CS 3710 Introduction to Cybersecurity
Term: Fall 2022

Lab Exercise 6 – Cryptography

Due Date: October 21, 2022 11:59pm
Points Possible: 7

Name: Jingtao Scott Hong jh4ctf

1. Overview

This lab exercise will provide some hands-on experience with symmetric and asymmetric encryption using command-line tools in Linux.

2. Resources required

This exercise requires Kali Linux VM running in the Virginia Cyber Range. Please log in at https://console.virginiacyberrange.net/.

3. Initial Setup

From your Virginia Cyber Range course, select the **Cyber Basics** environment. Click "start" to start your environment and "join" to get to your Linux desktop.

4. Tasks

Task 1: Symmetric Encryption with `mcrypt`

Mcrypt is a symmetric file and stream encryption utility for Linux and Unix that replaces the weaker **crypt** utility. Mcrypt can be used to encrypt files using several different symmetric encryption algorithms. By default it uses the Rijndael cipher, which is the algorithm on which the Advanced Encryption Standard (AES) is based.

Mcrypt is not installed by default on your virtual machine. Open a terminal and use the Linux package manager to install this software at the command line as follows (the second command may take a few minutes):

```
$ sudo apt-get update

$ sudo apt-get install mcrypt
```

Although we will be using mcrypt in default mode, it is very powerful and full-featured. To see all of the command-line options available to mcrypt, use the following command:

```
$ mcrypt --help
```

Mcrypt provides a variety of symmetric encryption techniques (you would use the **-m** option at the command line to access these). For a list of the various symmetric encryption modes available to mcrypt, use the following command:

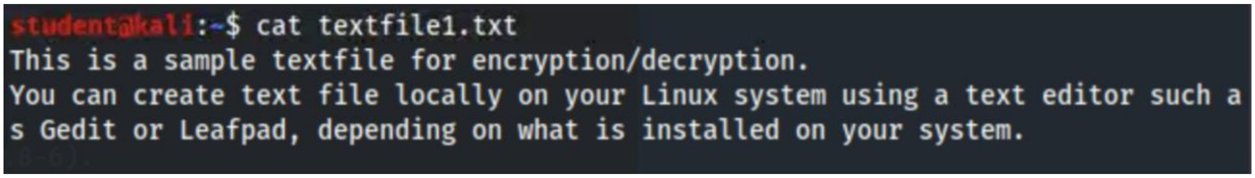
```
$ mcrypt --list
```

Next we need a file to encrypt. You can download a text file from the Virginia Cyber Range using the command below, or you can create a text file using a text editor (mousepad) on your Linux virtual machine and save it in your home directory.

```
$ wget artifacts.virginiacyberrange.net/gencyber/textfile1.txt
```

You can examine the contents of the file using the Linux **cat** command.

Question 1: CUT AND PASTE THE CONTENTS OF THE FILE HERE: (.5 point)



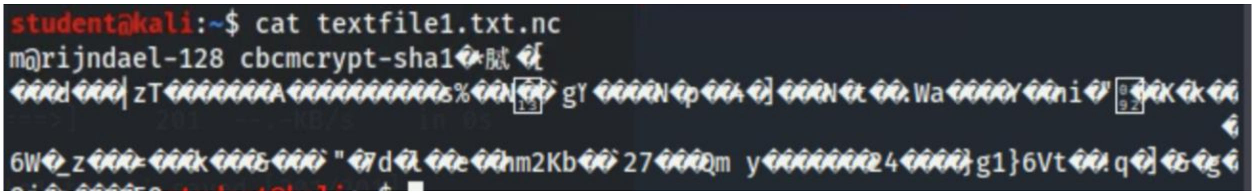
```
student@kali:~$ cat textfile1.txt
This is a sample textfile for encryption/decryption.
You can create text file locally on your Linux system using a text editor such a
s Gedit or Leafpad, depending on what is installed on your system.
```

Use **mcrypt** to encrypt your textfile. Mcrypt will ask for an encryption key – you can simply type a passphrase at the command line (you will use the same passphrase to decrypt the file so make sure to remember it). Be sure that you are in the same directory location as your text file and encrypt it as follows.

```
$ mcrypt textfile1.txt
```

If you list your directory you should see **textfile1.txt.nc** – the encrypted version of the file replaced the plaintext version. Use the **cat** command to view the file. It should be unintelligible.

Question 2: CUT AND PASTE THE CONTENTS OF THE FILE HERE: (.5 point)



```
student@kali:~$ cat textfile1.txt.nc
m@rijndael-128 cbcmcrypt-sha1��駄 ��
����d����|zT��������A�����������%���N���gY ����N�p���A�]����N�t��.Wa�����Y���ni�'���K�k��
                                                                                        �
6W�_z���=���k����5���"�7d�l��e��hm2Kb��27���Qm y�������24����}g1}6Vt��.q�]�5�g�
```

You could now send this file to someone else and as long as they have the passphrase, they can decrypt and read it. Now you can safely delete textfile1.txt (as long as you remember your passphrase so you can decrypt textfile1.txt.nc)!

```
$ rm textfile1.txt
```

Use **mcrypt** with the **–d** switch to decrypt your file. Be sure to use the same passphrase as in step 3, above.

```
$ mcrypt -d textfile1.txt.nc
```

Your unencrypted file should be restored to **`textfile1.txt`** (use **`cat`** to be sure).

Task 2: Asymmetric Encryption using Gnu Privacy Guard (gpg)

Asymmetric encryption using Gnu Privacy Guard (gpg), an open-source implementation of Pretty-Good Privacy (pgp). Gpg is included in your Kali Linux VM so we don't need to install anything. Below we will take basic steps to create a public/private key pair, then encrypt a file using our own public key and decrypt it using our own private key. There are lots more features and options, however. Review the man page for the gpg utility for more details.

First we have to create an encryption key

```
$ gpg --gen-key
```

You should be prompted for:

- Your name
- Your email address (and remember what you entered!).

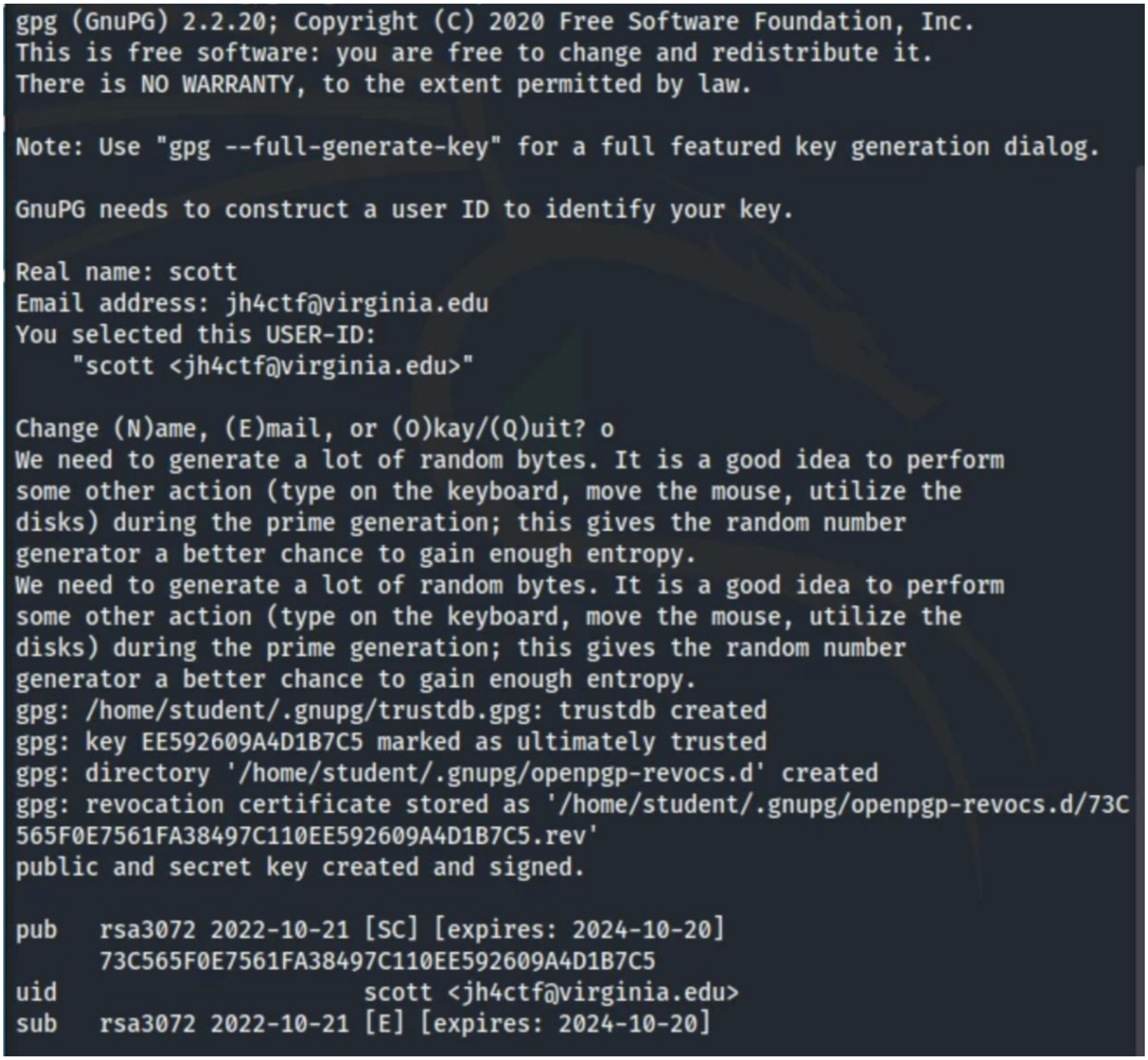
If everything looks ok you can select **O** for Okay when prompted.

You will next be prompted for a password to protect the key. Remember this password!

Now you must generate entropy by using the keyboard, moving the mouse, etc. until sufficient entropy is available to create your key. This entropy is needed in the generation of random numbers as part of the key creation process. This can take several minutes in a virtual machine.

Once complete, you should get output listing a public key fingerprint and some other data.

Question 3: CUT AND PASTE THE OUTPUT HERE: (.5 point)



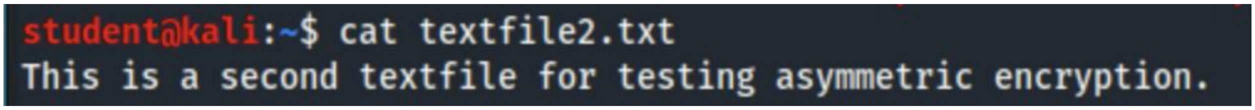
```
gpg (GnuPG) 2.2.20; Copyright (C) 2020 Free Software Foundation, Inc.
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.

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: scott
Email address: jh4ctf@virginia.edu
You selected this USER-ID:
    "scott <jh4ctf@virginia.edu>"

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: /home/student/.gnupg/trustdb.gpg: trustdb created
gpg: key EE592609A4D1B7C5 marked as ultimately trusted
gpg: directory '/home/student/.gnupg/openpgp-revocs.d' created
gpg: revocation certificate stored as '/home/student/.gnupg/openpgp-revocs.d/73C
565F0E7561FA38497C110EE592609A4D1B7C5.rev'
public and secret key created and signed.

pub   rsa3072 2022-10-21 [SC] [expires: 2024-10-20]
      73C565F0E7561FA38497C110EE592609A4D1B7C5
uid                      scott <jh4ctf@virginia.edu>
sub   rsa3072 2022-10-21 [E] [expires: 2024-10-20]
```

Download (or create) a second textfile.

```
$ wget artifacts.virginiacyberrange.net/gencyber/textfile2.txt
```

Use **cat** to examine the file.

Question 4: CUT AND PASTE THE CONTENTS OF THE FILE HERE: (.5 point)



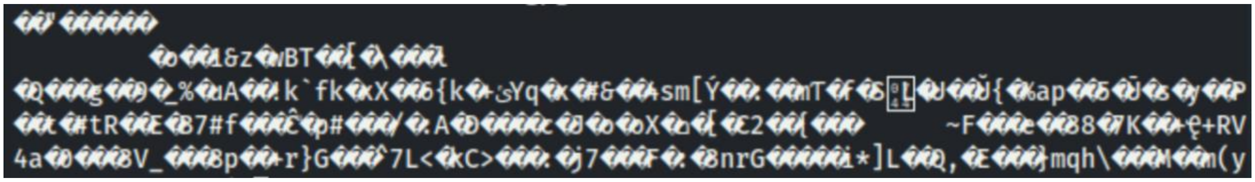
```
student@kali:~$ cat textfile2.txt
This is a second textfile for testing asymmetric encryption.
```

Now we'll encrypt the file using our public key.

```
$ gpg -e -r your-email-address textfile2.txt
```

A new file will be added called textfile2.txt.gpg. Use **cat** to examine the file. It should be unreadable.

Question 5: CUT AND PASTE THE CONTENTS OF THE FILE HERE: (.5 point)



```
��' ������
         �o���&z�wBT���[�\���
�Q����g���9�_%�uA���k`fk�xX���6{k�+ڏYq�x�#&���sm[Ý��.���mT�f�б�L��U���{�%ap���5��Ũ�y��P
���t�#tR���E�B7#f����Ĉp#���/�.A�D�����c�B�o�oX�a�[�£2���{���      ~F����e���8�7K���ȩ+RV
4a�0����8V_���8p���r}G���^7L<�kC>���.�j7���F�.�8nrG�����i*]L���,�E����mqh\����M���m(y
```

Next, delete the old file and use gpg to decrypt the file using your private key.

```
$ rm textfile2.txt
$ gpg -d textfile2.txt.gpg
```

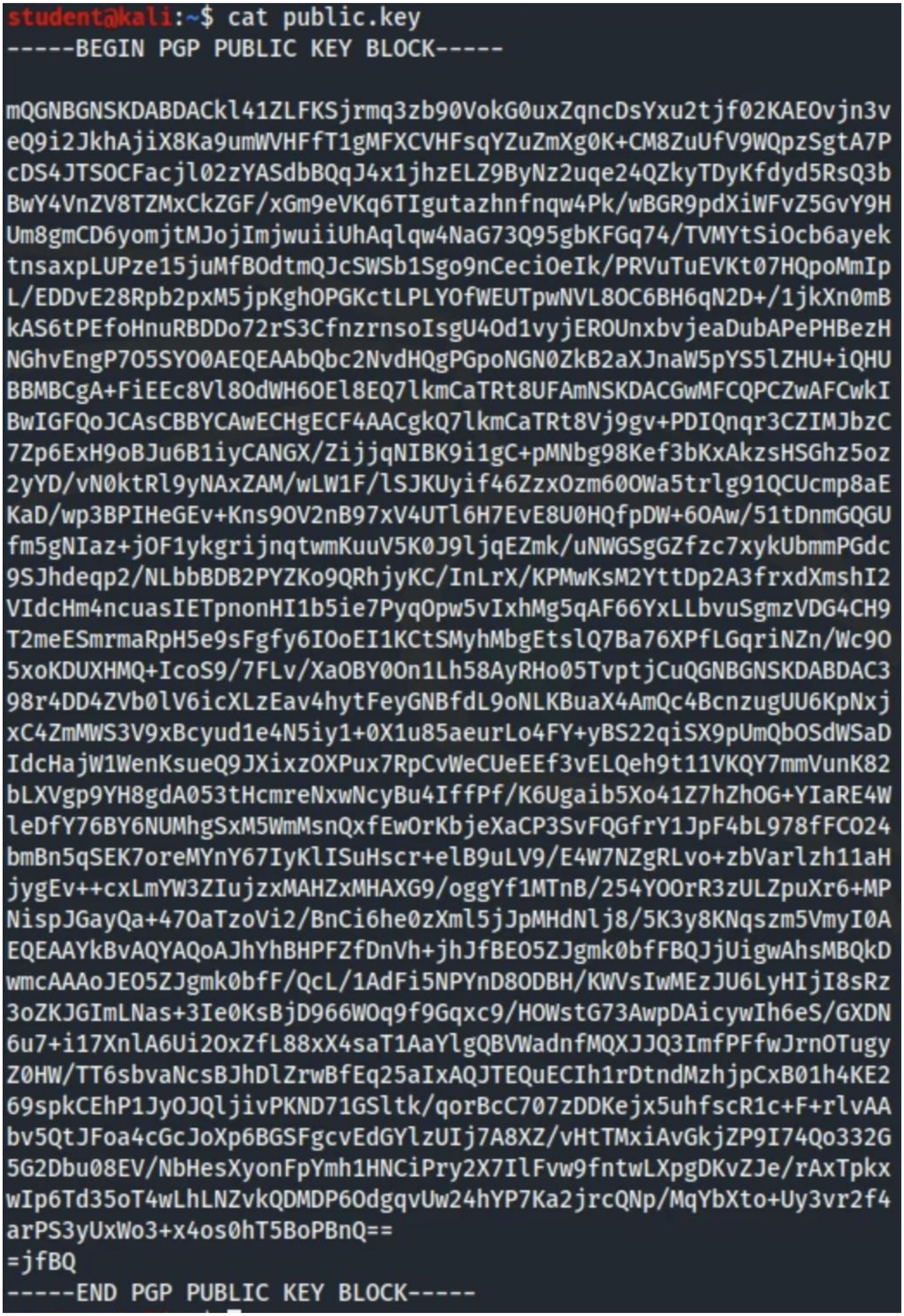
Enter the password that you created back in step 1. Your unencrypted file should be displayed!

Now that you know that your key works for encryption and decryption, you can share your public key with others so that they can encrypt files to be decrypted with your private key. Use the following syntax to export your key to a text file.

```
$ gpg --export -a your-email-address > public.key
```

Examine the key using **cat**. The **-a** flag has the key encoded in ascii (text). Some people append a text version of their public key to their email signatures, making is easy for others to use to encrypt files and send to them.

Question 6: CUT AND PASTE THE CONTENTS OF THE FILE HERE: (.5 point)



```
student@kali:~$ cat public.key
-----BEGIN PGP PUBLIC KEY BLOCK-----

mQGNBGNSKDABDACkl41ZLFKSjrmq3zb90VokG0uxZqncDsYxu2tjf02KAEOvjn3v
eQ9i2JkhAjiX8Ka9umWVHFfT1gMFXCVHFsqYZuZmXg0K+CM8ZuUfV9WQpzSgtA7P
cDS4JTSOCFacjl02zYASdbBQqJ4x1jhzELZ9ByNz2uqe24QZkyTDyKfdyd5RsQ3b
BwY4VnZV8TZMxCkZGF/xGm9eVKq6TIgutazhnfnqw4Pk/wBGR9pdXiWFvZ5GvY9H
Um8gmCD6yomjtMJojImjwuiiUhAqlqw4NaG73Q95gbKFGq74/TVMYtSiOcb6ayek
tnsaxpLUPze15juMfBOdtmQJcSWSb1Sgo9nCeciOeIk/PRVuTuEVKt07HQpoMmIp
L/EDDvE28Rpb2pxM5jpKghOPGKctLPLYOfWEUTpwNVL8OC6BH6qN2D+/1jkXn0mB
kAS6tPEfoHnuRBDDo72rS3CfnzrnsoIsgU4Od1vyjEROUnxbvjeaDubAPePHBezH
NGhvEngP7O5SYO0AEQEAAbQbc2NvdHQgPGpoNGN0ZkB2aXJnaW5pYS5lZHU+iQHU
BBMBCgA+FiEEc8Vl8OdWH6OEl8EQ7lkmCaTRt8UFAmNSKDACGwMFCQPCZwAFCwkI
BwIGFQoJCAsCBBYCAwECHgECF4AACgkQ7lkmCaTRt8Vj9gv+PDIQnqr3CZIMJbzC
7Zp6ExH9oBJu6B1iyCANGX/ZijjqNIBK9i1gC+pMNbg98Kef3bKxAkzsHSGhz5oz
2yYD/vN0ktRl9yNAxZAM/wLW1F/lSJKUyif46ZzxOzm60OWa5trlg91QCUcmp8aE
KaD/wp3BPIHeGEv+Kns9OV2nB97xV4UTl6H7EvE8U0HQfpDW+6OAw/51tDnmGQGU
fm5gNIaz+jOF1ykgrijnqtwmKuuV5K0J9ljqEZmk/uNWGSgGZfzc7xykUbmmPGdc
9SJhdeqp2/NLbbBDB2PYZKo9QRhjyKC/InLrX/KPMwKsM2YttDp2A3frxdXmshI2
VIdcHm4ncuasIETpnonHI1b5ie7PyqOpw5vIxhMg5qAF66YxLLbvuSgmzVDG4CH9
T2meESmrmaRpH5e9sFgfy6IOoEI1KCtSMyhMbgEtslQ7Ba76XPfLGqriNZn/Wc9O
5xoKDUXHMQ+IcoS9/7FLv/XaOBY0On1Lh58AyRHo05TvptjCuQGNBGNSKDABDAC3
98r4DD4ZVb0lV6icXLzEav4hytFeyGNBfdL9oNLKBuaX4AmQc4BcnzugUU6KpNxj
xC4ZmMWS3V9xBcyud1e4N5iy1+0X1u85aeurLo4FY+yBS22qiSX9pUmQbOSdWSaD
IdcHajW1WenKsueQ9JXixzOXPux7RpCvWeCUeEEf3vELQeh9t11VKQY7mmVunK82
bLXVgp9YH8gdA053tHcmreNxwNcyBu4IffPf/K6Ugaib5Xo41Z7hZhOG+YIaRE4W
leDfY76BY6NUMhgSxM5WmMsnQxfEwOrKbjeXaCP3SvFQGfrY1JpF4bL978fFCO24
bmBn5qSEK7oreMYnY67IyKlISuHscr+elB9uLV9/E4W7NZgRLvo+zbVarlzh11aH
jygEv++cxLmYW3ZIujzxMAHZxMHAXG9/oggYf1MTnB/254YOOrR3zULZpuXr6+MP
NispJGayQa+47OaTzoVi2/BnCi6he0zXml5jJpMHdNlj8/5K3y8KNqszm5VmyI0A
EQEAAYkBvAQYAQoAJhYhBHPFZfDnVh+jhJfBEO5ZJgmk0bfFBQJjUigwAhsMBQkD
wmcAAAoJEO5ZJgmk0bfF/QcL/1AdFi5NPYnD8ODBH/KWVsIwMEzJU6LyHIjI8sRz
3oZKJGImLNas+3Ie0KsBjD966WOq9f9Gqxc9/HOWstG73AwpDAicywIh6eS/GXDN
6u7+i17XnlA6Ui2OxZfL88xX4saT1AaYlgQBVWadnfMQXJJQ3ImfPFfwJrnOTugy
Z0HW/TT6sbvaNcsBJhDlZrwBfEq25aIxAQJTEQuECIh1rDtndMzhjpCxB01h4KE2
69spkCEhP1JyOJQljivPKND71GSltk/qorBcC707zDDKejx5uhfscR1c+F+rlvAA
bv5QtJFoa4cGcJoXp6BGSFgcvEdGYlzUIj7A8XZ/vHtTMxiAvGkjZP9I74Qo332G
5G2Dbu08EV/NbHesXyonFpYmh1HNCiPry2X7IlFvw9fntwLXpgDKvZJe/rAxTpkx
wIp6Td35oT4wLhLNZvkQDMDP6OdgqvUw24hYP7Ka2jrcQNp/MqYbXto+Uy3vr2f4
arPS3yUxWo3+x4os0hT5BoPBnQ==
=jfBQ
-----END PGP PUBLIC KEY BLOCK-----
```

From here, you could share your public key with others at a key-signing party, upload it to a key server, or otherwise make it available for others to use to encrypt documents that only you can decrypt.

Task 3: RSA Encryption/Decryption*

Let’s take another look at asymmetric encryption to perform RSA and generate keys to encrypt and decrypt a message. Pay particular attention to the output of the variables of the algorithm because you will be implementing them in your next programming assignment!

Let’s use the **openssl** library to generate a public/private key pair and encrypt a file. To begin, generate a pair of public and private keys by running the following command:

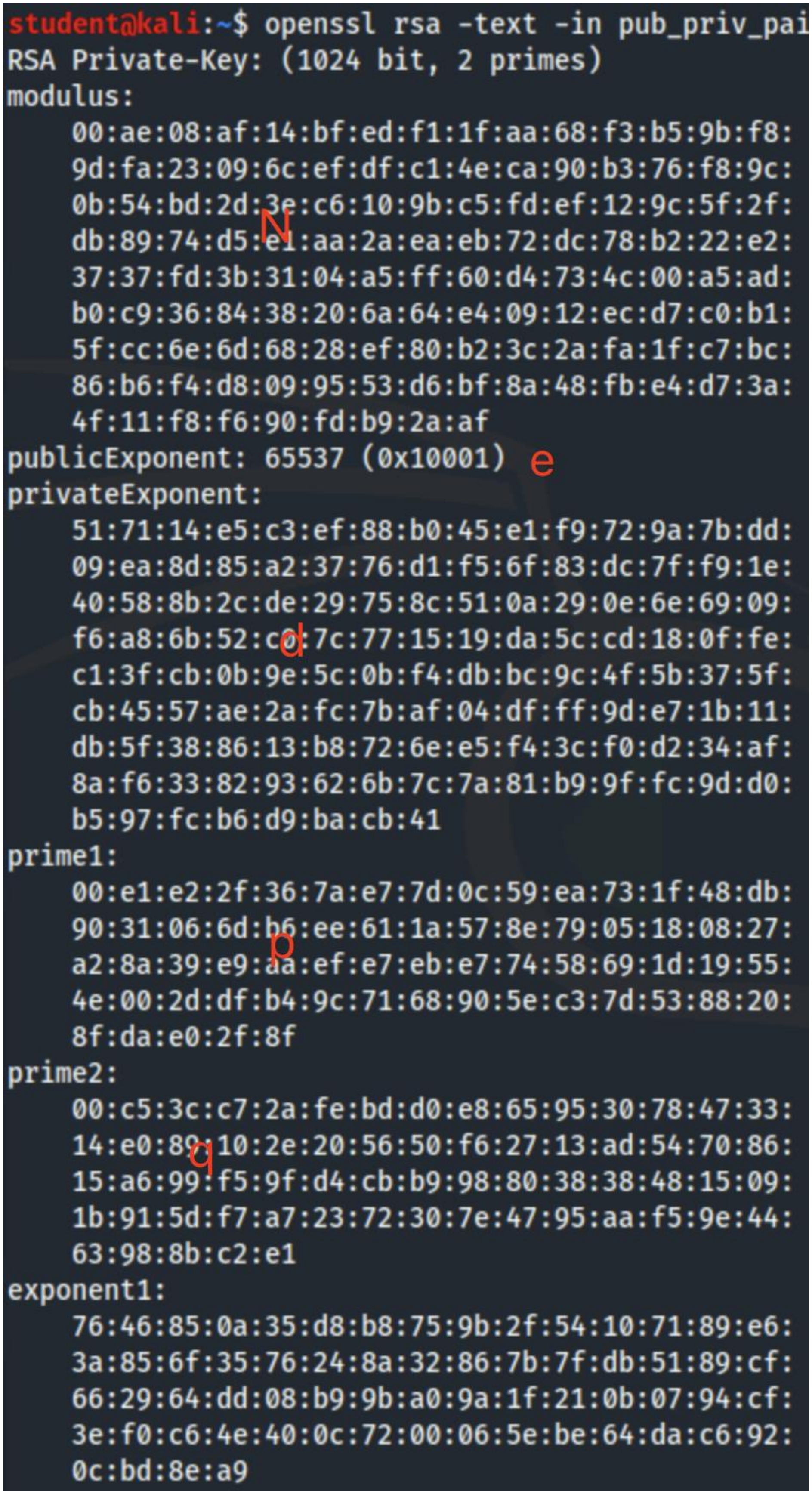
```
openssl genrsa -out pub_priv_pair.key 1024
```

The **genrsa** flag lets **openssl** know that you want to generate an RSA key, the **-out** flag specifies the name of the output file, and the value **1024** represents the length of the key. Longer keys are more secure. Remember: don’t share your private key with anyone. You can view the key pair you generated by running the following command:

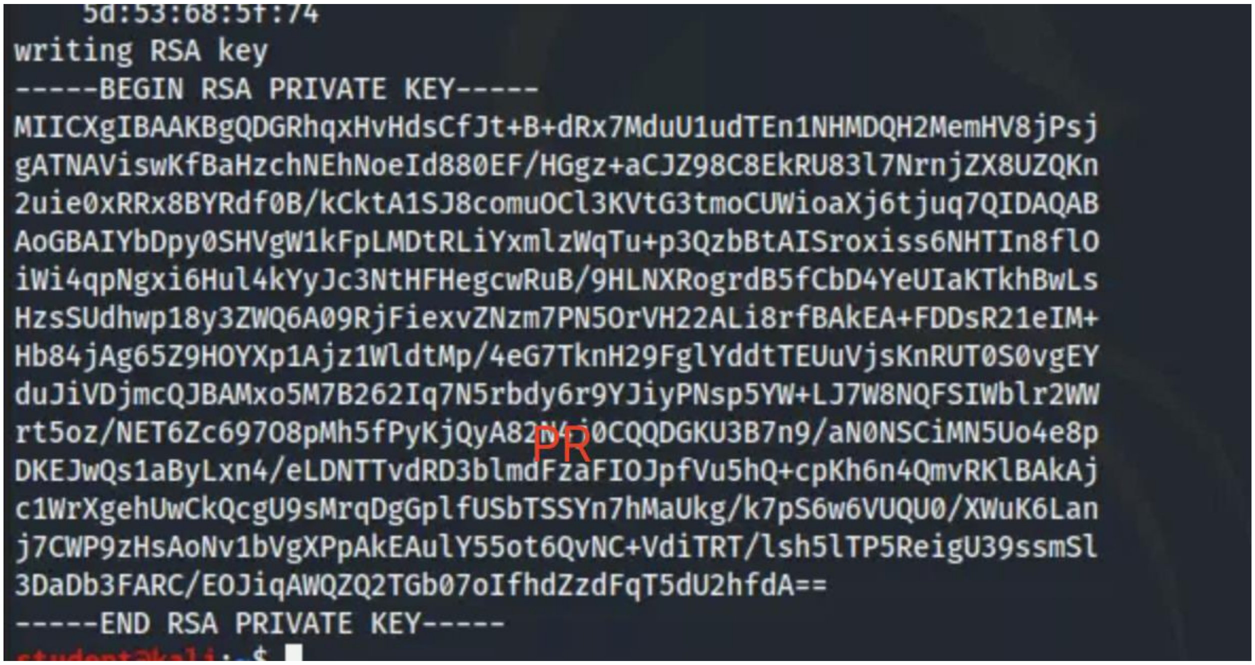
```
openssl rsa -text -in pub_priv_pair.key
```

The **rsa** flag tells **openssl** to interpret the key as an RSA key and the **-text** flag displays the key in humanreadable format.

Question 7: CUT AND PASTE YOUR OUTPUT HERE. Label the areas of the output that correspond to the RSA algorithm components (p, q, n, integer e, d, PR) (1 point) **Note: if you plan to use your public/private key pair in real life, please obfuscate your private key in the cut and paste.



```
student@kali:~$ openssl rsa -text -in pub_priv_pai
RSA Private-Key: (1024 bit, 2 primes)
modulus:
    00:ae:08:af:14:bf:ed:f1:1f:aa:68:f3:b5:9b:f8:
    9d:fa:23:09:6c:ef:df:c1:4e:ca:90:b3:76:f8:9c:
    0b:54:bd:2d:3e:c6:10:9b:c5:fd:ef:12:9c:5f:2f:
    db:89:74:d5:e1:aa:2a:ea:eb:72:dc:78:b2:22:e2:
    37:37:fd:3b:31:04:a5:ff:60:d4:73:4c:00:a5:ad:
    b0:c9:36:84:38:20:6a:64:e4:09:12:ec:d7:c0:b1:
    5f:cc:6e:6d:68:28:ef:80:b2:3c:2a:fa:1f:c7:bc:
    86:b6:f4:d8:09:95:53:d6:bf:8a:48:fb:e4:d7:3a:
    4f:11:f8:f6:90:fd:b9:2a:af
publicExponent: 65537 (0x10001)
privateExponent:
    51:71:14:e5:c3:ef:88:b0:45:e1:f9:72:9a:7b:dd:
    09:ea:8d:85:a2:37:76:d1:f5:6f:83:dc:7f:f9:1e:
    40:58:8b:2c:de:29:75:8c:51:0a:29:0e:6e:69:09:
    f6:a8:6b:52:c0:7c:77:15:19:da:5c:cd:18:0f:fe:
    c1:3f:cb:0b:9e:5c:0b:f4:db:bc:9c:4f:5b:37:5f:
    cb:45:57:ae:2a:fc:7b:af:04:df:ff:9d:e7:1b:11:
    db:5f:38:86:13:b8:72:6e:e5:f4:3c:f0:d2:34:af:
    8a:f6:33:82:93:62:6b:7c:7a:81:b9:9f:fc:9d:d0:
    b5:97:fc:b6:d9:ba:cb:41
prime1:
    00:e1:e2:2f:36:7a:e7:7d:0c:59:ea:73:1f:48:db:
    90:31:06:6d:b6:ee:61:1a:57:8e:79:05:18:08:27:
    a2:8a:39:e9:aa:ef:e7:eb:e7:74:58:69:1d:19:55:
    4e:00:2d:df:b4:9c:71:68:90:5e:c3:7d:53:88:20:
    8f:da:e0:2f:8f
prime2:
    00:c5:3c:c7:2a:fe:bd:d0:e8:65:95:30:78:47:33:
    14:e0:89:10:2e:20:56:50:f6:27:13:ad:54:70:86:
    15:a6:99:f5:9f:d4:cb:b9:98:80:38:38:48:15:09:
    1b:91:5d:f7:a7:23:72:30:7e:47:95:aa:f5:9e:44:
    63:98:8b:c2:e1
exponent1:
    76:46:85:0a:35:d8:b8:75:9b:2f:54:10:71:89:e6:
    3a:85:6f:35:76:24:8a:32:86:7b:7f:db:51:89:cf:
    66:29:64:dd:08:b9:9b:a0:9a:1f:21:0b:07:94:cf:
    3e:f0:c6:4e:40:0c:72:00:06:5e:be:64:da:c6:92:
    0c:bd:8e:a9
```



```
    5d:53:68:5f:74
writing RSA key
-----BEGIN RSA PRIVATE KEY-----
MIICXgIBAAKBgQDGRhqxHvHdsCfJt+B+dRx7MduU1udTEn1NHMDQH2MemHV8jPsj
gATNAViswKfBaHzchNEhNoeId880EF/HGgz+aCJZ98C8EkRU83l7NrnjZX8UZQKn
2uie0xRRx8BYRdf0B/kCktA1SJ8comuOCl3KVtG3tmoCUWioaXj6tjuq7QIDAQAB
AoGBAIYbDpy0SHVgW1kFpLMDtRLiYxmlzWqTu+p3QzbBtAISroxiss6NHTIn8flO
iWi4qpNgxi6Hul4kYyJc3NtHFHegcwRuB/9HLNXRogrdB5fCbD4YeUIaKTkhBwLs
HzsSUdhwp18y3ZWQ6A09RjFiexvZNzm7PN5OrVH22ALi8rfBAkEA+FDDsR21eIM+
Hb84jAg65Z9HOYXp1Ajz1WldtMp/4eG7TknH29FglYddtTEUuVjsKnRUT0S0vgEY
duJiVDjmcQJBAMxo5M7B262Iq7N5rbdy6r9YJiyPNsp5YW+LJ7W8NQFSIWblr2WW
rt5oz/NET6Zc69708pMh5fPyKjQyA82W4j0CQQDGKU3B7n9/aN0NSCiMN5Uo4e8p
DKEJwQs1aByLxn4/eLDNTTvdRD3blmdFzaFIOJpfVu5hQ+cpKh6n4QmvRKlBAkAj
c1WrXgehUwCkQcgU9sMrqDgGplfUSbTSSYn7hMaUkg/k7pS6w6VUQU0/XWuK6Lan
j7CWP9zHsAoNv1bVgXPpAkEAulY55ot6QvNC+VdiTRT/lsh5lTP5ReigU39ssmSl
3DaDb3FARC/EOJiqAWQZQ2TGb07oIfhdZzdFqT5dU2hfdA==
-----END RSA PRIVATE KEY-----
student@kali:~$ 
```

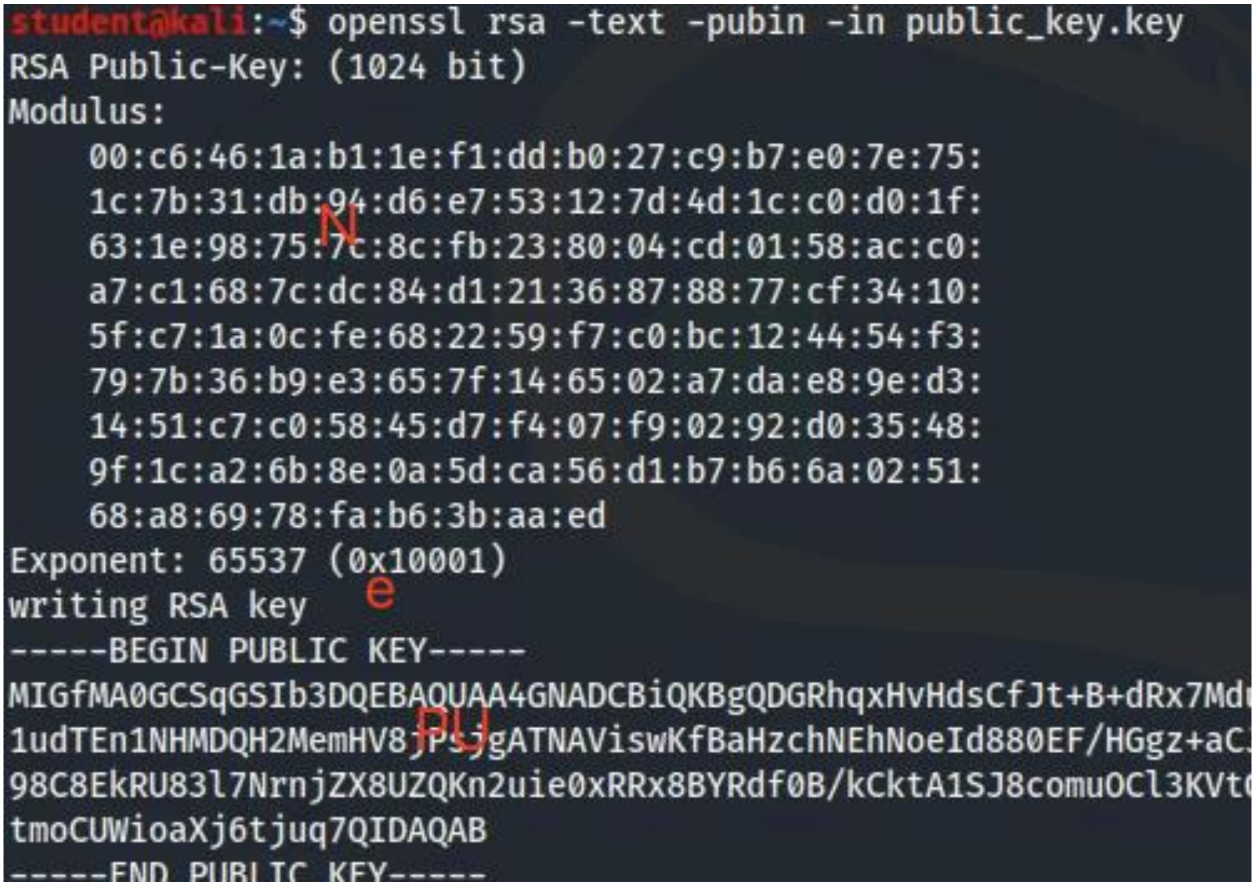
You can extract the public key from this file by running the following command:

`openssl rsa -in pub_priv_pair.key -pubout -out public_key.key`

The **`-pubout`** flag tells **`openssl`** to extract the public key from the file. You can view the public key by running the following command, in which the **`-pubin`** flag instructs **`openssl`** to treat the input as a public key:

`openssl rsa -text -pubin -in public_key.key`

Question 8: CUT AND PASTE YOUR OUTPUT HERE. Label the areas of the output that correspond to the RSA algorithm components (n, integer e, PU) (1 point)



```
student@kali:~$ openssl rsa -text -pubin -in public_key.key
RSA Public-Key: (1024 bit)
Modulus:
    00:c6:46:1a:b1:1e:f1:dd:b0:27:c9:b7:e0:7e:75:
    1c:7b:31:db:94:d6:e7:53:12:7d:4d:1c:c0:d0:1f:
    63:1e:98:75:7c:8c:fb:23:80:04:cd:01:58:ac:c0:
    a7:c1:68:7c:dc:84:d1:21:36:87:88:77:cf:34:10:
    5f:c7:1a:0c:fe:68:22:59:f7:c0:bc:12:44:54:f3:
    79:7b:36:b9:e3:65:7f:14:65:02:a7:da:e8:9e:d3:
    14:51:c7:c0:58:45:d7:f4:07:f9:02:92:d0:35:48:
    9f:1c:a2:6b:8e:0a:5d:ca:56:d1:b7:b6:6a:02:51:
    68:a8:69:78:fa:b6:3b:aa:ed
Exponent: 65537 (0x10001)
writing RSA key
-----BEGIN PUBLIC KEY-----
MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQDGRhqxHvHdsCfJt+B+dRx7Md
1udTEn1NHMDQH2MemHV8jPSjgATNAViswKfBaHzchNEhNoeId880EF/HGgz+aC
98C8EkRU83l7NrnjZX8UZQKn2uie0xRRx8BYRdf0B/kCktA1SJ8comuOCl3KVt
tmoCUWioaXj6tjuq7QIDAQAB
-----END PUBLIC KEY-----
```

Next, let's create a text file to encrypt:

echo "Cryptography is fun!" > plain.txt

Next, use the RSA utility **rsautl** to create an encrypt plain.txt to and encrypted binary file **cipher.bin** using your public key:

```
openssl rsautl -encrypt -pubin -inkey public_key.key -in plain.txt -
out cipher.bin -oaep
```

Notice that we included the **-oaep** flag. Secure implementations of RSA must also use the OAEP algorithm. Whenever you're encrypting and decrypting files using **openssl**, be sure to apply this flag to make the operations secure.

Next, decrypt the binary using the following command:

```
openssl rsautl -decrypt -inkey pub_priv_pair.key -in cipher.bin -out
plainD.txt -oaep
```

Lastly, you can view the decrypted message plainD.txt using the **cat** command and you should see your original message.

Task 4: Other Encryption/Decryption

Question 9: Decrypt the following Caesar Cipher: `psvclaolcpynpuphjfilyyhunl` (1 point)

`ilovethevirginiacyberrange`

Question 10: Generate the MD5 hash of the following sentence: I love hash browns for breakfast. (Do not include the period when generating the MD5). (1 point)

53ca9be5f40f02cab06b4541b0d9c8ea

By submitting this assignment you are digitally signing the honor code, "I pledge that I have neither given nor received help on this assignment".

END OF EXERCISE

References

Mcrypt: http://mcrypt.sourceforge.net/
Gpg: https://gnupg.org/
Openssl: https://www.openssl.org/

*Openssl task credit to *Ethical Hacking* by Daniel Graham