CS 3710 Introduction to Cybersecurity
Term: Spring 2023

Lab Exercise 6 – Cryptography

Due Date: March 24, 2023 11:59pm
Points Possible: 7

Name: Carlos Revilla

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 thatsg 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:

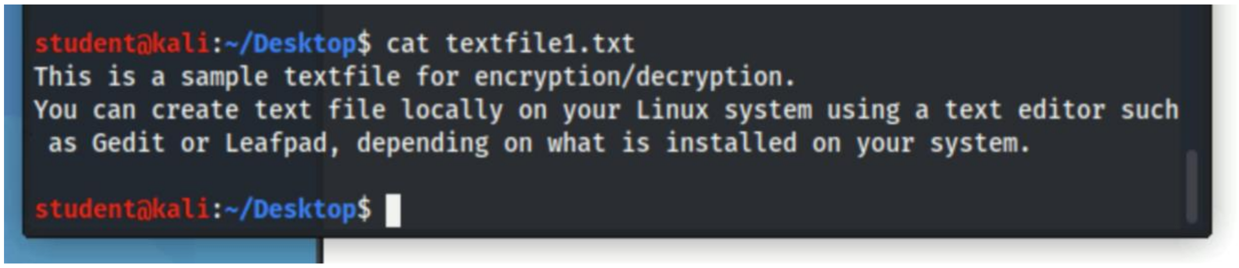
```
$ mcrypt --list
```

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)



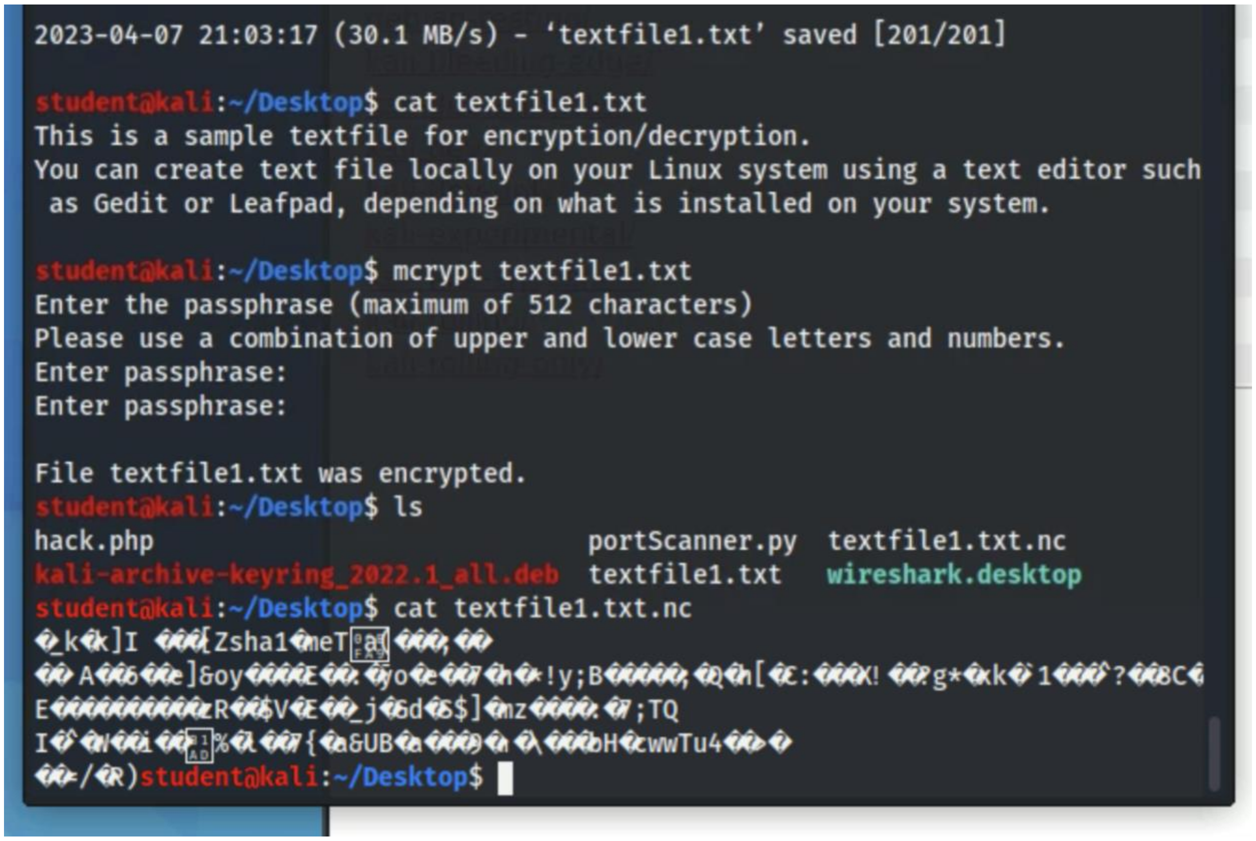
```
student@kali:~/Desktop$ cat textfile1.txt
This is a sample textfile for encryption/decryption.
You can create text file locally on your Linux system using a text editor such
 as Gedit or Leafpad, depending on what is installed on your system.

student@kali:~/Desktop$ 
```

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)



```
2023-04-07 21:03:17 (30.1 MB/s) - 'textfile1.txt' saved [201/201]

student@kali:~/Desktop$ 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
 as Gedit or Leafpad, depending on what is installed on your system.

student@kali:~/Desktop$ mcrypt textfile1.txt
Enter the passphrase (maximum of 512 characters)
Please use a combination of upper and lower case letters and numbers.
Enter passphrase:
Enter passphrase:

File textfile1.txt was encrypted.
student@kali:~/Desktop$ ls
hack.php                              portScanner.py  textfile1.txt.nc
kali-archive-keyring_2022.1_all.deb  textfile1.txt   wireshark.desktop
student@kali:~/Desktop$ cat textfile1.txt.nc
�_k�k]I ���Zsha1�meT��� ��
�� A��6��e]&oy����E�� �yo�e��7�h�!y;B�����,�Q�h[�E:���X! ��g*�k�1���?��8C�
E����������zR��$V�E��_j�6d�$]�mz����.�7;TQ
I� �N���i ��%�A��7{�a&UB�a���9� �\���bH�zwwTu4���
��=/�R)student@kali:~/Desktop$
```

Paraphrase:
carlosiscool

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!

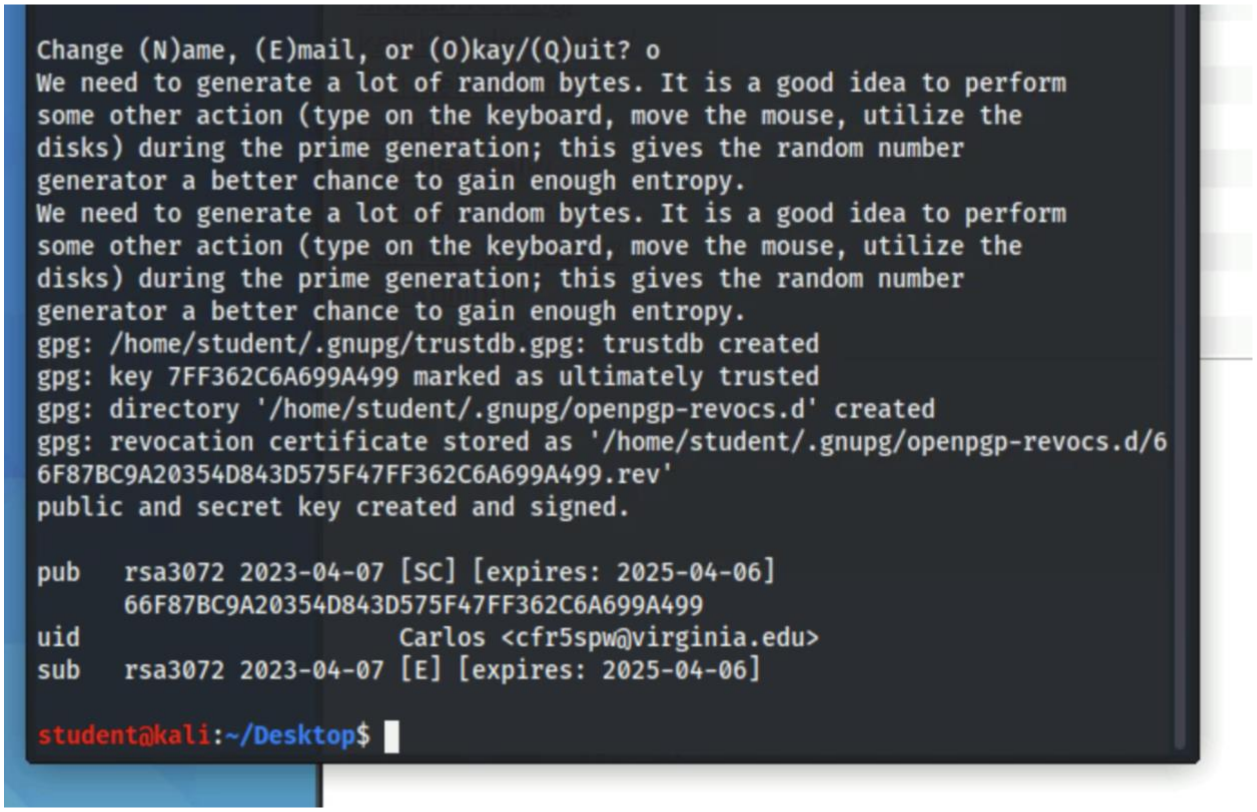
Pswd: carlosiscool

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)

```
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 7FF362C6A699A499 marked as ultimately trusted
gpg: directory '/home/student/.gnupg/openpgp-revocs.d' created
gpg: revocation certificate stored as '/home/student/.gnupg/openpgp-
revocs.d/66F87BC9A20354D843D575F47FF362C6A699A499.rev'
public and secret key created and signed.

pub   rsa3072 2023-04-07 [SC] [expires: 2025-04-06]
      66F87BC9A20354D843D575F47FF362C6A699A499
uid                 Carlos <cfr5spw@virginia.edu>
sub   rsa3072 2023-04-07 [E] [expires: 2025-04-06]
```



```
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 7FF362C6A699A499 marked as ultimately trusted
gpg: directory '/home/student/.gnupg/openpgp-revocs.d' created
gpg: revocation certificate stored as '/home/student/.gnupg/openpgp-revocs.d/6
6F87BC9A20354D843D575F47FF362C6A699A499.rev'
public and secret key created and signed.

pub   rsa3072 2023-04-07 [SC] [expires: 2025-04-06]
      66F87BC9A20354D843D575F47FF362C6A699A499
uid                      Carlos <cfr5spw@virginia.edu>
sub   rsa3072 2023-04-07 [E] [expires: 2025-04-06]

student@kali:~/Desktop$
```

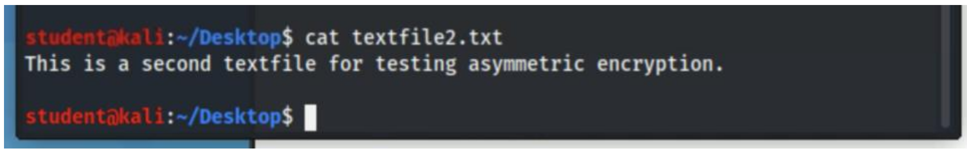
Download (or create) a second textfile.

$ 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)

This is a second textfile for testing asymmetric encryption.



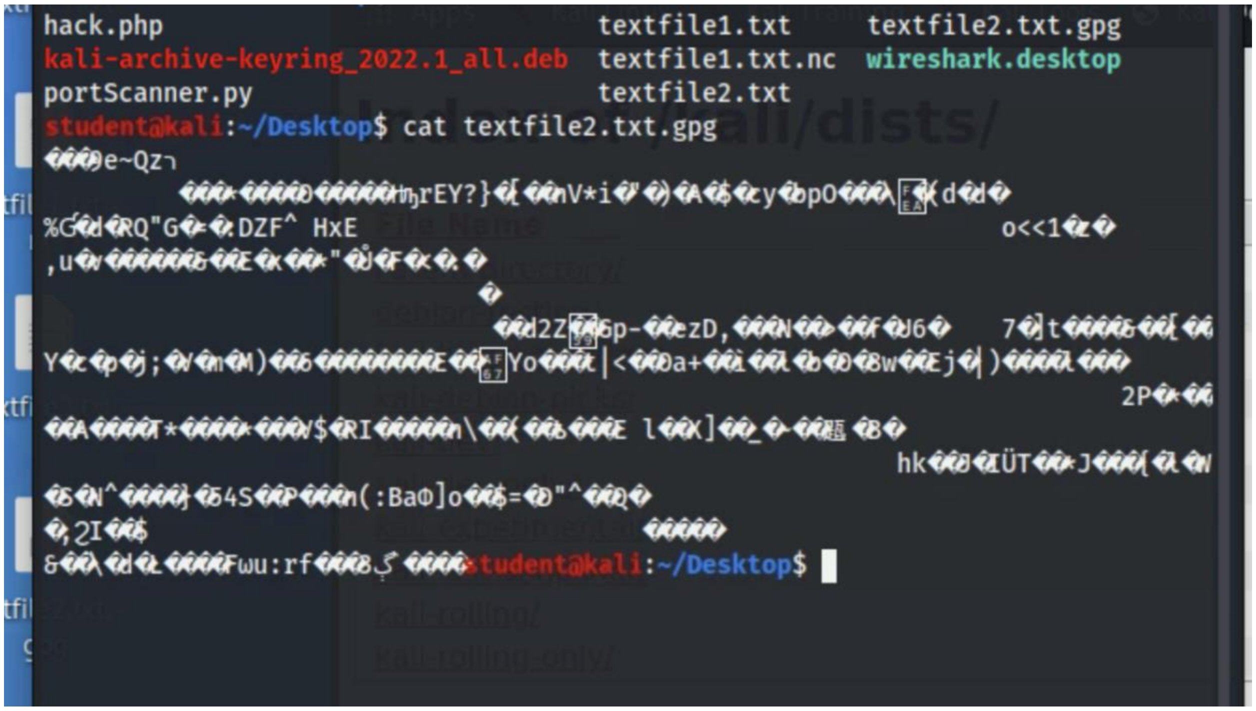
```
student@kali:~/Desktop$ cat textfile2.txt
This is a second textfile for testing asymmetric encryption.

student@kali:~/Desktop$
```

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)



```
hack.php                              textfile1.txt      textfile2.txt.gpg
kali-archive-keyring_2022.1_all.deb   textfile1.txt.nc   wireshark.desktop
portScanner.py                        textfile2.txt
student@kali:~/Desktop$ cat textfile2.txt.gpg
���e~Qz¬
      ���������������ɱrEY?}�[��nV*i�'�)�A�$�cy�bp0���\�(d�d�
%G�d�RQ"G�=�.DZF^ HxE                                    o<<1�z�
,u�v���������E�x��*"�ů�F�x�.�
                           �
                            ��d2Z��6p-��ezD,���N��>��f�U6�    7�]t����6��[��
Y�c�p�j;�V�m�M)��6���������E��Yo���t|<���a+��i��l�b�D�8w��Ej�|)����l���
                                                                   2P�*��
��A����T*�����*����V$�RI������n\��(��b���E l��X]��_�-��� �B�
                                                   hk���8�IÜT��*J���{�l�W
�5�N^����}�54S��P����n(:BaΦ]o��$=�D"^��Q�
�,2I��$                                        �����
&���\�d�L����Fwu:rf���B ���student@kali:~/Desktop$ █
```

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)

cat public.key
-----BEGIN PGP PUBLIC KEY BLOCK-----

mQGNBGQwh84BDADZ6lI4iiWt3DlC2O0MuCWouJwtVSo0LtpTv+RasQ9KHGFReI8f
rJBHyyuHz2lFUYkJK0hXAFI7VaCPwUP8y6QnBt5Xsfobfzs6lIJXH1p6WOBquxQz

```
3N0Eal1s00STnfwI5JnLzuK4EJQMTFAW9YbrsGz0Xz1YYbUvox9lyN6ayr/BNQxX
p5nDdf3Kh6ccfT+i0ekQGZK6GAv0mknHy+ksY+1aDSXgbBXcE8Bq4AAI+aGEuRoz
IOApSIvO8E2e1HxuP840iK/mIhA1Gi8XRQZPpSLE+j5K+jZDJvTUkQVGSogp0Z2e
Yg8C3QsKRTiN+Pw7AtmOIxmLu7mYePubuIn4piKdFE7XDO3PqzpQd/CzEY+hGHrV
Cf+2TCqaS+ZYqRHnirTrGa1yXhXfZ0WvXrqi7etEuze/dmi8381hEtJuB3zJhFPv
LNjjhA7yC3CmJv9Dr4PmHLKj4eo/v6Ck0IrwO6lYSXtq8JRHul3l4DOlgDyRNyxG
NM60W01lEcSyr48AEQEAAbQdQ2FybG9zIDxjZnI1c3B3QHZpcmdpbmlhLmVkdT6J
AdQEEwEKAD4WIQRm+HvJogNU2EPVdfR/82LGppmkmQUCZDCHzgIbAwUJA8JnAAUL
CQgHAgYVCgkICwIEFgIDAQIeAQIXgAAKCRB/82LGppmkmU8TDACV6uNPunSZcdOf
WAGF3OoycEfK+AwdcYQjHCmbUtKgvl96/hipfXX6mzGGruil5w5BDwNde9tdmNxv
fHx7OuPuixi/87mK5jO3mUtdE7DbWlC/ngVwkDv8yeHm5SpO94q7pydIDf4i3u+Y
zEmRq50kHtPJlNvBoERjoMfgdh8AVmdNLn6ILJrbJwwtuKBaXTtDblqG3A3Bn5N4
T+SadppQpNxEntyNqevf10GmLz95nIFiauAwgNg3IfMlFs5iIEmvSnqCFmCRfIWc
pMmTZqBLASkLGVWaF/kPp5QKxsmu19xNXKhmD9Wxyk4gWIIZE1s27Pjy9E62T4qp
ZAYrpZbmgSu+yu8lWLuw+deOxoJ+IcpFGe+AZ0o0kDhdt/iJNIJkyY35wAFbwjVS
xlMgFO4JMLVoDHsiT3qU5lpSRfhM/79sYYo/VsgWZ84o8ldt64fQNbxSnmNB8VYV
/Gps+C3prfnrYYlOiYUYSm0lWrNND5V8TvE0FJcM6nZ9caBiwvG5AY0EZDCHzgEM
ANa1AVgdobCxyGuFYyP6sf0rlDpQ0YlfnkMJxSOFUSpC/7FEbjvaVX4qP9IDVWCN
L9JjTZEprQNvwKlhVZdFHoFyQUpYmGTu6YZqZiKqidWAjx0XJ71eFz3pELSc4VCv
YLyycWu/u07mkjhEmfndwS1qjM9WuaisKsIzPTWr4ZtdsVqw1c02k/BrTXtw9+5E
gDK4bKOtR0sd3uaa5q7vChPxrhWhdX1I6nFuYXhsT9X2GYlN3lT9PC/iXR+EyxTe
oqGN5Rbb77ydSam3iIOYRpsv/MGJ0l15/yGx0+DhN8yqFh9mUQJn1YQA94eVtVNK
xbFMDBXoEgHwEjxgKRq8ZAZ06/jn6exlFhGPGz7FFAxdWKtBYYO+xRFIzVcqx621
Sx4oqizMzNLF0JMgT7A9WXWsv7if2Et3jR2Ay/obwJvjlXuAWUlfsWAnnHhgj9RR
YhnZAdm3OiWQYAStGzo9ALS4+GzOVzXeIl+r1PBnTGoKX5iQJ+PMn2Txs0IG2/RB
WQARAQABiQG8BBgBCgAmFiEEZvh7yaIDVNhD1XX0f/NixqaZpJkFAmQwh84CGwwF
CQPCZwAACgkQf/NixqaZpJlfcQwAoqLGjp6zSvC2UJHtqN31ZJo56+CbIOsNo9w+
qv/Gm/3GwhdDhbmMtukZju2YmDMA5u6vrPfunNjmUK6uBXkFnvvjxAS8JLBLFgaP
Ga+WVq1Qq3OENI+S8hkND3T/giEmdcqGOh8NtHARX4xw/yI0rsx9hKM6nShZOG8b
swy9DXpImB8I1LGu/S3WZ/Px/1o5e6ScYMThSTg75xlUtjBROSIoGQnlO5ZLL2QL
AmjjkfhJl2FReUfXvb76Ih5z2wvFN+3IVJVdH/XQnBHGF0xg/qmKubZ5NH5tRbvS
ewb6ufDIU7ZumWIYhQjxsKbc8nLuoV7LanY1vPgSvCweG8nXwhxtbEayiC2j7Qk0
UsUUnl6jjWq7cLPPvnZiak3opdPTlnXAj78BBGKJxrDnlNGnp8qyyFZ9TNOzG8HP
W7pIe+P994QN1xgZloNOqAFzzB4g6GknF7SpcuhvUOjhNn6N0ADIVu14jU6UIocm
IOyLWCNnTOLmGPUQg7d49Dt0Jbjy
=8sYF
-----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, 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.

```
RSA Private-Key: (1024 bit, 2 primes)
modulus: (n)
    00:c6:a7:16:71:79:e1:ac:4a:3c:36:67:26:f4:b5:
    03:32:d7:22:f2:43:5a:bd:ea:a4:88:1b:60:ac:ac:
    d2:e7:82:13:1f:29:53:2f:3c:84:06:3e:93:4c:91:
    b2:87:81:7b:2f:06:f4:3e:77:1f:17:6a:64:92:15:
    2d:10:f4:b6:bc:05:6a:3d:68:e3:52:05:9b:2f:ab:
    51:6b:55:96:1e:62:e4:3d:13:b4:95:1c:f8:83:5d:
    73:ad:9e:f2:d8:56:3c:ac:30:58:11:0f:e8:c9:09:
    00:2d:f3:f0:09:6d:e8:75:a7:ff:7d:26:5c:93:e0:
    94:68:ca:72:23:99:30:ab:d1
publicExponent: 65537 (0x10001) (e)
privateExponent: (d)
    00:9d:d6:94:d6:84:e8:f2:63:e9:83:b0:62:1b:7c:
    d7:95:c5:aa:56:a3:7d:ad:f8:89:d2:3b:2e:8d:04:
    a4:6e:9d:c3:63:b7:0a:09:36:24:10:72:17:c7:76:
    dc:4b:1e:6e:29:e7:74:99:4d:3c:be:f6:22:02:90:
    20:c6:a0:29:af:c3:f1:05:e5:07:a0:0d:7f:c5:1b:
    d0:4f:a6:fc:cd:c2:7a:03:5b:14:26:9f:06:2b:80:
    6e:4d:07:33:a5:75:7d:81:4a:24:14:97:32:94:e3:
    57:e8:90:ad:6f:54:68:3f:04:4c:87:b7:b2:3d:f1:
    38:93:fb:a7:55:ff:97:a5:81
prime1: (p)
    00:ef:ec:03:fe:42:49:69:2a:54:bc:b2:77:ca:1f:
    12:c7:f5:50:de:a8:89:96:c9:28:19:6f:5a:21:d7:
    30:e7:f6:be:b0:43:e0:90:8b:ce:9f:5a:f9:74:05:
    18:1e:98:a0:02:0f:b8:0b:78:0b:45:34:69:36:28:
    1f:5c:b6:a6:5d
prime2: (q)
```

```
        00:d3:f7:13:ab:4c:cd:83:12:e1:f4:60:33:e4:72:
        7f:c0:2e:18:fa:ae:fb:43:e3:de:98:91:60:66:b9:
        d1:27:8a:ff:5b:a1:fb:1c:41:59:2b:52:a1:c1:87:
        c0:7d:f5:a2:38:a8:f4:b6:8d:45:88:30:6c:30:37:
        85:f3:ce:5c:05
    exponent1:
        00:c1:68:32:63:e2:4d:c9:90:4d:54:fc:4d:a2:cb:
        5f:d9:7f:c1:9c:6c:a2:d5:c1:fd:28:5b:e2:7d:cb:
        3a:6a:94:37:6e:62:99:82:0c:a2:19:46:3e:37:af:
        ef:9c:a2:8e:c2:7a:a7:73:df:66:be:78:1c:a4:82:
        bd:9d:80:c4:25
    exponent2:
        00:8f:36:c9:5b:5c:31:1e:f6:8b:24:8a:3b:85:86:
        b7:3a:29:eb:46:b5:23:9f:e3:3e:6a:e7:0e:b3:59:
        a9:2f:86:82:b2:6e:e7:33:58:13:df:69:9d:51:1d:
        5f:b9:bb:55:37:e3:30:34:87:8f:0b:4d:6c:4b:c7:
        b8:01:da:d3:e1
    coefficient:
        15:ac:96:f1:e7:a9:0b:a0:7e:c8:d9:e2:0c:d7:f5:
        40:c9:9e:e5:2a:6f:4f:0f:4a:3c:75:c0:9c:93:e7:
        50:70:29:d8:0d:ed:98:20:e3:63:81:8a:bc:af:8e:
        25:6e:26:34:99:ce:ce:37:03:69:78:30:0f:c3:b8:
        c4:bd:e4:a3
    writing RSA key
    -----BEGIN RSA PRIVATE KEY----- (PR)
    MIICXgIBAAKBgQDGpxZxeeGsSjw2Zyb0tQMy1yLyQ1q96qSIG2CsrNLnghMfKVMv
    PIQGPpNMkbKHgXsvBvQ+dx8XamSSFS0Q9La8BWo9aONSBZsvq1FrVZYeYuQ9E7SV
    HPiDXXOtnvLYVjysMFgRD+jJCQAt8/AJbeh1p/99JlyT4JRoynIjmTCr0QIDAQAB
    AoGBAJ3WlNaE6PJj6YOwYht815XFqlajfa34idI7Lo0EpG6dw2O3Cgk2JBByF8d2
    3EsebinndJlNPL72IgKQIMagKa/D8QXlB6ANf8Ub0E+m/M3CegNbFCafBiuAbk0H
    M6V1fYFKJBSXMpTjV+iQrW9UaD8ETIe3sj3xOJP7p1X/l6WBAkEA7+wD/kJJaSpU
    vLJ3yh8Sx/VQ3qiJlskoGW9aIdcw5/a+sEPgkIvOn1r5dAUYHpigAg+4C3gLRTRp
    NigfXLamXQJBANP3E6tMzYMS4fRgM+Ryf8AuGPqu+0Pj3piRYGa50SeK/1uh+xxB
    WStSocGHwH31ojio9LaNRYgwbDA3hfPOXAUCQQDBaDJj4k3JkE1U/E2iy1/Zf8Gc
    bKLVwf0oW+J9yzpqlDduYpmCDKIZRj43r++coo7Ceqdz32a+eBykgr2dgMQlAkEA
    jzbJW1wxHvaLJIo7hYa3OinrRrUjn+M+aucOs1mpL4aCsm7nM1gT32mdUR1fubtV
    N+MwNIePC01sS8e4AdrT4QJAFayW8eepC6B+yNniDNf1QMme5SpvTw9KPHXAnJPn
    UHAp2A3tmCDjY4GKvK+OJW4mNJnOzjcDaXgwD8O4xL3kow==
    -----END RSA PRIVATE KEY-----
```

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, e, PU) (1 point)

```
RSA Public-Key: (1024 bit)
Modulus:  (n)
   00:c6:a7:16:71:79:e1:ac:4a:3c:36:67:26:f4:b5:
   03:32:d7:22:f2:43:5a:bd:ea:a4:88:1b:60:ac:ac:
   d2:e7:82:13:1f:29:53:2f:3c:84:06:3e:93:4c:91:
   b2:87:81:7b:2f:06:f4:3e:77:1f:17:6a:64:92:15:
   2d:10:f4:b6:bc:05:6a:3d:68:e3:52:05:9b:2f:ab:
   51:6b:55:96:1e:62:e4:3d:13:b4:95:1c:f8:83:5d:
   73:ad:9e:f2:d8:56:3c:ac:30:58:11:0f:e8:c9:09:
   00:2d:f3:f0:09:6d:e8:75:a7:ff:7d:26:5c:93:e0:
   94:68:ca:72:23:99:30:ab:d1
Exponent: 65537 (0x10001) (e)
writing RSA key
-----BEGIN PUBLIC KEY----- (PU)
MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQDGpxZxeeGsSjw2Zyb0tQMy1yLy
Q1q96qSIG2CsrNLnghMfKVMvPIQGPpNMkbKHgXsvBvQ+dx8XamSSFS0Q9La8BWo9
aONSBZsvq1FrVZYeYuQ9E7SVHPiDXXOtnvLYVjysMFgRD+jJCQAt8/AJbeh1p/99
JlyT4JRoynIjmTCr0QIDAQAB
-----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 shift substitution cipher: `psvclaolcpynpuphjfilyyhunl` (1 point)

`Decryption: ilovethevirginiacyberrange`

`Website used:` https://www.dcode.fr/caesar-cipher

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)

MD5 Hash: 53ca9be5f40f02cab06b4541b0d9c8ea

Website used: https://www.md5hashgenerator.com/

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