdV0qxdaofzds0XQUCYa4ZgAIbAwUJ
A8JnAAULCQgHAgYVCgkICwIEFgIDAQIeAQIXgAAKCRCxdaofzds0XdEiC/9ib8aK
P3ShPQ24vzfFQ6wdFZxbNGLDxVACA+wrLd8CHVk3MFJPgtKg0Ns1RVA3j6SLPZa2
SLseVTzDWWCw65f0gQ06CCUffSGze2P62ZzFaZaVMmo+HYFt6RjPxDd2tRz7Wq3X
wnUYnokBc71az35102/oyuZ8s+fmaxfn/dIN7MmBTcouSSyDdlwAhk1v1FLhXoV
P65Jlu1tRkxvngkErY7dnXd+hiuffLHqKfUGGNfI3qTDZGL0gGEKbjf2GmNL8jf
3A967Hrc/HTCAgt8h0q4xTwlz1zINCKnEvtw2qsMVMNzbTUngyGFWxcAXTggkjC6
HhkUpJxm0JbiPGtJDAbnu1g2IRfZGQMaziXcrDZ6w7icwxT6pInqKL2ADzoGL1WT
pUbfumF5/E0ryYBZbd4S02/z7mpxDYLpaoQaB+QDoTUD82bwJlCrXtafr9ydt5Db
ekzv+rdg4iZk4WvhXNUPw0iitA6hKl1SWJK/ohg7l0XiG/ql8fGW00HUhCC5AY0E
Ya4ZgAEMALNaJm4AcnTDoD5W2Pdxm7wq5gbn1CI8itGobzgUzX0rlI9IHZXkiVbc
```

2



A terminal window titled "deep@deep-HP-Pavilion-Gaming-Laptop-15-dk0xxx: ~/spit/css/CSS-Lab-Deep-Nayak/Experiment 4". The command "gpg --gen-key" is run, starting the key generation process.

```
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: Bhushan Patil
Email address: patilbhushan7869@gmail.com
You selected this USER-ID:
  "Bhushan Patil <patilbhushan7869@gmail.com>"

Change (N)ame, (E)mail, or (O)key/(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$ 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]
      7EDF6D6B9DA2EAF3F15D574AB175AA1FCDDDB345D
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$ 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$
```

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

G) TrueCrypt

No	Description	Result
1	<p>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.</p> <p>When you get to the Encryption Options, run the AES-Two-Sperate benchmark tests and outline the results:</p>	<p>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</p>

		<p>Which is the fastest: AES Which is the slowest: AES-Twofish-Serpent</p>
2	Select AES and RIPEMD-160 and create a 100MB file. Finally select your password and use FAT for the file system.	<p>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.</p>
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).

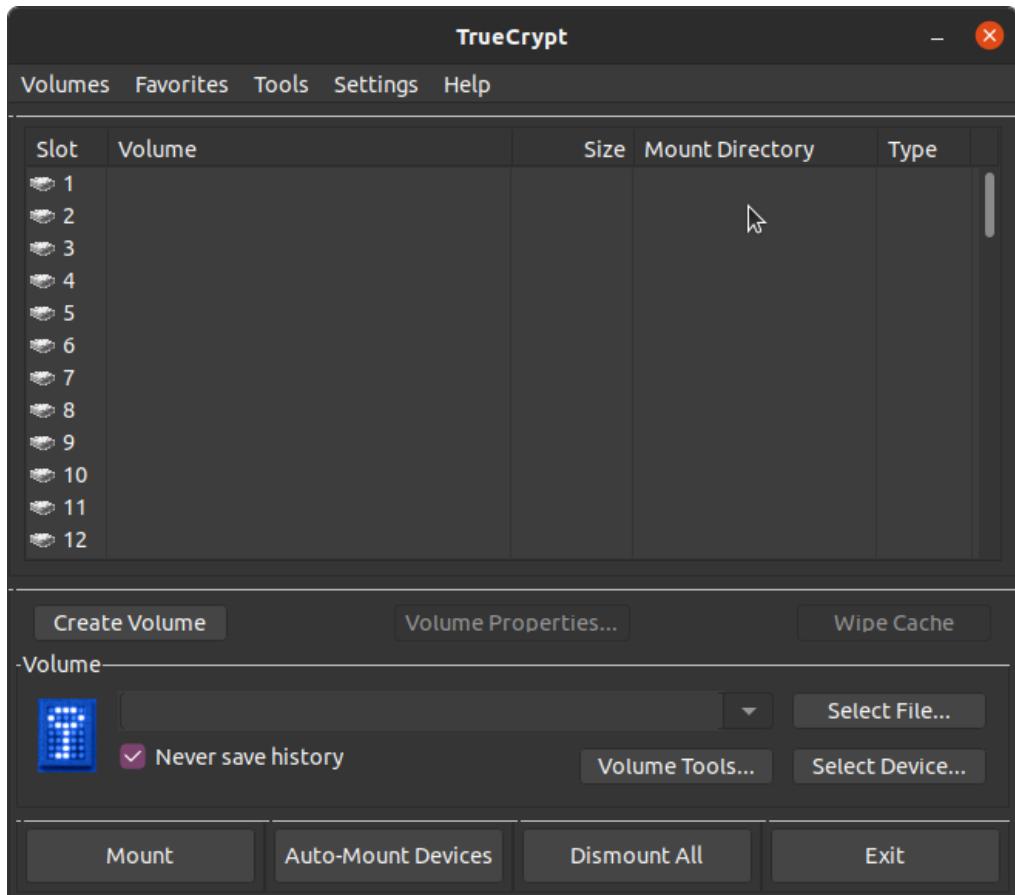
To install:
sudo add-apt-repository ppa:stefansundin/truecrypt
sudo apt-get update
sudo apt-get install truecrypt

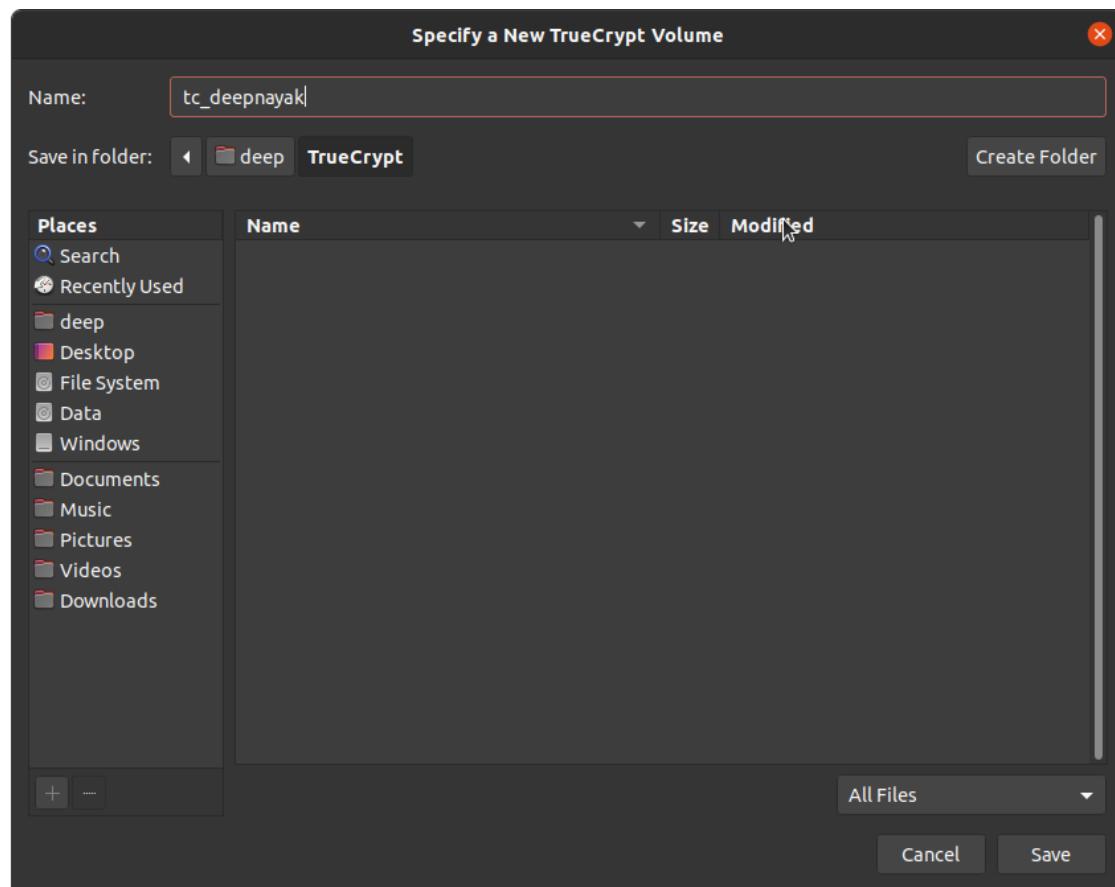
For server installations, you can install truecrypt-cli to install the
command-line only version. It is smaller and have fewer dependencies.

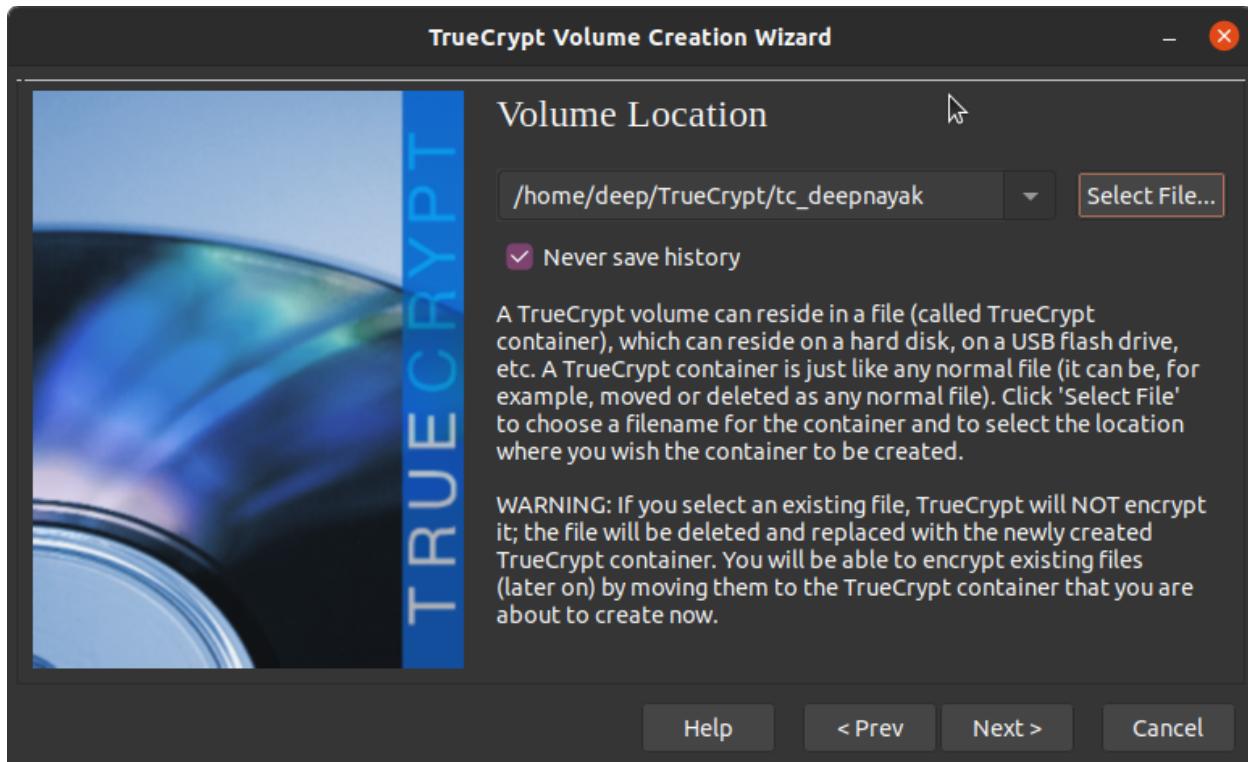
To automatically grant TrueCrypt root privileges to mount volumes, run:
sudo visudo -f /etc/sudoers.d/truecrypt
And put the following in the file:
%sudo ALL=(ALL) NOPASSWD:/usr/bin/truecrypt

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$ 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 libcurl4-openssl-dev libjs-inherits libjs-is-typedarray
  libjs-psl libjs-typedarray-to-buffer libpython2.7-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 - Encryption Algorithm Benchmark			
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

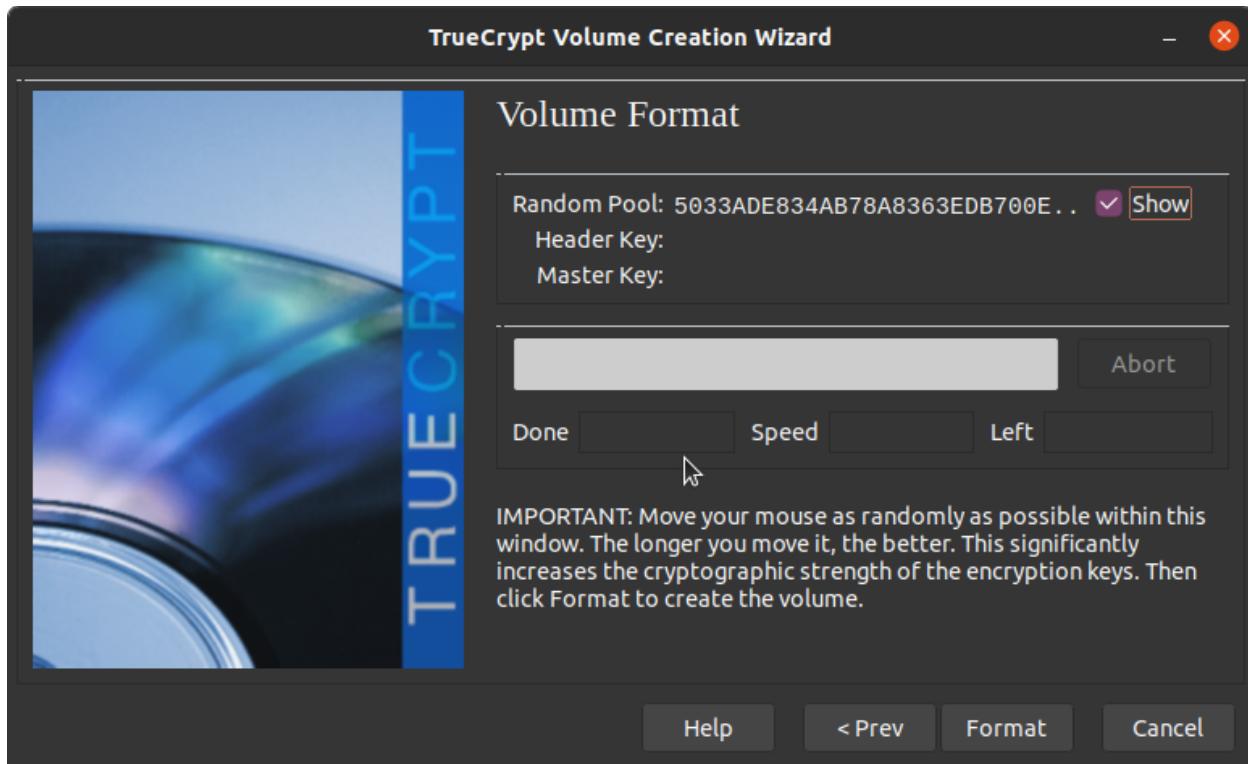
Buffer Size: 5.0 MB ▾

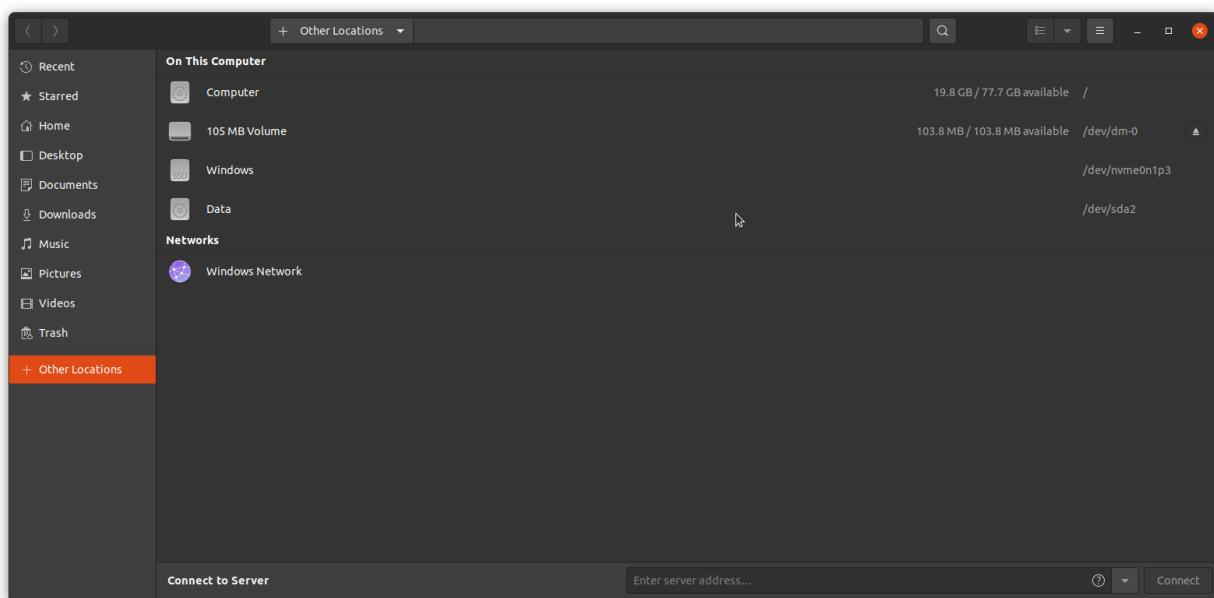
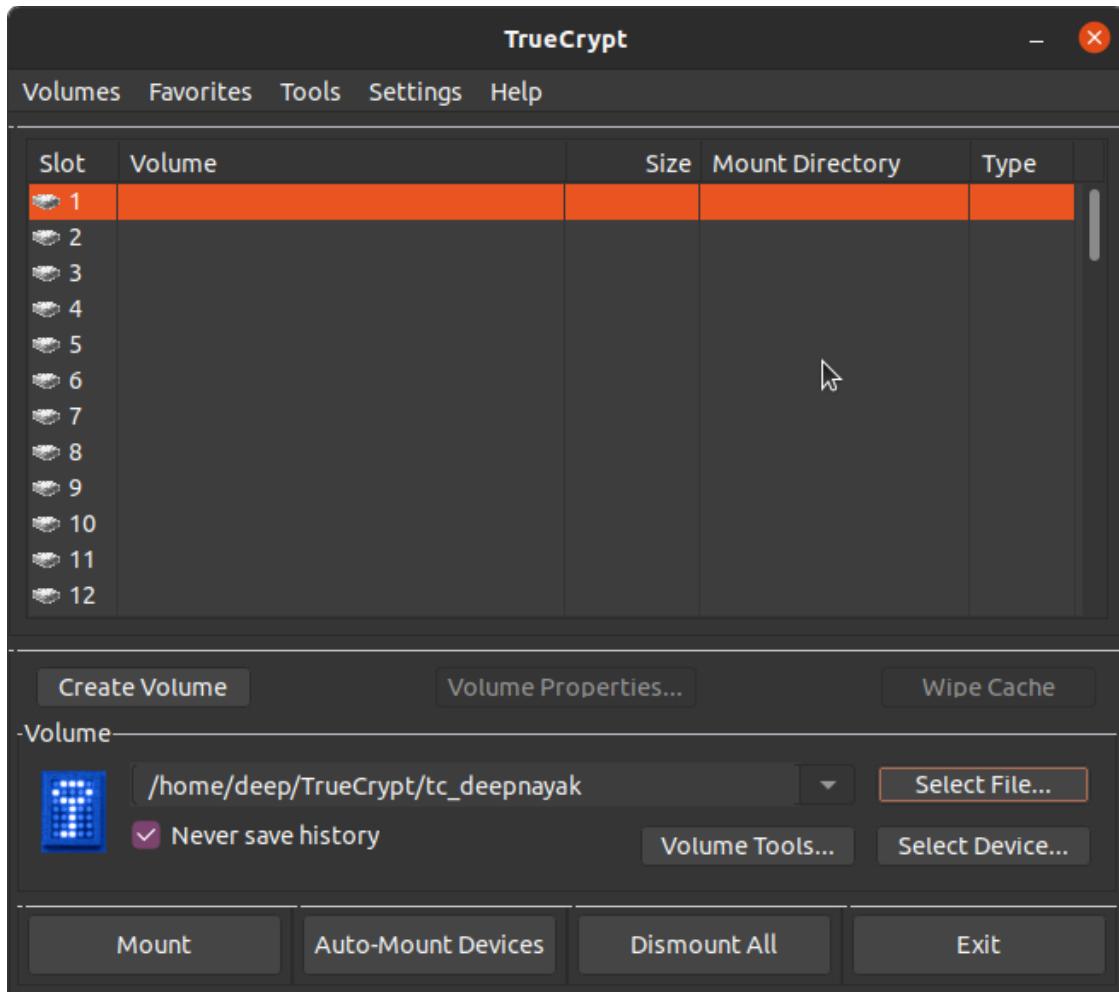
Benchmark Close

Speed is affected by CPU load and storage device characteristics.

These tests take place in RAM.







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>