

**NAME** Ameya Jangam

**BRANCH :** Comps

**UID :** 2019130025

**COURSE :** CSS LAB

**EXPERIMENT :** 4

**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**

[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 v1.4.6 (GNU/Linux)  
  
mQGiBE6l9PgRBADWXujraR0NZ80/mR6pzPWTFOB79HDdzlSyvel2WGhekbQVPYXu  
/sFSknzFu/Tp1up8q9MmNVVDkB/ItNaDuUYt3qZxbuNot10chnelu0mFztLc4GeB  
e6RvrzywRBqGZ1sfATrFSBXoTheDeep/oyIYrDShnDPFgeLluek8GCBHwCg9mSJ  
JBLkkrmG36j1s1X+WwpCrsEAMHQkgdASU6lfwwExzAI2gEG57voDxYAfLrQsUXA  
5BTxYnFT0CgmjeL6q70dn1jZ01ljFzkOKXTi14ylvoHyyA/smm8duuzJzFb091r8  
7KZyRpFRZLEdn0LkAa8X5XefzLqJvLaL1i+0WctUk1WH4nBP7UImNfAnze7y5PH  
txTuA/9PGB3uQIdj6jADLmXc3LVCUy0RzFvtFaZ1ETJ7Nk1EDKKUNzY9Jx0EGsG+  
D8AMG6r0/H3peBL96vBJYUFwrJN6JAjaDoBCd0waLKcA+X/A9NYsawoc0ic4JTr1  
uyZh8gp3i2VUTJAC6uSRx0GWgx6Y2fmo3F1Vb03K3Je7X12sIbqeU2fIFJLZWQg  
PHJLZWR5Qhdpa2ltZWRpYS5vcmc+iGAEEExECACAFAK6l9PgCGwMGCrkIBwMCBBUC  
CAMEFgIDAQIeAQIxgAAKCRCbabMqnTu3sIPtAKCTkL5JCewq6BVgfu6FfbuEh0Vc
```

Determine

Version:	4
User ID:	Sam Reed <reedy@wikimedia.org>
Key Fingerprint(20 Bytes in hex):	1a24253c8ba01e44a8cb470e3bbb95ce2b08bfd2
Key ID (8 bytes in hex):	3bbb95ce2b08bfd2

## Key Size :

```
1 str1="mQGiBE6l9PgRBADWXujraR0NZ80/mR6pzPWTFOB79HDdzlSyvel2WGhekbQVPYXu/sFSknzFu/Tp1up8q9Mm  
2 print(len(str1))
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

```
ameya@ameya-HP-Pavilion-Gaming-Laptop-15-dk0xxx:~/Ameya-Ubuntu/SPIT LAB SUBMISSIONS/SEM 5/css-exp4/task_a$ python3 a1.py  
1545
```

## Key Encryption Algorithm : RSA

- SATOSHI NAKAMOTO (Creator of Bitcoin)

[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):

```
SGJKqCnebuZvwutqy1vXRNVPQFVLVVo2jJCBHWjb03fmXmavIUTRCHoc8xgVJMQ
LrwvSS943GsqSbdOKZwdTnfEq+Uago+0fV66NpT3Yl0CXUiNBITZOJcJdjHDTB0
XRqomX2WSguv+btYdhQGGQiaEx73XMftXNCxb0pqwsODQns7xTcl2ENru9BNIQME
I7L9FYBQujKHM1k6RBy1as8XEls2jEos7GAmlf1wShFUx+Nf1VOPdbn3ZdFowq
suJkk+QbrwADBQgA9DiD4+uuRhkw2B1TmtrXnwwhcdkE7zBLhjxBfcslPAZiph8c
ICtV3S418i4HYC22Itcn8KAPOsGmipyS23AU1b7zJYPOdBn8E7apSPzHJfudMK
MqiCHljVJE23xsKTC0sIhhSKcr2G+6AR0G51wuqqJqEyDrblVQ0FpVxBNPHTStqu
05PoLXqc7PKgc5SyQuZbEALEKItl2SL2yBRRG0lVJLnvZ6eaovkAtgsb6dlieOr0
UwluJCwzzUBDrUMYAfyqBvVfXZun3m84rW7Jclp18mXITwGCVHg/P5n7QMBbfZQ
A25ymkuj636Nqh+c4zRnSINfyrdcID7AcqEb6InJBBgRAgAJBQJJCfqnAhmAoJ
EBjAnoZeyUihPrcAniVwl5M4RUgctje+IMNX4eVkc08AJ9v7Cxp5u0dQn08q3R
8RHwN4Gk8w==

-----END PGP PUBLIC KEY BLOCK-----
```

Version:	4
User ID:	Satoshi Nakamoto <satoshi@gmx.com>
Key Fingerprint(20 Bytes in hex):	ea4e9c907f7221b0b37d0940cf1857e6d6aaa69f
Key ID (8 bytes in hex):	cf1857e6d6aaa69f
Public Key (MPIs in base64):	<input type="button" value="ELGAMAL"/> s6mrCw4NCezvFnXYQ2u70E0hAwQjsv0VgFBsIoebWTpgshLqvzxcsvLaMSizsYCaV8XBKEVRf40XVU4 91s3d10WhaqxSMqT5BuvAMFCAD001Pj665GHCTYHvoa2tefDCFx2QtTlssePEF8Kws8BmI+HxwgJ9XdL jXyLgfVgLPYi1ycLwoA+hLqaKnJLbwBTUhVmLg84MGfwTt09I/Mcl+50woygIIewNUmsTbfGwpMLSwiGF IpyvYb7oBGgbmXC6iomoTI0tuVVBAwLxee08dJ0q47k+gtdBzs8qALLJC5lsQAsSq12XZIVbIFFEY6VU kue9np5qj+0CWcxsz2WJ46VRTBa4kLDnn4EOu4xqB/JAG9h9dm6fdmbzitbslyWnXyzChPAYJuEd8/mft Axsf9lAbdnkaS6Prfo2qH5zjNGdIg1/KsNwgPsByoRvo

## Key Size :

```
Get Started a1.py x
home > ameya > Ameya-Ubuntu > SPIT LAB SUBMISSIONS > SEM 5 > css-exp4 > task_a > a1.py > ...
1 str1="mQGiBEkj+qcRBADKDTcZLyDRtP1Q7/ShuzBJzUh9hoVwogf2W07U6G9BqKW24rpi0xYmErjMFfVntozN
2 str1=str1.strip()
3 print(len(str1))
4
5
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

```
ameya@ameya-HP-Pavilion-Gaming-Laptop-15-dk0xxx:~/Ameya-Ubuntu/SPIT LAB SUBMISSIONS/SEM 5/css-exp4/task_a$ python3 a1.py
1573
ameya@ameya-HP-Pavilion-Gaming-Laptop-15-dk0xxx:~/Ameya-Ubuntu/SPIT LAB SUBMISSIONS/SEM 5/css-exp4/task_a$
```

## Key Encryption Algorithm : ELGAMAL

## ● Tim Starling (Founder of wikimedia )

[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-----
z50zPAUCXPis3Q0JUDE2+EwAKCRBz8Ub+z50zPG5QD/43Zbtk9fTPKxWneuH5vsPK
k00s9TzSqdCwDA6VL2/d5LCB9xoh26CddEaxNbVZpxzzYf+3ABJg4Adqzff+h3VP
SujefPXi9UBJUX1ZzK9MFda3U++S7+m9eLiAA+y0g0Hd4NCV2ia2BWYTDRll0if
qSKFPc9JXIXnnmLy2cNZ1z0Ms0T0aJ04kt5nvi9cJowavokKRR6JfDsduxS62
whUp3tCOLgh+wxvqirJbv3xRNf8BAoig6wA1GgEX8M/0lsecYRZ24ExAMcJpvor
4AzHm3I+f4TH/isq4/yUchtELeonSCVLT04aTzJAUev/paIn7c7MKLlr8zs04/
7s/jAxccsM47wtbjx3/bMu51XDSoBscsndlPICs11sCNwo8m5B0+ya2pRhba22c
c80tYF6h0pN2MX1LNK/SQ73A7KJXINW/mirhl/6KCex6jLMzUoGFTfTLaoyn5f2VQ
CeVPgsoS2cbDv7W6wYubERFgz/ANp0bAt6rC0JFdly7Lhs3yIi6imlDPM+YLFZH
HpX8QQ5nTB9RJY5xEbiyVFjENcls7qecJg/9PKmRPAs0c432reIASRGupbTExet5
vqLVFA6/Rw2Hde+KT0UamQtywZ1fNZLDaiydZy86ruR0dWmUGiZcU0kIebpegeV
AVntafwpGnbetpsZysVvdw==
=TX3F
-----END PGP PUBLIC KEY BLOCK-----
```

<input type="button" value="Determine"/>	
Version:	4
User ID:	Tim Starling <tstarling@wikimedia.org>
Key Fingerprint(20 Bytes in hex):	bc4195238ffdd816a24d37111075249fccc9caaf
Key ID (8 bytes in hex):	1075249fccc9caaf
Public Key (MPIs in base64):	RSA kevc8nK6VeN3+LE3CTE8aHEWod9vThVRCC+uWf6iv/ZkOYijiU07ulPwPKr0its51H8JDUGtd+WseaPa

## Key Size :

```
home > ameya > Ameya-Ubuntu > SPIT LAB SUBMISSIONS > SEM 5 > css-exp4 > task_a > a.py > ...
1 str1="mQINBFRT08oBEADIUo4XuMpVQWPKBhjovVNt8buSdUGuhZOYt4Nru3zRqWFyEhZQ lHm+HmMD060XiuCiK30
2 print(len(str1))
3
4
5 
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

```
ameya@ameya-HP-Pavilion-Gaming-Laptop-15-dk0xx:~/Ameya-Ubuntu/SPIT LAB SUBMISSIONS/SEM 5/css-exp4/task_a$ python3 a1.py
3865
ameya@ameya-HP-Pavilion-Gaming-Laptop-15-dk0xx:~/Ameya-Ubuntu/SPIT LAB SUBMISSIONS/SEM 5/css-exp4/task_a$ 
```

## 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
<b>B.1</b>	First we need to generate a key pair with: openssl genrsa -out private.pem 1024 This file contains both the public and the private	What is the type of public key method used: <b>RSA key generation algorithm</b>

	key	How long is the default key: <b>1024 bits</b>  How long did it take to generate a 1,024 bit key? : <b>About 1 second</b>
<b>B.2</b>	Use following command to view the output file: cat private.pem	<b>Screenshot for the same has been shown below the table</b>
<b>B.3</b>	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"> <li>● <b>Modulus</b></li> <li>● <b>Public Exponent</b></li> <li>● <b>Private Exponent</b></li> <li>● <b>Prime1</b></li> <li>● <b>Prime2</b></li> <li>● <b>Exponent1</b></li> <li>● <b>Exponent2</b></li> <li>● <b>Coefficient</b></li> </ul> Which number format is used to display the information on the attributes: <b>Hexadecimal</b>
<b>B.4</b>	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?  <b>The inclusion of a password on the private key provides an additional layer of security. Anyone who has access to the file can log into whatever we have access to if it is taken.</b>
<b>B.5</b>	Next we will export the public key: openssl rsa -in private.pem -out public.pem -outform PEM -pubout	View the output key. (Screenshot below the table)  What does the header and footer of the file identify?  <b>This indicates that the key in this file is the public key.</b>
<b>B.6</b>	Now create a file named "myfile.txt" and put a message into it. Next encrypt it with your public	<b>Myfile gets encrypted and a bin file is generated as shown in the below</b>

	key: openssl rsautl -encrypt -inkey public.pem -pubin -in myfile.txt -out file.bin	screenshots.
B.7	And then decrypt with your private key: openssl rsautl -decrypt -inkey private.pem -in file.bin -out decrypted.txt	The contents of file.bin get decrypted successfully and contain the same text as myfile.txt

## Screenshots

### B1.

```
ameya@ameya-HP-Pavilion-Gaming-Laptop-15-dk0xxx:~/Ameya-Ubuntu/SPIT LAB SUBMISSIONS/SEM 5/css-exp4/t
ask_b$ openssl genrsa -out private.pem 1024
Generating RSA private key, 1024 bit long modulus (2 primes)
.....+++++
.....+++++
e is 65537 (0x010001)
```

### B2.

```
ameya@ameya-HP-Pavilion-Gaming-Laptop-15-dk0xxx:~/Ameya-Ubuntu/SPIT LAB SUBMISSIONS/SEM 5/css-exp4/t
ask_b$ cat private.pem
-----BEGIN RSA PRIVATE KEY-----
MIICXQIBAAKBgQC+L/4g0ySJxgGkuyU4C00cQCIxvKon+GnEY6HtJzgtlfefwIo2
stRn2dDb87A857Bb4F4RmI/k5HGrLxTJySuBQgnTrGIjSd5IH4MTE0kwvxw2/xYp
//0I//92+1vu1/zPyzgoWttaP054JNhrjV6BLEjHsW8xZ72gAbEuzYmIjQIDAQAB
AoGAJ5eAFL8/rPquXueioN6g22cydx4qYfXbtuNy0xbGCB9fFD+VxjtGInRimVdK
idLL30SBULJsyh6QVy20607YokysTwMEAZ/i8BI086l+OKlwTCCV9pwblafW0CN
ltXMpp9mngzLE68i9ytRjL3/X9vNF/BzI7a30nvkJgnx/pUCQQDxMTKkbzKTyjkC
bGFaVarXv0VSHH80AEUG7uzk2wtXGNVpceNL1v1ZCiLCHfACrznieUlKdfVxqr7u
vUk5YnlrAkEAyd0o0JXgaDvgocnLElFoJTrL8Wzk+7hEj18gUNgnlPhPftEk//Ju
oxsQHx6jHVMhUzZYRpq+UaAYRqx7Rxsr5wJAWOSC/6uj/mg3nrcHImKArasd16x9p
mmBHcgkg8i/w/AbBEP3Sge8YrKpZqSWTgcnHEbsV23Rxk5UUjbThue6bRQJBKRy
OwrlsftkFB0zDDkBAn0Z03yo9l2t/orBvWS+wkczeidYn+i2CbVsHZUMc1s3VqWg
Vwellje00Vf5uS/zDa8CQQCSx43VFKjQQk949IVFOcDiVvk3U9WmJt4dWT3fsLrb
HsE4pdMHSqm0IiW54eW4+ELgger0vME+QQD8j6NXk+sx
-----END RSA PRIVATE KEY-----
```

### B3.

```

ameya@ameya-HP-Pavilion-Gaming-Laptop-15-dk0xxx:~/Ameya-Ubuntu/SPIT LAB SUBMISSIONS/SEM 5/css-exp4/1
ask_b$ openssl rsa -in private.pem -text
RSA Private-Key: (1024 bit, 2 primes)
modulus:
00:be:2f:fe:20:d3:24:89:c6:01:a4:bb:25:38:08:
e3:9c:40:22:31:bc:aa:27:f8:69:c4:63:a1:ed:27:
38:2d:95:f7:9f:c0:8a:36:b2:d4:67:d9:d0:db:f3:
b0:3c:4b:b0:5b:e0:5e:11:98:8f:e4:e4:71:ab:2f:
14:c9:c9:2b:81:42:09:d3:ac:62:23:49:de:48:1f:
83:13:13:49:30:bf:1c:36:ff:16:29:ff:fd:08:ff:
ff:76:fb:5b:ee:d7:fc:cfc:38:28:5a:db:5a:3c:
ee:78:24:d1:eb:8d:5e:81:2c:48:c7:b1:6f:31:67:
bd:a0:01:b1:2e:cd:89:88:8d
publicExponent: 65537 (0x100001)
privateExponent:
27:97:80:14:bf:3f:ac:fa:ae:5e:e7:a2:a0:de:a0:
db:67:32:77:1e:2a:61:f5:c1:b6:e3:72:d3:16:c6:
08:1f:5f:14:3f:95:c6:3b:46:22:74:62:99:57:4a:
89:d2:cb:dc:e4:81:50:b2:6c:ca:1e:90:57:2d:8e:
eb:e4:d8:a2:4c:ac:4f:03:04:01:9f:e2:f0:12:34:
f3:a9:7e:38:a5:b0:4c:20:95:f6:9c:1b:96:56:9f:
5b:40:8d:96:d5:cc:a6:9f:66:9e:0c:e5:13:af:22:
f7:2b:51:8c:bd:ff:5f:db:cd:17:f0:73:23:b6:b7:
3a:7b:e4:26:09:f1:fe:95

prime1:
00:f1:31:32:a4:6f:32:93:ca:39:02:6c:61:5a:61:
aa:d7:bf:45:52:1c:7f:34:00:45:06:ee:ec:e4:db:
0b:57:18:d5:69:71:e3:4b:d6:fd:59:0a:22:c2:fd:
f0:02:af:39:e2:79:49:64:0d:f5:71:aa:be:ee:bd:
49:39:62:79:6b
prime2:
00:c9:dd:28:38:95:e0:68:3b:e0:a1:c9:e5:12:51:
68:25:3a:cb:f1:6c:e4:fb:b8:44:8f:5f:20:50:d8:
27:58:f8:4f:7e:d1:24:ff:f2:6e:a3:1b:10:1f:1e:
a3:1d:53:21:53:36:58:46:9a:be:51:a0:18:46:ac:
7b:47:1b:2b:e7
exponent1:
58:e4:82:ff:ab:a3:fe:68:37:9e:b7:07:22:62:80:
ae:c7:65:eb:1f:69:9a:66:c7:0a:09:20:f2:2f:f0:
fc:06:c1:10:fd:d2:81:ef:18:ac:aa:59:a9:25:93:
81:c9:c7:11:bb:15:db:74:71:93:95:14:8d:b4:e1:
b9:ee:9b:45
exponent2:
00:a4:72:3b:0a:e5:e5:fb:64:14:1a:19:0c:39:01:
02:73:99:d3:7c:a8:f6:5d:ad:fe:84:41:bd:64:be:
c2:47:33:7a:27:58:9f:e8:b6:09:b5:6c:1d:95:0c:
73:5b:37:56:a5:a0:57:07:a5:96:37:8e:d1:57:f9:
b9:2f:f3:0d:af
coefficient:
00:92:c7:8d:d5:14:a8:d0:42:4f:78:f4:85:45:39:
c0:e2:56:f9:37:53:d5:a6:26:de:1d:59:3d:df:b0:
ba:db:1e:c1:38:a5:d3:07:4a:a9:b4:22:25:b9:e1:
e5:b8:f8:42:e0:81:ea:f4:bc:c1:3e:41:00:fc:8f:
a3:57:93:eb:31
...

```

```

a3:57:93:eb:31
writing RSA key
-----BEGIN RSA PRIVATE KEY-----
MIICXQIBAAKBgQC+L/4g0ySJxgGkuyU4C00cQCIXvKon+GnEY6HtJzgtlfefwIo2
stRn2db87A857Bb4F4RmI/k5HGrLxTJySuBQgnTrGIjSd5IH4MTE0kwvxw2/xYp
//0I//92+1vu1/zPyzgoWttap054JNHrjV6BLEjHsW8xZ72gAbEuzYmIjQIDAQAB
AoGAJ5eAFL8/rPquXueioN6g22cydx4qYfXBuNy0xbGCB9fFD+VxjtGInRimVdK
idLL30SBULjsyh6QVy20607YokstsMEAZ/i8BI086l+OKWwTCCV9pbllafW0CN
ltXmpp9mngzle68i9ytRjL3/X9vNF/BzI7a30nvkJgnx/pUCQQDxMTKkbzKTyjkC
bGFaYarXv0VSHH80AEUG7uzk2wtXGNVpceNL1v1ZcILCHfACrznieUlkDfVxqr7u
vUk5YnlrAkEAyd0o0JXgaDvgocnlElFoJTrL8Wzk+7hEj18gUNgnWPhPftEk//Ju
oxsQHx6jHVMhUzZYRpq+UaAYRqx7Rxsr5wJA0OSC/6uj/mg3nrcHIMKArndl6x9p
mmBHCgkg8i/w/AbBEP3Sge8YrKpZqSWTgcnHEbsV23Rxx5UUjbThue6bRQJBKRy
0wrl5ftkFB0zDDkBAAn0Z03yo9l2t/orBvWS+wkczeidYn+i2CbVsHZUMc1s3VqWg
Vwellje00Vf5uS/zDa8CQQCSx43VFkjQQk949IVF0cDiVvk3U9WmJt4dWT3fsLrb
HsE4pdMHSqm0IiW54eW4+ELgger0vME+QD8j6NXk+sx
-----END RSA PRIVATE KEY-----
ameya@ameya-HP-Pavilion-Gaming-Laptop-15-dk0xxx:~/Ameya-Ubuntu/SPIT LAB SUBMISSIONS/SEM 5/css-exp4/t

```

## B4

```

ameya@ameya-HP-Pavilion-Gaming-Laptop-15-dk0xxx:~/Ameya-Ubuntu/SPIT LAB SUBMISSIONS/SEM 5/css-exp4/t
ask_b$ openssl rsa -in private.pem -des3 -out key3des.pem
writing RSA key
Enter PEM pass phrase:
Verifying - Enter PEM pass phrase:

```

## B5

```

ameya@ameya-HP-Pavilion-Gaming-Laptop-15-dk0xxx:~/Ameya-Ubuntu/SPIT LAB SUBMISSIONS/SEM 5/css-exp4/t
ask_b$ openssl rsa -in private.pem -out public.pem -outform PEM -pubout
writing RSA key
ameya@ameya-HP-Pavilion-Gaming-Laptop-15-dk0xxx:~/Ameya-Ubuntu/SPIT LAB SUBMISSIONS/SEM 5/css-exp4/t
ask_b$ cat public.pem
-----BEGIN PUBLIC KEY-----
MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQC+L/4g0ySJxgGkuyU4C00cQCIX
vKon+GnEY6HtJzgtlfefwIo2stRn2db87A857Bb4F4RmI/k5HGrLxTJySuBQgnT
rGIjSd5IH4MTE0kwvxw2/xYp//0I//92+1vu1/zPyzgoWttap054JNHrjV6BLEjH
sW8xZ72gAbEuzYmIjQIDAQAB
-----END PUBLIC KEY-----
ameya@ameya-HP-Pavilion-Gaming-Laptop-15-dk0xxx:~/Ameya-Ubuntu/SPIT LAB SUBMISSIONS/SEM 5/css-exp4/t

```

## B6

```

ameya@ameya-HP-Pavilion-Gaming-Laptop-15-dk0xxx:~/Ameya-Ubuntu/SPIT LAB SUBMISSIONS/SEM 5/css-exp4/t
ask_b$ cat myfile.txt
There's no match that you can't win, and there's no match that you'll win for sure
ameya@ameya-HP-Pavilion-Gaming-Laptop-15-dk0xxx:~/Ameya-Ubuntu/SPIT LAB SUBMISSIONS/SEM 5/css-exp4/t
ask_b$ openssl rsautl -encrypt -inkey public.pem -pubin -in myfile.txt -out file.bin
ameya@ameya-HP-Pavilion-Gaming-Laptop-15-dk0xxx:~/Ameya-Ubuntu/SPIT LAB SUBMISSIONS/SEM 5/css-exp4/t
ask_b$ cat file.bin
♦t♦j♦e♦f♦>♦♦♦♦♦♦i♦U. X'♦♦♦♦H♦♦♦♦I♦♦♦♦k♦{♦-♦W♦6 Q♦W♦V♦C♦L♦♦♦7 P♦♦♦.♦"x♦t♦e♦O♦P♦e♦X♦n,♦♦{♦]v,♦b?♦m♦D♦x♦p♦f♦T:♦u♦M♦e♦♦♦@♦i♦e♦G♦
&8;♦♦♦;♦!i♦q♦a♦m♦e♦y♦a♦@♦a♦m♦e♦y♦a♦-♦H♦P♦-♦P♦a♦v♦l♦i♦o♦n♦-♦G♦a♦m♦i♦n♦-♦L♦a♦p♦t♦o♦-♦1♦5♦-♦d♦k♦0♦x♦x♦x♦
ameya@ameya-HP-Pavilion-Gaming-Laptop-15-dk0xxx:~/Ameya-Ubuntu/SPIT LAB SUBMISSIONS/SEM 5/css-exp4/t

```

## B7

```
ameya@ameya-HP-Pavilion-Gaming-Laptop-15-dk0xxx:~/Ameya-Ubuntu/SPIT LAB SUBMISSIONS/SEM 5/css-exp4/t
ask_b$ openssl rsa -decrypt -inkey private.pem -in file.bin -out decrypted.txt
ameya@ameya-HP-Pavilion-Gaming-Laptop-15-dk0xxx:~/Ameya-Ubuntu/SPIT LAB SUBMISSIONS/SEM 5/css-exp4/t
ask_b$ cat decrypted.txt
There's no match that you can't win, and there's no match that you'll win for sure
ameya@ameya-HP-Pavilion-Gaming-Laptop-15-dk0xxx:~/Ameya-Ubuntu/SPIT LAB SUBMISSIONS/SEM 5/css-exp4/t
ask_b$
```

## PART 2 :

### Generating the SSH key:

```
ameya@ameya-HP-Pavilion-Gaming-Laptop-15-dk0xxx:~/.ssh$ ssh-keygen -t rsa -C "ameya.g.jangam@gmail.com"
Generating public/private rsa key pair.
Enter file in which to save the key (/home/ameya/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/ameya/.ssh/id_rsa
Your public key has been saved in /home/ameya/.ssh/id_rsa.pub
The key fingerprint is:
SHA256:5syYxC2956M0NFb80VEhXJpvV0E9MjFPZ4MmQnKo7f8 ameya.g.jangam@gmail.com
The key's randomart image is:
+---[RSA 3072]----+
| .+..+++++o |
| oo..* ==oo |
| o o = +o.o. |
| o = . = o . |
| 0 S + o |
| + @ . o |
| + * . |
| . +. |
| .o.oE |
+---[SHA256]----+
```

The private key and public key files have been generated as shown below

```
ameya@ameya-HP-Pavilion-Gaming-Laptop-15-dk0xxx:~/.ssh$ ls
id_ed25519 id_ed25519.pub id_rsa id_rsa.pub known_hosts
ameya@ameya-HP-Pavilion-Gaming-Laptop-15-dk0xxx:~/.ssh$ |
```

The screenshot shows the GitHub account settings page for the user 'ameyajangam22'. On the left, a sidebar lists various account management options: Account settings, Profile, Account, Appearance, Accessibility, Account security, Billing & plans, Security log, Security & analysis, Sponsorship log, Emails, and Notifications. The 'Account settings' option is currently selected. The main area is titled 'SSH keys / Add new'. A text input field labeled 'Title' contains 'jangam\_ssh\_key'. Below it is a larger text area containing an SSH key (beginning with 'ssh-rsa AAAAAB3NzaC1yc2EAAAQAB...'). At the bottom right of this area is a green button labeled 'Add SSH key'.

The screenshot shows a GitHub repository page for 'ameyajangam22/memeify-app'. The repository is public. The code tab is selected. The repository details show it was initialized with 'Initial Setup with pages'. It contains one branch ('main'), no tags, and no releases. The 'Code' dropdown menu is open, showing options for 'Clone' (via HTTPS, SSH, or GitHub CLI) and 'Download ZIP'. The 'About' section indicates there is no description or website provided. A 'Readme' link is available.

```
ameya@ameya-HP-Pavilion-Gaming-Laptop-15-dk0xxx:~/Ameya-Ubuntu/SPIT LAB SUBMISSIONS/SEM 5/css-exp4/task_b$ git clone git@github.com:ameyajangam22/memeify-app.git
Cloning into 'memeify-app'...
remote: Enumerating objects: 52, done.
remote: Counting objects: 100% (52/52), done.
remote: Compressing objects: 100% (48/48), done.
remote: Total 52 (delta 2), reused 52 (delta 2), pack-reused 0
Receiving objects: 100% (52/52), 525.41 KiB | 692.00 KiB/s, done.
Resolving deltas: 100% (2/2), done.
```

Yes I was able to clone the repository.

### 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	Can you view your key? <b>Yes</b>

	<p>The file will only contain the private key (and should have 256 bits).</p> <p>Now use “cat priv.pem” to view your key.</p>	
C.2	<p>We can view the details of the ECC parameters used with:</p> <pre>openssl ecparam -in priv.pem -text -param_enc explicit -noout</pre>	<p>Outline these values:</p> <p>Prime (last two bytes): <b>fc:2f</b></p> <p>A: <b>0</b></p> <p>B: <b>7</b></p> <p>Generator (last two bytes): <b>d4:b8</b></p> <p>Order (last two bytes): <b>41:41</b></p>
C.3	<p>Now generate your public key based on your private key with:</p> <pre>openssl ec -in priv.pem -text -noout</pre>	<p>How many bits and bytes does your private key have: <b>256 bits</b></p> <p>How many bit and bytes does your public key have (Note the 04 is not part of the elliptic curve point):</p> <p><b>64 bytes and 512 bits</b></p> <p>What is the ECC method that you have used?</p> <p><b>secp256k1</b></p>

## C1

```
ameya@ameya-HP-Pavilion-Gaming-Laptop-15-dk0xxx:~/Ameya-Ubuntu/SPIT LAB SUBMISSIONS/SEM 5/css-exp4/task_c$ openssl ecparam -name secp256k1 -genkey -out priv.pem
ameya@ameya-HP-Pavilion-Gaming-Laptop-15-dk0xxx:~/Ameya-Ubuntu/SPIT LAB SUBMISSIONS/SEM 5/css-exp4/task_c$ cat priv.pem
-----BEGIN EC PARAMETERS-----
BgUrgQQACg==
-----END EC PARAMETERS-----
-----BEGIN EC PRIVATE KEY-----
MHQCAQEJHMaANgWLhkPcrTYD/M2JzZt96RXR0L7q6NxRjUQxRvXoAcGBSuBBAK
oUDQgAEPapNiEsfw2lcHOFW2D2AP4Q4qnhGRGZH+LmFlZ7h1dXvKOHUiaJicRKJ
WxDx8EQY2xg+v2C4VR8fbHzn+egrA==
-----END EC PRIVATE KEY-----
```

## C2

```

ameya@ameya-HP-Pavilion-Gaming-Laptop-15-dk0xxx:~/Ameya-Ubuntu/SPIT LAB SUBMISSIONS/SEM 5/css-exp4/task_c$ 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:ff:ff:ff:ff:ff:ff:ff:ff:ff:ff:ff:ff:ff:ff:ff:ff:ff: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:
  ff:fe:ba:ae:dc:e6:af:48:a0:3b:bf:d2:5e:8c:d0:
  36:41:41
Cofactor: 1 (0x1)

```

## C3

```

ameya@ameya-HP-Pavilion-Gaming-Laptop-15-dk0xxx:~/Ameya-Ubuntu/SPIT LAB SUBMISSIONS/SEM 5/css-exp4/task_c$ openssl ec -in priv.pem -text -noout
read EC key
Private-Key: (256 bit)
priv:
  73:1a:00:d8:16:94:79:0f:72:b4:d8:0f:f3:36:27:
  36:6d:f7:a4:57:47:42:fb:ab:a3:71:46:35:10:c5:
  1b:d7
pub:
  04:3d:aa:4d:88:4b:1f:c3:69:5c:1c:e1:56:d8:3d:
  80:3f:84:38:aa:78:46:44:66:47:f8:b9:85:95:9e:
  e1:d5:d5:ef:28:e1:d4:21:a2:62:71:12:89:5b:1f:
  c3:c7:c1:10:63:6c:60:fa:fd:82:e1:54:7c:7d:b1:
  f3:9f:e7:a0:ac
ASN1 OID: secp256k1

```

## D) Elliptic Curve Encryption

### D.1) OUTPUT :

The first four characters are 1b36

```

++++Keys++++
Bob's private key: 027f57da49b443220a941a3ec4edd6e8911090ed0344c64fda88675573b4572cb5e462b7
Bob's public key:
0405ab4932e3a2c73e6ab1d0767ba0dba8e23894664544d28237eb9fbcd99c778b49c4539502de67927c38a834cd5a60129f75c113423a1ac7a467
e504ce3b81b46046558fd0277ece

Alices's private key: 0217cf76058c40aca09582dcc8717e18cb2218f7648917393a2a2760fa20c40648869b48
Alices's public key:
0407de177af236383cb59f9559a4bbb219f5604e1437c47dab76938d16fe653c5fa869452f01fab21e156a4af931d0861db2d444f0881af2df9e6d
ec216cf2898fe9a7d50b9b57c4c3

++++Encryption++++
Cipher:
1b36f3320827c6afb0278a9e52a8acf50401c37ad4962799231c52338264ce76da9ac28a4aaecf47dbb7776052bea20629aa55906105b6891d59f5
693536a5492649e431f6bf7129e79c08fb0e54836ad13820dda61c677f24e1db4d5fff7608b95f0f37920c95be4d38cc8e0604e086743ad3480c1d
f75007cb3b72b466b54769bf9a0bb1cb4eebf0
Decrypt: Hello. Alice

Bob verified: True

++++ECDH++++
Alice:00b8b09875a078bfb5f22de5edcf853bfc3ea4f5bba4f94b765ee0f1a03f664
Bob: 00b8b09875a078bfb5f22de5edcf853bfc3ea4f5bba4f94b765ee0f1a03f664

```

How is the signature used in this example?

**Since Bob is encrypting and transmitting the message to Alice, she must first verify that the message came from Bob, so she verifies his authenticity using Bob's signature and public key.**

## D.2) OUTPUT :



## Simple Finding 20 Elliptic Curve 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.



<b>Parameters</b>	
a: <input type="text" value="0"/>	A: 0
b: <input type="text" value="7"/>	B: 7
Prime: <input type="text" value="89"/>	Prime number: 89 Elliptic curve is: $y^2=x^3+7$ Finding the first 20 points
<input type="button" value="Determine"/>	(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)
<b>Examples</b>	
The following are some examples:	
<ul style="list-style-type: none"><li>• <math>y^2 = x^3 + 7, p=23</math>. <a href="#">Try!</a></li><li>• <math>y^2 = x^3 + 7, p=101</math>. <a href="#">Try!</a></li><li>• <math>y^2 = x^3 + 7, p=802283</math>. <a href="#">Try!</a></li></ul>	

### First Five Points :

**(14,9) , (15,0) , (16,3) , (17,5) , (22,8)**

### D.3) OUTPUT :

## NIST 192p:

```
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 OUTPUT TERMINAL DEBUG CONSOLE

(task\_d) ameya@ameya-HP-Pavilion-Gaming-Laptop-15-dk0xxx:~/Ameya-Ubuntu/SPIT LAB SUBMISSIONS/SEM 5/css-exp4/task\_d\$ python3 ecc.py

Message: b'Bob'  
Type: NIST192p  
=====  
Signature: b'eEspAOjAHDfc54iTDrv9GvnhvxsN1BkL5WK/6zVXjc4ppkG7upUrefbxS06Svlbl'  
=====

## NIST 521p:

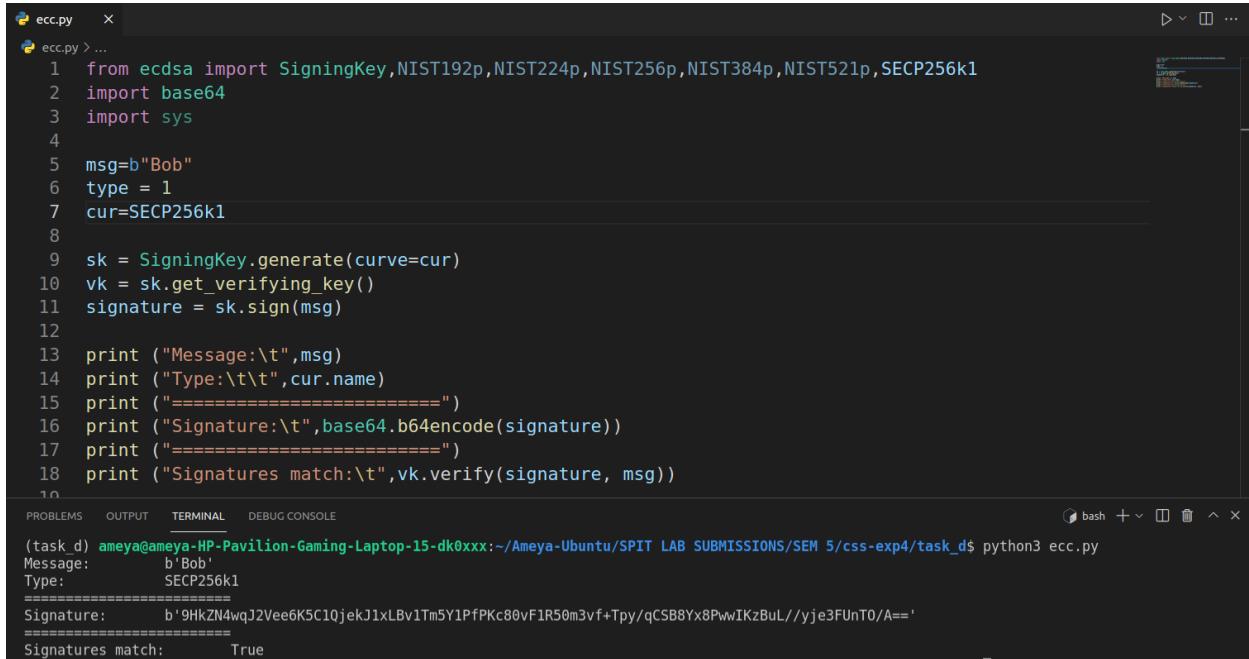
```
ecc.py > ...
1 from ecdsa import SigningKey,NIST192p,NIST224p,NIST256p,NIST384p,NIST521p,SECP256k1
2 import base64
3 import sys
4
5 msg=b"Bob"
6 type = 1
7 cur=NIST521p
8
9 sk = SigningKey.generate(curve=cur)
10 vk = sk.get_verifying_key()
11 signature = sk.sign(msg)
12
13 print ("Message:\t",msg)
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 OUTPUT TERMINAL DEBUG CONSOLE

(task\_d) ameya@ameya-HP-Pavilion-Gaming-Laptop-15-dk0xxx:~/Ameya-Ubuntu/SPIT LAB SUBMISSIONS/SEM 5/css-exp4/task\_d\$ python3 ecc.py

Message: b'Bob'  
Type: NIST521p  
=====  
Signature: b'AXH6pDLNiL55BFsHop++SnHsTUSuSno0AKG0iHmy2N2Te1XY1PiR1hWi83Yl1q0FGETQxk89NKAtInYTRSPGRgqbAXGRGhmsE+mpq5xk8xSJNXYXd0fyF6jJ/2iTjgZq34qYnMOCQW3sSPdL4ainxmIfODxJ0tWwm9t+YTqdIEJ2zP'  
=====  
Signatures match: True

## SECP256k1:



```
ecc.py > ...
1  from ecdsa import SigningKey,NIST192p,NIST224p,NIST256p,NIST384p,NIST512p,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    OUTPUT    TERMINAL    DEBUG CONSOLE
(task_d) ameya@ameya-HP-Pavilion-Gaming-Laptop-15-dk0xxx:~/Ameya-Ubuntu/SPIT LAB SUBMISSIONS/SEM 5/css-exp4/task_d$ python3 ecc.py
Message:      b'Bob'
Type:        SECP256k1
=====
Signature:   b'9HZN4wqJ2Vee6K5C1QjekJ1xLBv1Tm5Y1PfPKc80vF1R50m3vf+Tpy/qCSB8Yx8PwwIKzBuL//yje3FUnT0/A=='
=====
Signatures match:  True
```

## Application Areas of SECP256k1 :

SECP256k1 finds it's major application in the implementation of Bitcoin as it was constructed in such a way that it allows for efficient computation and also performs significantly better than other curves if it is optimised.

## Difference between Hashes :

The difference between the hash signatures that I observed was their size, since NIST192p uses 192 bit size elliptic curve the corresponding hash signature is 64bit and as this elliptic curve size increases like in NIST512 and SECP256 the size correspondingly increases.

## E) RSA

### E.1) OUTPUT :

**Picking the prime numbers**

**P = 11**

**Q = 5**

**N = 55 ,** where  $N = P \times Q$

**PHI = 40 ,** where  $\Phi = (P-1) \times (Q-1)$

**e = 3 ,** where **PHI** [gcd(PHI,e)=1]

**d = 27,** where  $(e \times d) \bmod \Phi = 1$

**Now for a message of M = 5, cipher is calculated as**

$$C = M^e \pmod{N} = 5^3 \pmod{55} = 125 \pmod{55} = 70$$

**After decrypting the above ciphertext**

$$C^d \pmod{N} = 70^{27} \pmod{55} = 5$$

**3 more examples :**

```
rsa.py      X  inverse.py
rsa.py > ...
1  p=11
2  q=5
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=5
13 cipher = M**e % N
14
15 print (cipher)

PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE  bash

ameya@ameya-HP-Pavilion-Gaming-Laptop-15-dk0xxx:~/Ameya-Ubuntu/SPIT LAB SUBMISSIONS/SEM 5/css-exp4/task_e$ python3 rsa.py
3 55
27 55
15
5
ameya@ameya-HP-Pavilion-Gaming-Laptop-15-dk0xxx:~/Ameya-Ubuntu/SPIT LAB SUBMISSIONS/SEM 5/css-exp4/task_e$
```

```
rsa.py      X  inverse.py
rsa.py > ...
1  p=7
2  q=3
3  N=p*q
4
5  PHI=(p-1)*(q-1)
6  e=5
7  for d in range(1,10000):
8  |    if ((e*d % PHI)==1): break
9  print (e,N)
10
11 print (d,N)
12 M=5
13 cipher = M**e % N
14
15 print (cipher)

PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE  bash

ameya@ameya-HP-Pavilion-Gaming-Laptop-15-dk0xxx:~/Ameya-Ubuntu/SPIT LAB SUBMISSIONS/SEM 5/css-exp4/task_e$ python3 rsa.py
5 21
5 21
17
5
```

```

rsa.py      x  inverse.py
rsa.py > ...
1  p=41
2  q=5
3  N=p*q
4
5  PHI=(p-1)*(q-1)
6  e=3
7  for d in range(1,1000):
8      if ((e*d % PHI)==1): break
9  print (e,N)
10
11 print (d,N)
12 M=5
13 cipher = M**e % N
14
15 print (cipher)

PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE
ameya@ameya-HP-Pavilion-Gaming-Laptop-15-dk0xxx:~/Ameya-Ubuntu/SPIT LAB SUBMISSIONS/SEM 5/css-exp4/task_e$ python3 rsa.py
3 205
107 205
125
5

```

## E.2) OUTPUT :

**Multiplicative inverse of n mod m ((Euclidean algorithm))**

[Encryption Home][Home]

In cryptography, we often need  $n^{-1}$ , which is a multiplicative inverse of  $n$  mod  $m$ , i.e.  $n/(n^{-1}) = 1 \pmod{m}$ . It is used in the calculation of the decryption key in RSA, and in other cryptography methods. With RSA, we get  $(e \times d) \pmod{N} = 1$ , where we have  $e$  and  $N$ , and must calculate  $d$  using the multiplicative inverse of  $n$  mod  $m$ .

**N (inverse value):**

**M (mod value):**

**Determine**

**Try an example**

- Inverse of 53 mod 120. [Test](#)
- Inverse of 65537 mod 10347768518374182260 12406113933120080. [Test](#)

Inverse of 53 mod 120  
Result: 77

 Multiplicative inverse of n mod m ((Euclidean algorithm))

[Encryption Home][Home] 

In cryptography, we often need  $n^{-1}$ , which is a multiplicative inverse of n mod m, i.e.  $n/(n^{-1}) = 1 \pmod{m}$ . It is used in the calculation of the decryption key in RSA, and in other cryptography methods. With RSA, we get  $(e \cdot d) \pmod{N} = 1$ , where we have e and N, and must calculate d using the multiplicative inverse of n mod m.

N (inverse value): 65537	Inverse of 65537 mod 1034776851837418226012406113933120080 Result: 568411228254986589811047501435713
M (mod value): 10347768518374182260	
<input type="button" value="Determine"/>	
<input type="button" value="Try an example"/>	
<ul style="list-style-type: none"> <li>• Inverse of 53 mod 120. <a href="#">Test</a></li> <li>• Inverse of 65537 mod 1034776851837418226012406113933120080. <a href="#">Test</a></li> </ul>	

```
rsa.py      inverse.py ×

inverse.py > ...
32     assert (n * x + p * y) % p == gcd
33
34     if gcd != 1:
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=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
ameya@ameya-HP-Pavilion-Gaming-Laptop-15-dk0xxx:~/Ameya-Ubuntu/SPIT LAB SUBMISSIONS/SEM 5/css-exp4/task_e$ python3 inverse.py
Inverse of 65537 mod 1034776851837418226012406113933120080
Result: : 568411228254986589811047501435713
```

```

rsa.py inverse.py
inverse.py > ...
30     """
31     gcd, x, y = extended_euclidean_algorithm(n, p)
32     assert (n * x + p * y) % p == gcd
33
34     if gcd != 1:
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
ameya@ameya-HP-Pavilion-Gaming-Laptop-15-dk0xxx:~/Ameya-Ubuntu/SPIT LAB SUBMISSIONS/SEM 5/css-exp4/task_e$ python3 inverse.py
Inverse of 53 mod 120
Result: : 77

```

### E.3) OUTPUT :

```

rsa.py e3.py inverse.py
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
ameya@ameya-HP-Pavilion-Gaming-Laptop-15-dk0xxx:~/Ameya-Ubuntu/SPIT LAB SUBMISSIONS/SEM 5/css-exp4/task_e$ python3 e3.py
b'-----BEGIN RSA PRIVATE KEY-----\nMIIEowIBAKCAQEAtheadR3y6yzDr/mEV+aIDUf60e0nHZENl2REuMjmt6xCf2o\n|ncf81AFZ3FGwAWeTtnHS9CzRSRiLa2KIHsvLUVl+EbQa\n|vUmWw+nGgpGbYL+0E\\nZPiwu0syfpKnWp+jCNCzouFD1rcdBXs/Jw/nU55qGnd+2LMbIw34Z6jrsyVWHTg\\nBVo2wxLHP/VbGHSIHQtu2bWCtzSaPKnLvguUZE9uW3l2c\n|j20GkadwVQ3\n|02/m/\\nQv1qj0t3wsJE+y8bvQcNDLpkUY05fj0thvgIv+5vx9640FkznLea3l8AS/lMw\\ndgYxE90o8t186iC8wF/khpyesGYlQc0v32XAwIDAQABaoIBAf5Fg\n|ov6HJecCSx\\niqlc5iLDs\n|m0UrRx4reBAvUyG7rAEIKj7U61a9BL6B5tn2GeMVLknkkG3B+CX2F3BKvnzje4oGGPqAnhzqvigOUJE\n|xF1vsLhLGVi4W/C0n0up4l9fDR2ACvs+19L3HVYak0nZ7xf6N1Fe133SmMz/0x1\n|Acj5/mnAeCx/AlmaC8F9G+3L7K0jrp0Bsff1q67sds/n/4znh8b4CN3EJEv0208J5Y0hRUMjB7E32qK8uShyoH1G0TA91SHL8/NUVVkl\\nB1/600EzR1zx\n|p47AR3rbBaAzu7ynwLjxkuRAkv/RgKoY0h+MxEtDCFzGsuVrwmxoi\\nqFCKLDEcgYEAtjRG22V7ZZEXBxx1NR9qLngKx8hwG1ZKksbmTij\n|o2RUQf8MKDTQ\\nrk7IuEPGn1me/5PjghDhvLi\\ze+RjSzL05YR7izJ7uTYUOKCGM0j\n|p4ey5EioVy9w\\nQtU55/DYhZg/VdZG6ZFw8c/NudruCHe\n|rtgdYQsX83oe\n|A/fKlwLvcEcgYEA/86Vn3e0yJm1K83AE04IDANHcMeq8LMtcJ0rg+F+Rk3JF0REedgmA3B7HFthuMwLox1\\n\n|a5LxwqnaQa/ex43L4d24Yq/9jTq2o0YpgETMgsY0HQ7fsb9jrlaURDn+3boPHRRnyM\n|+0AkTj44utjeEJWxAcxe511ZC41b6v\n|iDTMcgYEp9nv/Xq2dNF2Ep)f5U6w\\nCwK8p/Vq00zeywa2dtv7zqk2D402nx2j8r\n|h28iiEu140FwXy40FwS100MsxuLkh\\n295j0a2Lu26Teb2J8+G0xumZf+jlFpm\n|doBHGu4x2qwmvpj453+0N04fwwezs26X\\nra1+Y0AE0Dw2IAUthf47FSECgYB+MiEXAaqwd1ZaZap870+kWNBslhHG+u3g23in\\nL/gOnNhVlc9sNdx/Qa0/7m\n|9406509yaExhUxCBJ1l0t1MmuzFwX+0YmJQH+4TP1\\nuTQDRUezKcjM4vk+Pdg6zE2ifuanJYMFHtdLS\n|/Lep5FFNftB0MjJeA/+x+22Es/\\nJBFS5pwkBgCu0afFjBHS\n|TE65X6qsaAwLBSXPTraT6/WG6ib3egAiywN8g7blqPI\\nuByDy0V/fH5iv1s15icQc/7vphs+gDUAmoszTK4aJLTWuoi409mW2KJwdiPOE5\\nKmcGUv5\\BtdecUhj0KCT1\\n\\rXxUj+KjUy/m6gDqg0t8Xy1PAx\\n-----END RSA PRIVATE KEY-----'\nb'-----BEGIN PUBLIC KEY-----\\nMIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMII\n|BCgKCAQEAt\\nheaDr3y6yzTDr/mEV+a\\nIDUf60e0nHZENl2REuMjmt6xCf2ocf81AFZ3FGwAWeTtnHS9CzRSRiLa2KIHsvLUVl+EbQa\n|VW\\nWw+nGgpGbYL+0E\\nZPiwu0syfpKnWp+jCNCzouFD1rcdBXs/Jw/\\nmU55qGnd+2LMbIw34Z6jrsyVWHTg\\nBVo2wxLHP/VbGHSIHQtu2bWCtzSaPKnLvguUZE9uW3l2c\n|j20GkadwVQ3\n|02/m/\\nQv1qj0t3wsJE+y8bvQcNDLpkUY05fj0thvgIv+5vx9640FkznLea3l8AS/lMw\\ndgYxE90o8t186iC8wF/khpyesGYlQc0v32XAwIDAQABoIBAf5Fg\n|ov6HJecCSx\\niqlc5iLDs\n|m0UrRx4reBAvUyG7rAEIKj7U61a9BL6B5tn2GeMVLknkkG3B+CX2F3BKvnzje4oGGPqAnhzqvigOUJE\n|xF1vsLhLGVi4W/C0n0up4l9fDR2ACvs+19L3HVYak0nZ7xf6N1Fe133SmMz/0x1\n|Acj5/mnAeCx/AlmaC8F9G+3L7K0jrp0Bsff1q67sds/n/4znh8b4CN3EJEv0208J5Y0hRUMjB7E32qK8uShyoH1G0TA91SHL8/NUVVkl\\nB1/600EzR1zx\n|p47AR3rbBaAzu7ynwLjxkuRAkv/RgKoY0h+MxEtDCFzGsuVrwmxoi\\nqFCKLDEcgYEAtjRG22V7ZZEXBxx1NR9qLngKx8hwG1ZKksbmTij\n|o2RUQf8MKDTQ\\nrk7IuEPGn1me/5PjghDhvLi\\ze+RjSzL05YR7izJ7uTYUOKCGM0j\n|p4ey5EioVy9w\\nQtU55/DYhZg/VdZG6ZFw8c/NudruCHe\n|rtgdYQsX83oe\n|A/fKlwLvcEcgYEA/86Vn3e0yJm1K83AE04IDANHcMeq8LMtcJ0rg+F+Rk3JF0REedgmA3B7HFthuMwLox1\\n\n|a5LxwqnaQa/ex43L4d24Yq/9jTq2o0YpgETMgsY0HQ7fsb9jrlaURDn+3boPHRRnyM\n|+0AkTj44utjeEJWxAcxe511ZC41b6v\n|iDTMcgYEp9nv/Xq2dNF2Ep)f5U6w\\nCwK8p/Vq00zeywa2dtv7zqk2D402nx2j8r\n|h28iiEu140FwXy40FwS100MsxuLkh\\n295j0a2Lu26Teb2J8+G0xumZf+jlFpm\n|doBHGu4x2qwmvpj453+0N04fwwezs26X\\nra1+Y0AE0Dw2IAUthf47FSECgYB+MiEXAaqwd1ZaZap870+kWNBslhHG+u3g23in\\nL/gOnNhVlc9sNdx/Qa0/7m\n|9406509yaExhUxCBJ1l0t1MmuzFwX+0YmJQH+4TP1\\nuTQDRUezKcjM4vk+Pdg6zE2ifuanJYMFHtdLS\n|/Lep5FFNftB0MjJeA/+x+22Es/\\nJBFS5pwkBgCu0afFjBHS\n|TE65X6qsaAwLBSXPTraT6/WG6ib3egAiywN8g7blqPI\\nuByDy0V/fH5iv1s15icQc/7vphs+gDUAmoszTK4aJLTWuoi409mW2KJwdiPOE5\\nKmcGUv5\\BtdecUhj0KCT1\\n\\rXxUj+KjUy/m6gDqg0t8Xy1PAx\\n-----END PUBLIC KEY-----'

```

```

rsa.py          e3.py      inverse.py
e3.py > ...
1  from Crypto.PublicKey import RSA
2
3  key = RSA.generate(2048)
4
5  binPrivKey = key.exportKey('DER')
6
7  binPubKey = key.publickey().exportKey('DER')
8
9  print(binPrivKey)
10 print(binPubKey)

PROBLEMS   OUTPUT   TERMINAL   DEBUG CONSOLE
ameya@ameya-HP-Pavilion-Gaming-Laptop-15-dk0xxx:~/Ameya-Ubuntu/SPIT LAB SUBMISSIONS/SEM 5/css-exp4/task_e$ python3 e3.py
b'0\x82\x04\x45\x02\x01\x00\x02\x82\x01\x01\x01\x00\xaf\xbf\x05I\x13r\xb5\xag\xc2;\xa5\x68\x08\xb1\'q"\xccl\xb7\xd2$\xbc\xf7\xe6#\xe5\x99qD\xbb\x
fe\xb\xb8@:\xb5\x97<t\xe5\x9c\x1f\xad\xf2:\x16\xd50\x1b\xce+d\x14\x9a\x9\xbf\x99\xd2->ID\xe6\xfa\x1c\x08L\xc3\xf8K\xc2 \xd5\xaf\x9a\x8b\x81\x95\x
c2\x1dK\xbd\xd7\xfc\x9f\xbaC\x8c\x81\xd5\xc2\xaf\x0\x86T\x06p\\\'x03\xc\xdb\x90\x4Juv\xc3\xed\x10\xdc\x9f\x8c\xda\xb6Co\x80\x1f\x9f\xb3\xacN\x
e4w\xc2\x7f\xc7\x03Y\x13V\x08\x9f"+\x0\xe0\xF\xed \x1f\xe3\x0\xf\xde\xdc\x0\xeh\x82\x1\x9a\x12\xc9\xd0\x89\xda\x9a\xf0\x07\x9c\xd5\xbe\x0\x\x
f2\x0\xf\xe1\xc8L\xea\xd16M"\xb8v\xab\xaf\xed\xee\x19oC\xd3\x01\xe57\x18\x98\x98E\xeb\xb9\xfa\x98h\x83A%\x05n-\xfdB\xab\xf95\xb68\x5\xt1\x9
8\x16\xbd\xba\xf3\xfa\x0\x80\xt\xae\x2l\x83\xf0\xec\xb4\xb1\x02\x03\x01\x00\x01\x02\x82\x01\x00\x04\x1dh\x8b\x97\x11\xdd\x1b*\x92d\xf8\xfc\xd7\

```

**DER (Distinguished Encoding Rules)** is the method of encoding the data that makes up the certificate. DER itself could represent any kind of data, but usually it describes an encoded certificate or a CMS container whereas **PEM (Privacy Enhanced Mail)** is a method of encoding binary data as a string (ASCII armor). It contains a header and a footer line (specifying the type of data that is encoded and showing begin/end if the data is chained together) and the data in the middle is the base 64 data. PEM stands for Privacy Enhanced Mail; mail cannot contain un-encoded binary values such as DER directly.

## F) PGP

### F.1) OUTPUT :

The following is a PGP key pair. Using <https://asecuritysite.com/encryption/pgp>, can you determine the owner of the keys:

-----BEGIN PGP PUBLIC KEY BLOCK-----

Version: OpenPGP.js v4.4.5

Comment: <https://openpgpjs.org>

xk0EXE0YvQECAIpLP8wfLxzgc01MpwgzcUzTlH0icgg0IyuQKsHM4XNPugzUX0NeawrJhfi+f8hDRojJ5Fv8jBI0m/KwFMNTT8AEQEAc0UYmlsbCA8Ymls  
bEBob211LmNvbT7CdQ00AQgAHwUCXE0YvQYLCQcIAwIEF0gKAgnMWAgECGQEC  
GwMCHgEACgkQoNsXEDYt2ZjkTAH/b6+pDfQLi6zg/Y0tHSSPPRvL323cwoay  
vMcPjrnWq+VFiNyXzY+UJKR1PXskzDvHML0yVpUcj1e5ChyT5L0w/ZM5NBfxD  
mL0BAGdY1TsT06vVQxu3jmflzKMAr4kLqqIuFFRCapRuHYLOjwLgJZS9p0bF  
S0qS8zMEGpN90ZxkG8YEch3gHx1rvALTABEAAHCXwQYAQgACQUCXE0YvQIB  
DAAKCRCg2xcQNi3ZmMAGAf9w/XazFELDGLW3512zwL2rKwM7rK97aFrTxz5W  
XwA/5gqoVP0iQxk1b9qpX7RVd6rLku7zoX7F+sQodlsCWrMw=cXT5

-----END PGP PUBLIC KEY BLOCK-----

-----BEGIN PGP PRIVATE KEY BLOCK-----

Version: OpenPGP.js v4.4.5

Comment: <https://openpgpjs.org>

xBmBFxDmL0BAgCKSz/MHy8c4HKJTKcIM3FM05R9InIIDiMrkCrBzOFzT7oM  
1F9DXmmsKyYX4vn/IQ0aIyeRb/IwSNJvysBTDU0/ABEBAAH+CQMIBNTT/OPv  
TJzgvF+fLOsLsNYP64QfNHav5O744y0MLV/EZT3gsBwO9v4XF2SsZj6+EHb  
k09gWi31BAIDgSaDsJYf7xP0hp8iEWlwrrUkC+j1GpdTsGDjpeYMIsvVv8Yc  
am0g7MSRsL+dYQauIgtVb3d1oLMPtuL59nVAYuIgD8HXyaH2vsEgSZSQn0kf  
fvF+dWeqJxwFM/uX5PVKcuYsroJFBEO1z as4ERfxbbwns0gNHpjdiPu  
eHx6/4E0b1kmh0d6UT7B amubY7bcma1PB Sv8PH31Jt8SzRRiaWxsIDxiaw  
xsQGhvbWUuY29tPsJ1BBABCAsBQJcQ5i9BgsJBwgDAgQVCAoCAxYCAQIZAQ  
IbAwTeAQAKCRCg2xcQNi3ZmMAGAf9v6kN9AuLr0D9jS0dlk89G/XfbdzChr  
K8xw+0dar5V+I3Jfnj5QkpHU9eyTM08cws7JWlRy0V7kKHJPks7D9kx88m  
BFxDmL0BAGdY1TsT06vVQxu3jmflzKMAr4kLqqIuFFRCapRuHYLOjw1gJZ  
S9p0bFS0qS8zMEGpN90ZxkG8YEch3gHx1rvALTABEAAH+CQM2Gyk+BqV0  
gzgZX3C80JRLBRMT4sLCHOUGlwaspe+qat0VjeEuxA5DuSs0bVMrw7mJY  
QZLtjNkFAT921SwfxYgavS/bIL1w3QGA0CT5mqijKr0nurKkekKBD  
SGjkjVbIoPLMYHfe pPOju1322Nw4V3JQ04LBh/sdgGbRnwW3LHEK  
4Qe70ciuert8C+S5xfG+T5RWADi5HR8uUTyH8x1h0Zr0F7K0Wq4  
UcNvrUm6c35H61C1C4Zaar4JSN8fZPqVKL1HTVcL91pDzX  
xqXkjS05KXXZBh5w18EGAEIAAkFA1xDmL0CGwwACgkQoNsXEDYt2ZjkA  
BgH/cP12s3xCwxvtVt+Zds8NdqysD06yve2ha7cc+V18AP+YKqFT9Ik  
MZJW/aqV+0VXeqyyru86F+xfrEKHdbA1qzMA== =5NaF

-----END PGP PRIVATE KEY BLOCK-----

Version: OpenPGP.js v4.4.5

Comment: <https://openpgpjs.org>

xBmBFxDmL0BAgCKSz/MHy8c4HKJTKcIM3FM05R9InIIDiMrkCrBzOFzT7  
oM  
1F9DXmmsKyYX4vn/IQ0aIyeRb/IwSNJvysBTDU0/ABEBAAH+CQMIBNTT/  
OPv  
TJzgvF+fLOsLsNYP64QfNHav5O744y0MLV/EZT3gsBwO9v4XF2SsZj6+EH  
b  
k

O9gWi31BAIDgSaDsJYf7xPOhp8iEWWwrUkC+jIGpdTsGDJpeYMIsvVv8Yc  
am  
0g7MSRsL+dYQaulgtVb3dloLMPtuL59nVAYuIgD8HXyaH2vsEgSZSQn0kfV  
F  
+dWeqJxwFM/uX5PVKcuYsroJFBEO1zas4ERfxbbwnsQgNHpjdlpueHx6/4  
EO  
b1kmhOd6UT7BamubY7bcma1PBSv8PH31Jt8SzRRiaWxsIDxiaWxsQGhv  
bWUu  
Y29tPsJ1BBABCAAfBQJcQ5i9BgsJBwgDAgQVCAoCAxYCAQIZAQIbAwleA  
QAK  
CRCg2xcQNi3ZmORMAf9vr6kN9AuLrOD9jS0dLk89G/XfbdzChrK8xw+Od  
ar5  
V+I3JfNj5QkpHU9eyTMO8cws7JWIRyOV7kKHJPks7D9kx8BmBFxDmL0B  
AgDY  
ITsT06vVQxu3jmflzKMAr4kLqqluFFRCapRuHYLOjw1gJZS9p0bFS0qS8zM  
E  
GpN9QZxkG8YEch3gHxIrvALtABEBAAH+CQMI2Gyk+BqVOgzgZX3C80JRL  
BRM  
T4sLCHOUGlwaspe+qatOVjeEuxA5DuSs0bVMrw7mJYQZLkjNkFAT92ISwf  
xY  
gavS/bILLw3QGA0CT5mqijKr0nurKkekKBDSGjkjVbIoPLMYHfepPOju1322  
Nw4V3JQO4LBh/sdgGbRnwW3LhHEK4Qe70cuiert8C+S5xfG+T5RWADi5  
HR8u  
UTyH8x1h0ZrOF7K0Wq4UcNvrUm6c35H6IClC4Zaar4JSN8fZPqVKLIHTVc  
L9  
lpDzXxqxKjS05KXXZBh5wl8EGAEIAAkFAIxDmL0CGwwACgkQoNsXEDYt2Z  
jA  
BgH/cP12s3xCwxtVt+Zds8NdqysDO6yve2ha7cc+VI8AP+YKqFT9IkMZJW/  
a qV+0VXeqyyru86F+xfrEKHdbAlqzMA== =5NaF

-----END PGP PRIVATE KEY BLOCK-----

PGP encryption allows for the encryption of email and with a signature from the sender. Enter the receiver's public key here:



**PGP key**

You can enter your own key here by pasting it in, or use Bill's key:

```
gWnICngCAGyKQUNsXcU1TlzzJKTAn/ uotpuv1qLlozg/TWLn5oPPRVI5Z5cwuay  
vMcPjnWq+VfiNyXZY+UJKR1PXskzDvHML0yVpUcjle5ChyT5L0w/ZM5NBFXD  
mL0BAgDYlTsT06vVQxu3jmflzKMAR4kLqqIuFFRCapRuHYL0jw1gJZS9p0F  
S0qs8zMEGpN9QZxkG8YEch3gHxlrvtABEBAAHCxwQYAQgACQUCE0YvQIb  
DAAKCRCg2xcQNj3zmMAGAf9w/XazfELDG1W35l2zw12rKwM7rK97aFrTxz5W  
XwA/5gqoVP0iQxk1b9qpX7RVd6rLKu7zoX7F+sQod1scWrMw =cXT5
```

Version:

User ID:

Key ID (8 bytes in hex):

Public Key type and values:

## F.2) OUTPUT :

**Using the code at the following link, generate a key:**

<https://asecuritysite.com/encryption/openpgp>

[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: ameya  
Email: ameya.jangam@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

xk0EYbODowEB/3J08XVJ0hoaXB0@jS4hs9paUJ8cx2hCnWEaqJs/4707F3gz
cg0WrGhuq1yr6o0tsxiy0+I5M!6ne1xCdk48sAE0EAc0UYm1sbCa8Ym1s
b6Bob21LLmNbT7CdQQAQgAHwUCybODowYLQcCIAwIEFQgKAghMwAgECQ0EC
GwMCQgEACgKQ4ja4R4mLTaxogH/YJ28RZj6jyBySpoi7bluoLsgJ6FQH1Ao
DRLV8Wk1lmoHXhXf6gjzjumptUCSG01q14oxSeafkxGFp+HY7bbM5NBGz
g6MBAgC53G+jG3FHG71uFXufFu4/vIwr0kBvE8p9bTu4vMgHxAAT28DlRhisi
tY5jLB2zFDLwOrJL13RD4Bg9v5y5pH/1ABEBAAHCxwQYAqgACQUCYbODowIb
DAAKCRD1NrHh1YPTuqnAF9EtQm9SnqQeCjVx0KH+Pyslc5YpB9vgWlh1TL
uuqV18GDyjbDrsoqt+ML88AxhApplJrzhY4nyTqROX0walip
=bxN
-----END PGP PUBLIC KEY BLOCK-----

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

xcBmBGGzg6MBAf9yTvF1SToaGlwdNI0uIbPawlCfHMDoQp1hGq1bP+090xd4
M3IKFqxh7qomk+oPbbF4l8jtP10TK+p3pcQnSuPLABEAAH+CQMIdv26Ed7o
GRXg3uuWirlszaBt8XXTtyy0lBSXPpCLR2PfA/bmohViIgR3s8tAKxKALbwB
NRT96nN47aqkgGeengZmqKw65NrbmMNvxh89Eke0Uouqlq210+S+yGsuAxu
txvQUlP5pM8xDUgMzKhIwMTFSK0tYkj4Ufeg4rvLV9axxBF+9377vcrBqS4
qbQCw0qSFwj15hnBNd45LeBespLU/b6jPJk/UEgtOwiGg7ox1PxZGd+GzP4
zB1Zp+6dGSwbb3cr0xdPeB6RKZNxpxzWkjzR1awxsIDxiawxsGhvbwUu
Y29tPsJ1BABCAAfBqJhs40jBgsJBwgDAgQVCAoCaxYCAQIZAQ1bAwIeAQAK
CRDiNrhHiYtPtrGiaf9gnbx+mPSPIHJI-gj9su6gvAnoVAFUcGNetUFYqMw
agdeFch/qD006am1QjIbTwqXjFIRoWTEYU/4etjttuEx8BmGGzg6MBAgC5
```

### F.3) OUTPUT :

Create a key pair with (RSA and 2,048-bit keys):

**gpg --gen-key**

```
ameya@ameya-HP-Pavilion-Gaming-Laptop-15-dk0xxx:~/Ameya-Ubuntu/SPIT LAB SUBMISSIONS/SEM 5/css-exp4$ 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: Ameya
Email address: ameya.jangam@spit.ac.in
You selected this USER-ID:
    "Ameya <ameya.jangam@spit.ac.in>

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 25F80F2C5C81CDA7 marked as ultimately trusted
gpg: directory '/home/ameya/.gnupg/openpgp-revocs.d' created
gpg: revocation certificate stored as '/home/ameya/.gnupg/openpgp-revocs.d/FBDBBD1270CAF AE8318734982
5F80F2C5C81CDA7.rev'
public and secret key created and signed.

pub    rsa3072 2021-12-06 [SC] [expires: 2023-12-06]
      FBDBBD1270CAF AE83187349825F80F2C5C81CDA7
uid            Ameya <ameya.jangam@spit.ac.in>
sub    rsa3072 2021-12-06 [E] [expires: 2023-12-06]
```

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`

```
ameya@ameya-HP-Pavilion-Gaming-Laptop-15-dk0xxx:~/Ameya-Ubuntu/SPIT LAB SUBMISSIONS/SEM 5/css-exp4$ gpg --export -a "Ameya" > mypub.key
ameya@ameya-HP-Pavilion-Gaming-Laptop-15-dk0xxx:~/Ameya-Ubuntu/SPIT LAB SUBMISSIONS/SEM 5/css-exp4$ gpg--export-secret-key -a "Ameya" > mypriv.key
gpg--export-secret-key: command not found
ameya@ameya-HP-Pavilion-Gaming-Laptop-15-dk0xxx:~/Ameya-Ubuntu/SPIT LAB SUBMISSIONS/SEM 5/css-exp4$ gpg --export-secret-key -a "Ameya" > mypriv.key
ameya@ameya-HP-Pavilion-Gaming-Laptop-15-dk0xxx:~/Ameya-Ubuntu/SPIT LAB SUBMISSIONS/SEM 5/css-exp4$
```

How is the randomness generated?

**PGP generates a session key, a secret key that can only be generated once. When you hit this key, the movement of your cursor and the keystrokes you make generate a random number. The plaintext is encrypted with this session key using a symmetric encryption approach that is highly safe and quick, yielding ciphertext.**

Outline the contents of your key file:

**Both the files contain a header and footer signifying that they are either PGP Public or Private key blocks and the content between them is the actual key.**

```
ameya@ameya-HP-Pavilion-Gaming-Laptop-15-dk0xxx:~/Ameya-Ubuntu/SPIT LAB SUBMISSIONS/SEM 5/css-exp4$ cat mypriv.key
-----BEGIN PGP PRIVATE KEY BLOCK-----
lQWFBGGuG+sBDADEmMM/SRfV0cP9/zactw5VTN/FY79QVsW5HZteFIKGtBK14Wr7
e0V2aMRemlt9rbfuqlp3Y84bcNCkrV4SLu2vEBDNRzr0yPm4lIOLViD7jt0Xtiah
IFMxi2sG7Ex4ANDo3owXJjcHShJTMdUtanjaDljGA+e/l0ziN+i0GDzvRsJXkycA
tUEp2hV+MXWJcb2dEW3Z8ujRyfXPfkPrG9RE7FjEz3GJC/E5wEBUt6Ch0FrnJzu/
-----END PGP PRIVATE KEY BLOCK-----

ameya@ameya-HP-Pavilion-Gaming-Laptop-15-dk0xxx:~/Ameya-Ubuntu/SPIT LAB SUBMISSIONS/SEM 5/css-exp4$ cat mypub.key
-----BEGIN PGP PUBLIC KEY BLOCK-----
mQGNBGGuG+sBDADEmMM/SRfV0cP9/zactw5VTN/FY79QVsW5HZteFIKGtBK14Wr7
e0V2aMRemlt9rbfuqlp3Y84bcNCkrV4SLu2vEBDNRzr0yPm4lIOLViD7jt0Xtiah
IFMxi2sG7Ex4ANDo3owXJjcHShJTMdUtanjaDljGA+e/l0ziN+i0GDzvRsJXkycA
tUEp2hV+MXWJcb2dEW3Z8ujRyfXPfkPrG9RE7FjEz3GJC/E5wEBUt6Ch0FrnJzu/
-----END PGP PUBLIC KEY BLOCK-----
```

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):

Now , list the keys

```
ameya@ameya-HP-Pavilion-Gaming-Laptop-15-dk0xxx:~/Ameya-Ubuntu/SPIT LAB SUBMISSIONS/SEM 5/css-exp4/t
ask_f$ gpg --list-keys
/home/ameya/.gnupg/pubring.kbx
-----
pub rsa4096 2016-11-11 [SC]
    7D2BAF1CF37B13E2069D6956105BD0E739499BDB
uid      [ unknown] Piotr Kuczynski <piotr.kuczynski@gmail.com>
sub    rsa4096 2016-11-11 [E]

pub rsa4096 2014-10-28 [SC]
    409B6B1796C275462A1703113804BB82D39DC0E3
uid      [ unknown] Michal Papis (RVM signing) <mpapis@gmail.com>
uid      [ unknown] Michal Papis <michal.papis@toptal.com>
uid      [ unknown] [jpeg image of size 5015]
sub    rsa2048 2015-11-02 [E]
sub    rsa4096 2014-10-28 [S] [expires: 2022-03-12]

pub rsa3072 2021-12-06 [SC] [expires: 2023-12-06]
    FBDBBD1270CAF8E83187349825F80F2C5C81CDA7
uid      [ultimate] Ameya <ameya.jangam@spit.ac.in>
sub    rsa3072 2021-12-06 [E] [expires: 2023-12-06]

pub rsa4096 2021-02-17 [SC] [expires: 2025-02-17]
    8816FCA02D2FC9253253F6F648AAE36CA93AE3D8
uid      [ unknown] Bill Buchanan <B.Buchanan@napier.ac.uk>
sub    rsa4096 2021-02-17 [E] [expires: 2025-02-17]

pub rsa3072 2021-12-06 [SC] [expires: 2023-12-06]
    0B62119F46E6D43861DDCE545B7A8461A2AB2688
uid      [ultimate] Gajendra <gajendrajangam@yahoo.co.in>
sub    rsa3072 2021-12-06 [E] [expires: 2023-12-06]
```

**Which keys are stored on your key ring and what details do they have?**

It displays the UID,email and the public key of a user and also displays the expiry date of the key.

**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
```

```
ameya@ameya-HP-Pavilion-Gaming-Laptop-15-dk0xxx:~/Ameya-Ubuntu/SPIT LAB SUBMISSIONS/SEM 5/css-exp4/task_f
ameya@ameya-HP-Pavilion-Gaming-Laptop-15-dk0xxx:~/Ameya-Ubuntu/SPIT LAB SUBMISSIONS/SEM 5/css-exp4/task_f$ cat hello.txt
Yo,what is up?
I am currently watching a show named 'Peaky Blinders' that you should definitely watch
```

```
ameya@ameya-HP-Pavilion-Gaming-Laptop-15-dk0xxx:~/Ameya-Ubuntu/SPIT LAB SUBMISSIONS/SEM 5/css-exp4/task_f$ gpg -e -a -u "Ameya" -r "Gajendra" hello.txt
gpg: checking the trustdb
gpg: marginal 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
File 'hello.txt.asc' exists. Overwrite? (y/N) y
```

### **What does the –a option do:**

It creates the ASCII armored output.

### **What does the –r option do:**

Encrypts for user id name.

### **What does the –u option do:**

Use name as the key to sign with.

### **Which file does it produce and outline the format of its contents:**

It produces a .asc file which has a header and a footer indicating a pgp file.

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:

Now, create a file (such as hello.asc) and decrypt the email using the public key received from them

**gpg -d hello.asc >hello-decrypted.txt**

The terminal window displays the GPG man page, which includes options for importing keys, changing card PINs, and setting TOFU policies. It also lists various command-line options such as --armor, --recipient, and --output. A modal dialog box titled "Passphrase:" is overlaid on the terminal, prompting for a passphrase to unlock a specific RSA key. The dialog shows the key's details: "Gajendra <gajendrajangam@yahoo.co.in>" and its creation date "2021-12-06". The user has entered a five-dot passphrase ("\*\*\*\*\*") in the input field. Below the input field is a checkbox labeled "Save in password manager". At the bottom of the dialog are "Cancel" and "OK" buttons.

```

ameya@ameya-HP-Pavilion-Gaming-Laptop-15-dk0xxx:~/Ameya-Ubuntu/SPIT LAB SUBMISSIONS/SEM 5/css-exp4/task_f
File Edit View Help
Default Style
--import import/merge keys
--card-status print the card status
--edit-card change data on a card
--change-pin change a card's PIN
--update-trustdb update the trust database
--print-md print message digests
--server run in server mode
--tofu-policy VALUE set the TOFU policy for a key

Options:
-a, --armor create ascii arm
-r, --recipient USER-ID encrypt for USE
-u, --local-user USER-ID use USER-ID to
-z N set compress level
--textmode use canonical t
-o, --output FILE write output to
-v, --verbose verbose
-n, --dry-run do not make any
-i, --interactive prompt before o
--openpgp use strict Open

(See the man page for a complete listing of

Examples:
-se -r Bob [file] sign and encrypt for user Bob
--clear-sign [file] make a clear text signature
--detach-sign [file] make a detached signature
--list-keys [names] show keys
--fingerprint [names] show fingerprints

Please report bugs to <https://bugs.gnupg.org>.
ameya@ameya-HP-Pavilion-Gaming-Laptop-15-dk0xxx:~/Ameya-Ubuntu/SPIT LAB SUBMISSIONS/SEM 5/css-exp4/task_f$ ls
bill.key hello.txt hello.txt.asc mypriv.key mypub.key
ameya@ameya-HP-Pavilion-Gaming-Laptop-15-dk0xxx:~/Ameya-Ubuntu/SPIT LAB SUBMISSIONS/SEM 5/css-exp4/task_f$ gpg hello.txt.asc
gpg: WARNING: no command supplied. Trying to guess what you mean ...

```

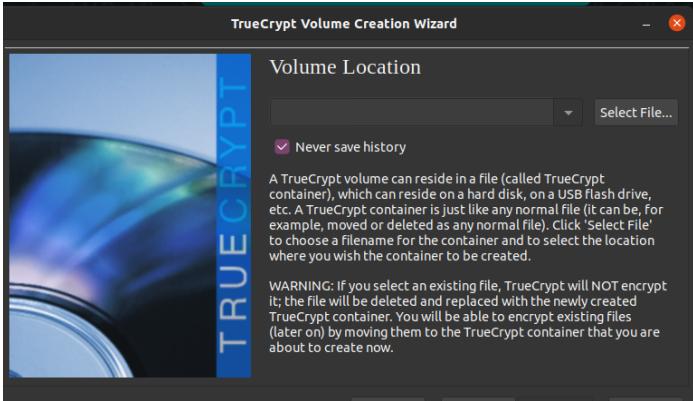
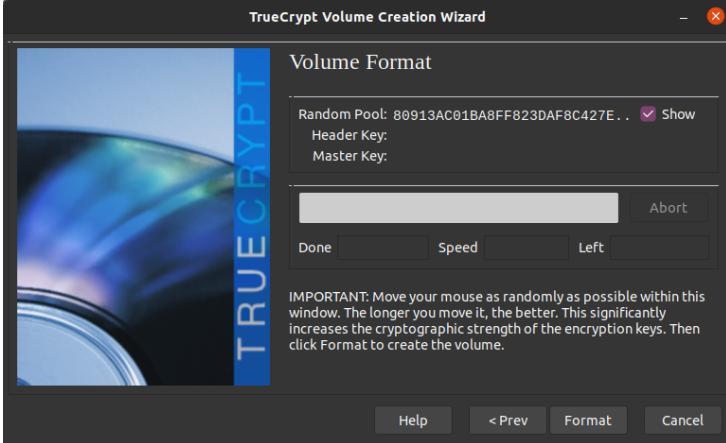
```

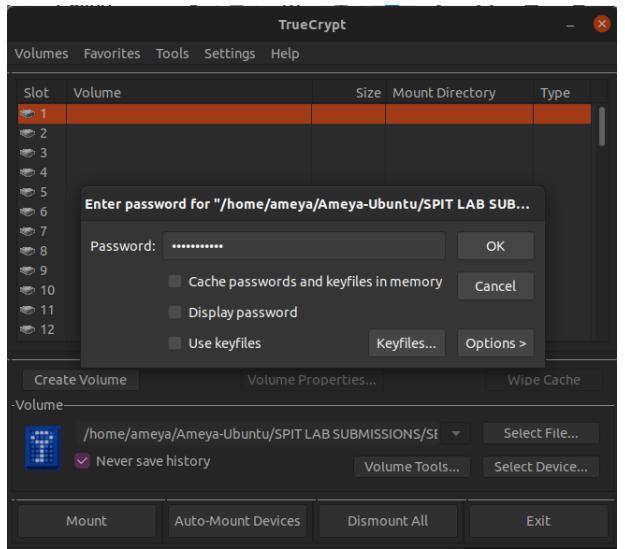
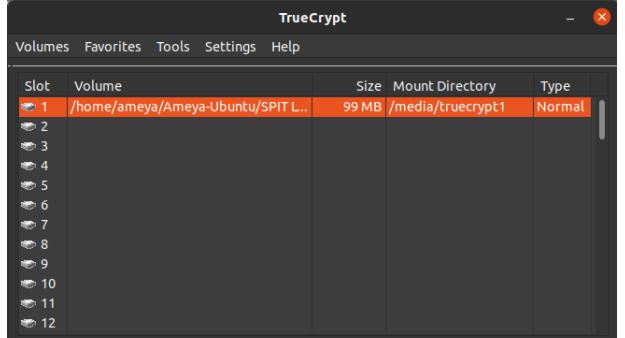
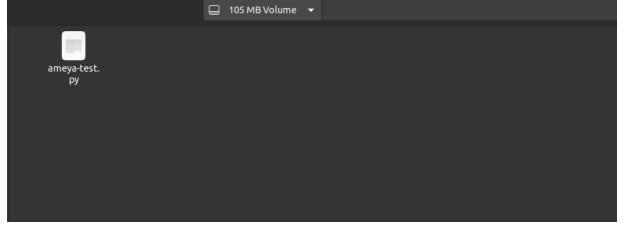
ameya@ameya-HP-Pavilion-Gaming-Laptop-15-dk0xxx:~/Ameya-Ubuntu/SPIT LAB SUBMISSIONS/SEM 5/css-exp4/task_f$ gpg hello.txt.asc
gpg: WARNING: no command supplied. Trying to guess what you mean ...
gpg: encrypted with 3072-bit RSA key, ID 424F3DF233F1BED2, created 2021-12-06
    "Gajendra <gajendrajangam@yahoo.co.in>"
File 'hello.txt' exists. Overwrite? (y/N) y
ameya@ameya-HP-Pavilion-Gaming-Laptop-15-dk0xxx:~/Ameya-Ubuntu/SPIT LAB SUBMISSIONS/SEM 5/css-exp4/task_f$ cat hello.txt
Yo,what is up?
I am currently watching a show named 'Peaky Blinders' that you should definitely watch

```

**Can you decrypt the message:** Yes,we can

## 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_yourusername) with a Standard TrueCrypt volume.</p> <p>When you get to the Encryption Options, run the AES-Two-Separate benchmark tests and outline the results:</p> 	<p>CPU (Mean)  <b>AES: 10.8 GB/s</b>  <b>AES-Twofish: 1.1GB/s</b>  <b>AES-TwoFish-Serpent: 387 MB/s</b>  <b>Serpent-AES: 688 MB/s</b>  <b>Serpent: 738 MB/s</b>  <b>Serpent-Twofish-AES: 452 MB/s</b>  <b>Twofish: 1.3 GB/s</b></p> <p>Twofish-Serpent: <b>471 MB/s</b>  Which is the fastest:  <b>AES</b></p> <p>Which is the slowest:  <b>AES-Twofish-Serpent</b></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?</p> <p><b>The random pool basically keeps on capturing the mouse movements of the user continuously as it is also alerted on the screen that the user must move the mouse within the window as randomly as possible which helps in increasing the cryptographic strength of the encryption keys.</b></p> 

3	<p>Now mount the file as a drive.</p>	<p>Can you view the drive on the file viewer and from the console? Yes</p>   

		<pre>ameya@ameya-HP-Pavilion-Gaming-Laptop-15-dk0xxx:/media\$ ls ameya  truecrypt1 ameya@ameya-HP-Pavilion-Gaming-Laptop-15-dk0xxx:/media\$ ls -al truecrypt1/ total 5 drwx----- 2 ameya ameya 1024 Jan  1 1970 . drwxr-xr-x  4 root  root  4096 Dec 10 21:52 .. -rw-------  1 ameya ameya    0 Dec 10 21:53 ameya-test.py ameya@ameya-HP-Pavilion-Gaming-Laptop-15-dk0xxx:/media\$  </pre>
4	Create some files on your TrueCrypt drive and save them.	<p><b>Without giving them the password, can they read the file?</b></p> <p>No one can read the file without giving the correct password.</p> <p><b>With the password, can they read the files ?</b></p> <p>Yes, they can read the files.</p>

## 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?

**Because bitcoin addresses are the 256-bit SHA hash of an ECDSA public key, any flaws in those algorithms would be considered a flaw in bitcoin itself. Breaking this degree of encryption, on the other hand, would need a massive amount of computer power.**

**Coincidentally, it takes the same amount of processing power as bitcoin mining, and in virtually every situation, mining would be far more profitable than hacking. It needs a lot of computing power, but it's extremely doable with 1 Tera keys per second crackers.**