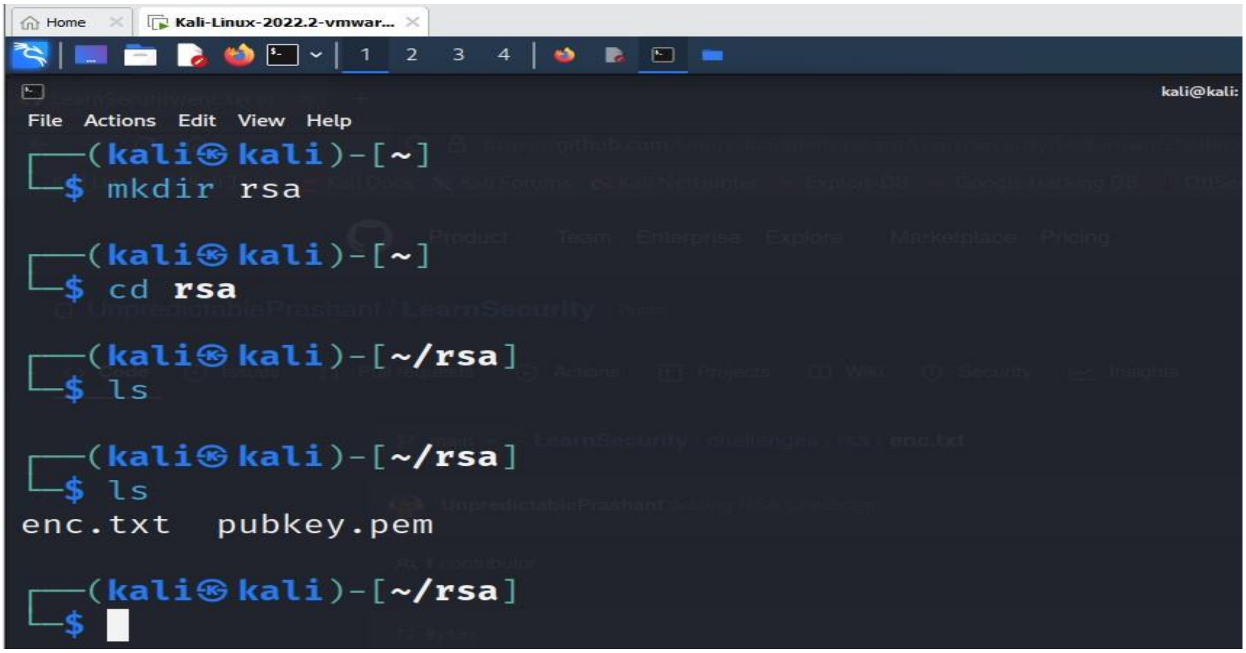
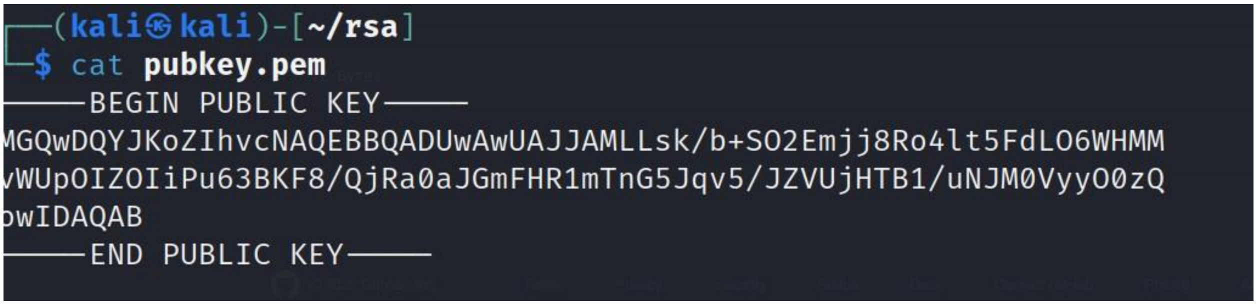
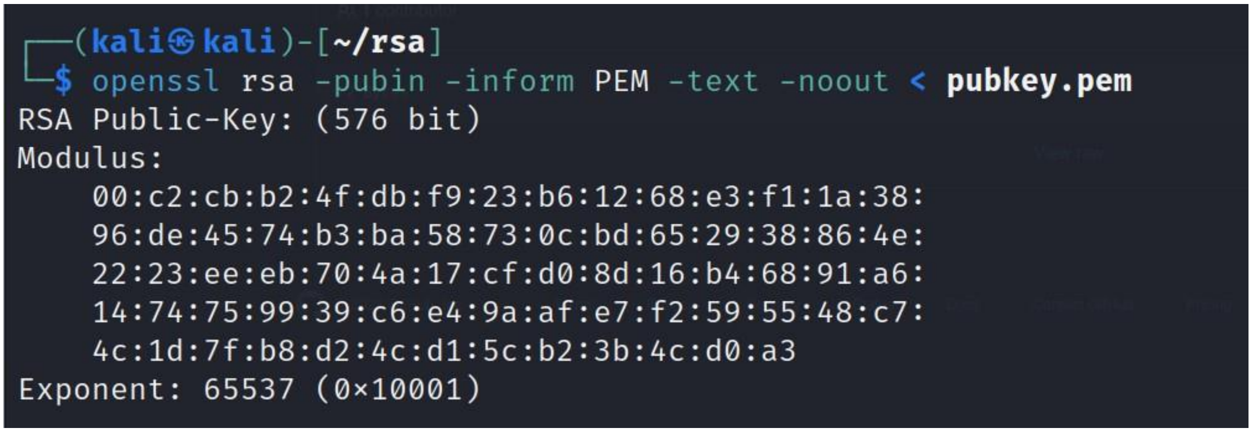
# Experiment 2: Implementation of Cryptanalysis using RSA.



```
┌──(kali㉿kali)-[~]
└─$ mkdir rsa

┌──(kali㉿kali)-[~]
└─$ cd rsa

┌──(kali㉿kali)-[~/rsa]
└─$ ls

┌──(kali㉿kali)-[~/rsa]
└─$ ls
enc.txt  pubkey.pem

┌──(kali㉿kali)-[~/rsa]
└─$ 
```



```
┌──(kali㉿kali)-[~/rsa]
└─$ cat pubkey.pem
─────BEGIN PUBLIC KEY─────
MGQwDQYJKoZIhvcNAQEBBQADUwAwUAJJAMLLsk/b+SO2Emjj8Ro4lt5FdLO6WHMM
vWUpOIZOIiPu63BKF8/QjRa0aJGmFHR1mTnG5Jqv5/JZVUjHTB1/uNJM0VyyO0zQ
owIDAQAB
─────END PUBLIC KEY─────
```

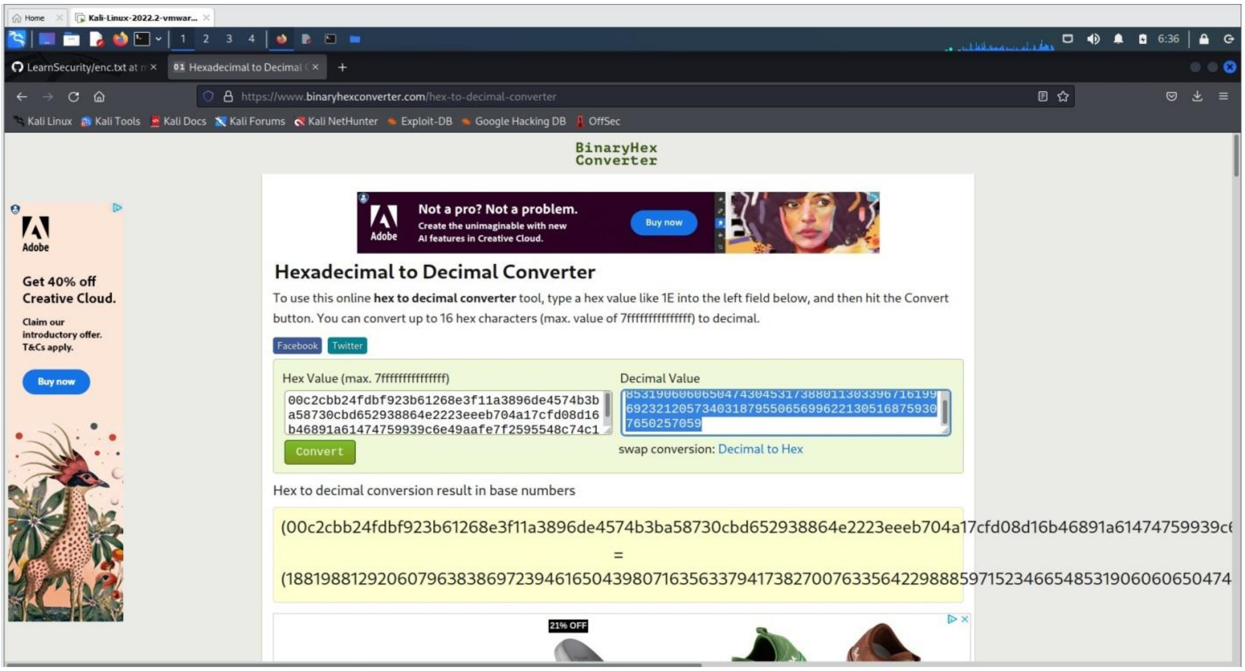


```
┌──(kali㉿kali)-[~/rsa]
└─$ openssl rsa -pubin -inform PEM -text -noout < pubkey.pem
RSA Public-Key: (576 bit)
Modulus:
    00:c2:cb:b2:4f:db:f9:23:b6:12:68:e3:f1:1a:38:
    96:de:45:74:b3:ba:58:73:0c:bd:65:29:38:86:4e:
    22:23:ee:eb:70:4a:17:cf:d0:8d:16:b4:68:91:a6:
    14:74:75:99:39:c6:e4:9a:af:e7:f2:59:55:48:c7:
    4c:1d:7f:b8:d2:4c:d1:5c:b2:3b:4c:d0:a3
Exponent: 65537 (0x10001)
```

**x**

Copy the hexadecimal decimal code into a notepad as n value. As it is a hexadecimal we can convert it into decimal for gaining the plaintext.

## Hexadecimal to decimal convertor



### Hexadecimal to Decimal Converter

To use this online **hex to decimal converter** tool, type a hex value like 1E into the left field below, and then hit the Convert button. You can convert up to 16 hex characters (max. value of 7fffffffffffffff) to decimal.

Hex Value (max. 7fffffffffffffff)

```
00c2cbb24fdbf923b61268e3f11a3896de4574b3b
a58730cbd652938864e2223eeeb704a17cfd08d16
b46891a61474759939c6e49aafe7f2595548c74c1
```

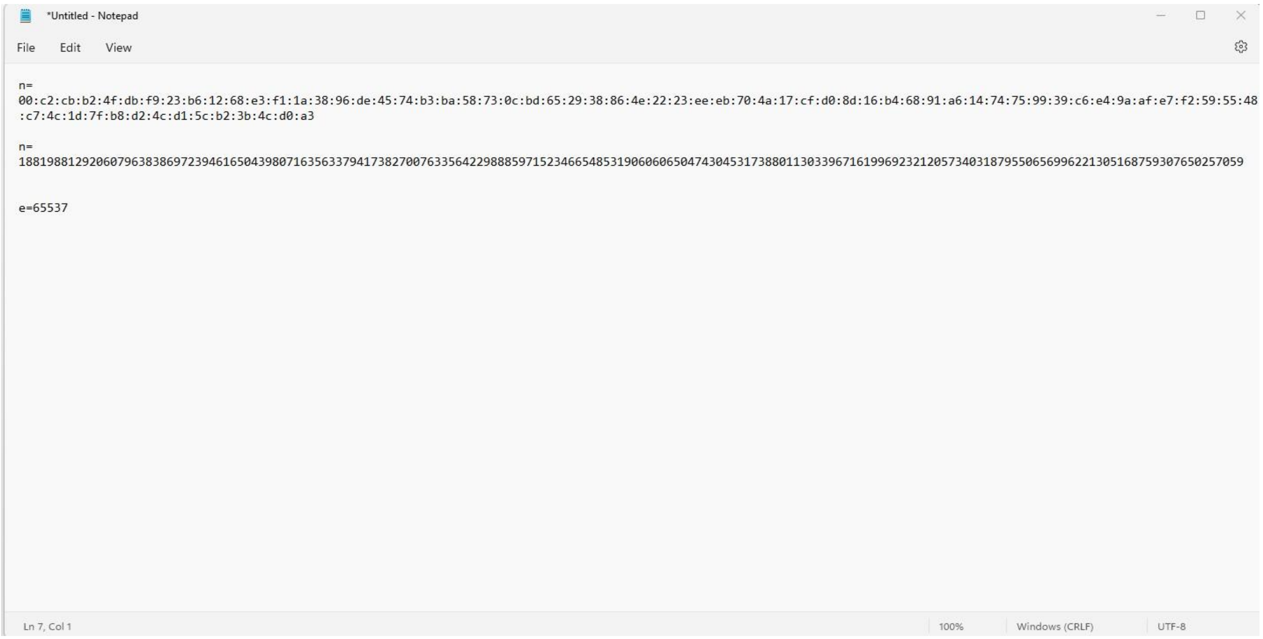
Decimal Value

```
853190606065047430453173880113033967161 99
69232120573403187955065699622130516875930
7650257059
```

Hex to decimal conversion result in base numbers

```
(00c2cbb24fdbf923b61268e3f11a3896de4574b3ba58730cbd652938864e2223eeeb704a17cfd08d16b46891a61474759939c(
=
(1881988129206079638386972394616504398071635633794173827007633564229888597152346654853190606060650474
```

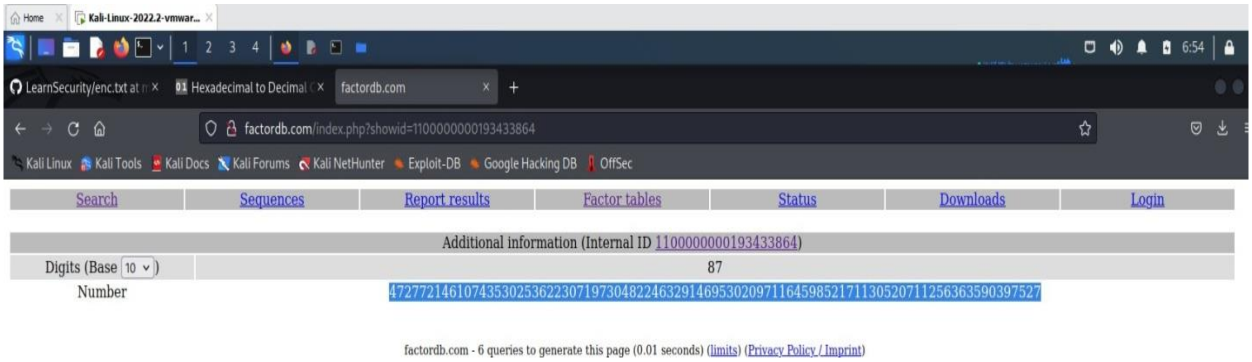
Paste the decimal code in the **notepad** as n value



```
n=
00:c2:cb:b2:4f:db:f9:23:b6:12:68:e3:f1:1a:38:96:de:45:74:b3:ba:58:73:0c:bd:65:29:38:86:4e:22:23:ee:eb:70:4a:17:cf:d0:8d:16:b4:68:91:a6:14:74:75:99:39:c6:e4:9a:af:e7:f2:59:55:48
:c7:4c:1d:7f:b8:d2:4c:d1:5c:b2:3b:4c:d0:a3

n=
1881988129206079638386972394616504398071635633794173827007633564229888597152346654853190606065047430453173880113033967161996923212057340318795506569962213051687593076502570 59


e=65537
```

Need to factorize n

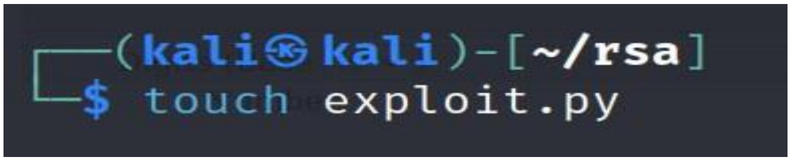
So goto website **factordb.com** click search, paste decimal value of n



| Additional information (Internal ID 1100000000193433864) | |
|---|---|
| Digits (Base 10) | 87 |
| Number | 472772146107435302536223071973048224632914695302097116459852171130520711256363590397527 |

factordb.com - 6 queries to generate this page (0.01 seconds) (limits) (Privacy Policy / Imprint)

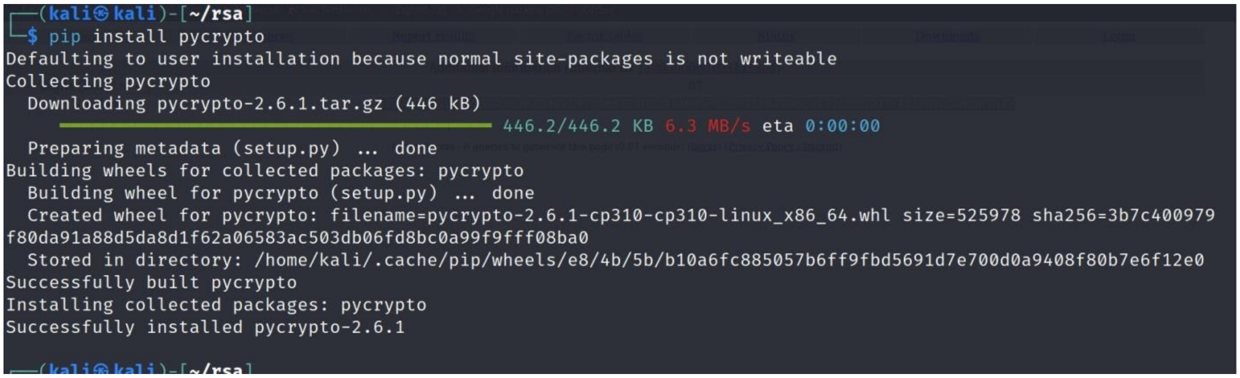
Create a exploit.py



```
┌──(kali㉿kali)-[~/rsa]
└─$ touch exploit.py
```

To install pycrypto

pip install pycryptodome



```
┌──(kali㉿kali)-[~/rsa]
└─$ pip install pycrypto
Defaulting to user installation because normal site-packages is not writeable
Collecting pycrypto
  Downloading pycrypto-2.6.1.tar.gz (446 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 446.2/446.2 KB 6.3 MB/s eta 0:00:00
  Preparing metadata (setup.py) ... done
Building wheels for collected packages: pycrypto
  Building wheel for pycrypto (setup.py) ... done
  Created wheel for pycrypto: filename=pycrypto-2.6.1-cp310-cp310-linux_x86_64.whl size=525978 sha256=3b7c400979
f80da91a88d5da8d1f62a06583ac503db06fd8bc0a99f9fff08ba0
  Stored in directory: /home/kali/.cache/pip/wheels/e8/4b/5b/b10a6fc885057b6ff9fbd5691d7e700d0a9408f80b7e6f12e0
Successfully built pycrypto
Installing collected packages: pycrypto
Successfully installed pycrypto-2.6.1

┌──(kali㉿kali)-[~/rsa]
```

Copy the code in the exploit.py file

and paste itfrom Crypto.PublicKey

import RSA

```
from Crypto.Util.number
import inverseimport base64
n =
1881988129206079638386972394616504398071635633794173827007633564229888
597152
3466548531906060650474304531738801130339671619969232120573403187955065699622
1305168759307650257059
e = 65537
p =
3980750864240649373971255005503864911990643623425267084063851895759463
889572
61768583317
q =
4727721461074353025362230719730482246329146953020971164598521711305207112563
7112563
63590397527
phi_n = (p
- 1)*(q -
1)d =
inverse(e,
phi_n)
key = RSA.construct((n,
e, d, p, q))fn =
"private.pem"
with open(fn,
        "wb") as f:
        f.write(key.e
        xportKey())
```

**Execute exploit.py file**

```
-->python exploit.py
```

**To decrypt the text**

```
-->openssl pkeyutl -decrypt -in encryptedFile -out decryptedFileName -inkey privateKey.pem
```

**Result:**

Thus the implementation of RSA algorithm was executed sucessfully.