

## Lab Exercise 6 – Cryptography

Due Date: March 25, 2022 11:59pm  
Points Possible: 7

**Name:** Joseph Bannon (jbwar)

### 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 `mccrypt`

Mccrypt is a symmetric file and stream encryption utility for Linux and Unix that replaces the weaker `crypt` utility. Mccrypt 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.

Mccrypt 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 mccrypt
```

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

```
$ mccrypt --help
```

Mccrypt 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 mccrypt, use the following command:



```
$ mcrpyt --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)

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.

Use `mcrpyt` to encrypt your textfile. Mcrpyt 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.

```
$ mcrpyt 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)

```
E<N<[R???(?!(??mBV?!(???.GQ??1U??Y??W7?gU{S??pK>Po??m  
??B@'?=?43??A{??27N:2l??2~??5-!@9??V  
?C)???ж??B?q?+?y?c?x???'??MRÍ??2?=??DF?b??.rC5
```

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 `mcrpyt` with the `-d` switch to decrypt your file. Be sure to use the same passphrase as in step 3, above.

```
$ mcrpyt -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)

```
pub rsa3072 2022-03-22 [SC] [expires: 2024-03-21]
    859C336C43C87E36C29A9C02B1DBD047A6601D3B
uid          Joseph Bannon <josephpbannon@gmail.com>
sub rsa3072 2022-03-22 [E] [expires: 2024-03-21]
```

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.



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)

```
#####\j*#z
#####
#####%#####"#####A~#####7#####$I#####2#####_9y#####qtC#####Y#####
M#####fC]#####!n#####M#####<#####

D>j#####+#####H#####^'#####α#####5#####Y'#####B#####j#####;#####!d#####C#####Xø#####k#####U#####h#####kcRX(#####b#####
#####
#####I/#####<#####3Q#####0X#####N@u#####?#####8#####AzgF$+#####j#####Qi#####ýhS#####b#####>#####<#####
-&#####~#####R#####Y#####:#####
&g#####xw'#####i,#####%6q#####č#####k#####B#####2#####.#####K#####Ly#####oV#####<#####RxSQb#####
h#####`mi#####C~#####5#####+z#####+F#####'#####:#####5xŁcE#####K#####5#####j#####?#####+#####Qje||
#####h#####vHg#####n#####R#####`#####Q#####2#####z*;Q60
```

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)

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

```
mQGNBGI5JB4BDAC9Vm99RNycye6lW3wEOYlpNY81f9vRZ6wKVZh+xTmbRBI2VQNU
2xGNAr7YH9LE0tdik/vxgBYtgZSHcZw9oW4Fi0Wd9hPAN+KvUGEd6tAiwpPjLd3X
8UThu8MYee+Oc8mGRI60ERbdU95p0q4G0PmZ/0/b8xSeYcLuzVQvbtIC2yJQw9Ro
PzwhmskBYeptlwVvyj0YiYPu9SSmNuh+2w3wCUXoJQYU/fUyJVcsKoN+4W1NXPYv
OD6F/7RFBELm+nSLyzS8Kg6PdmYdfuhBQSC8w+CD+FoS6UywuC/HH9zZbqCWtIP
PyKb6MJHYp7zrlSdK7eIZk+1+csFN/FVvbJwDZvnuUysx8/moover0xdXiWbf9m
jkPiazbCVde6OjXZYEGTFgh+pnDj9u803S5OI/TAav+stMY5FLeL//dvVflfPGRC
UdWQIo60m+bC3t4h13EX+NHqnSV48H3379Z69r+sZXLqwmIZc/DThFuLU2dFky5Z
HqA8aZGOyFwduv8AEQEAAABQnSm9zZXBoIEJhbm5vbiA8am9zZXBocGJhbm5vbkBn
bWFpbc5jb20+iQHUBBMBCgA+FiEEhZwzbEPfjbCmpwCsduQR6ZgHTsFAMi5JB4C
GwMFCQPCZwAFCwkIBwIGFQoJCAAsCBByCAwECHgECF4AACgkQsduQR6ZgHTvlgwA
q0/wAnCoO8azlsYANvwVhglaw1NPMdPbttmZ5YUGCNZJQtfbv8GEBW6o0U7iSgGt
AvkQOvkCzLKM+3Pfk9MmU+KpoGJ4qEiUriEABa1i9MD1gHDSkd21vNn6RF8OZLJv
T7hTiLVYBhAgxcwoFNFoG2JZZtNJwo8D4H5ON0t7DmfKhnZ6X+HPnQNNxEDP4MM
OGWBqrivdYNhwM8UMsMQx1sB5cTm70A3fscgsuRdr1H49IX4AITAUg4WlsGYn0qE
m9eLUQoSxgqhDE9J5CgXvqvTt0YD8moIPHZfwQ17WtYX5hCRaxh7IJ48bFtf2Umk
f1pSVjHpOwGUWfZjhIXOFaiNouH5UVR0kosGOuEpFMRk8j9HUIJCw89aM+BaAD0d
yhRuBoja45deSKYLaS33xqBm6kwFP3T/dNleFz0lyGiu1kXKShlu/CdxKtSo0MEU
DiF9R1ZRkFPEy7yHjltCUKoiJmpcPDPmOZ3Ph2jzyCk4KoBsJ997/mvwrxCweh8L
uQGNBGI5JB4BDAD2o6VkMYJGSVSW/KBlgDsmVwH9VjU5zu26Q5wTz1JvJVyGBnML
PuRxzg/SnqzYTAG2IFszOuJpMJmdYf9J5IJ5cC808s2F1WMsZz/JOudmu5vTySoz
62WixUoQ7yZ2W3gC/yi+o4JipH8ide5Xv4b7AycrL8bXEPcmrDNNy3OqjsilHWUL
lcp+sdVfc68EPbtp4XhTISdvE976ZCel4TOUpbGrCcX2iSSbDu/LWRR5TlvLHKOR
yo16EDngXPbEDvSXRUX3GKP9O4n562y4Dx+tma/CsbUcaagnp+5T40Rxx/j/lKGz
dGvoDAITS25MXjudwDtA3Qoa1LJHiQjiTQmJYJ4ybLTQ3ERWZ+nx73kgHrqhu5v
7HUTs8HS6ZYQCHppb+cbWBRM/LJb4quTplcQUNEn//ZJ9fkFOOe4uvXXmc+3s2t
vMmqyYG0KmJLONEZCwoy9tiOpQB5iPG4MyQxCttTRelukQRss3llc+IOUSNUrson
qMO0urDdCvotin0AEQEAAyKbVQAQoAJhYhBIWcM2xDyH42wpqcArHb0EemYB07
BQJiOSQeAhsMBQkDwmcAAAOJELHb0EemYB0711ML/OP5hotnilrhprJu8X07TJXC
i42aNcSg2rM0rSzEBhejMzCH/itOmIer5ukjn40qFyGwuRFeHOGDeUiS1xPFYNrw
iEvGx6s3Xobr9JyEMZPbc9q4F4z3JlpoZEEJuHUQJGoRO7W1//JBgenG+Q2YTQQw
TPFXwmXa0nZiW7tf9qzaT07F9S2+dDSKZEawhBgMNTAxixAgn/3fGyfySCx9oIh
tUGML4fgiM89y7XzTRjK8t7lzbFA+DiTU2kh0YeHowJMcAmnjdc62Z5RTkyj6b/l
6w+qA7/gJ3SgAS3kmPqfrthoAllphuH47vhWwleif/BKM426us0HK6bHYvu+NeJq
RFra3LDf9lzpQ5iPGOdAJGVO48ypWcGRbzaXJewSwZPHkmvxWQK91Goxo6S++0Ba
9xvSx37fdA24atGmsneZJCeZCCpl1h03AjfwD9E9mkUm/EP8LkseDoy7aVX5VChX
F4F08D/tlXCCwlaYdpiJvNV/mhFiGXqMslrKPKgshg==
=iNyv
```

-----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_pair.key
RSA Private-Key: (1024 bit, 2 primes)
modulus:
 00:cf:a9:32:19:b0:4e:25:0e:00:e3:5f:53:3d:d0:
 a9:a6:78:fc:66:4d:4f:d0:30:65:f4:bf:2c:73:44:
 78:c7:fb:86:8c:cb:a1:d8:d1:bc:9d:2f:2d:0d:92:
 2a:53:b2:36:5a:1e:45:15:1e:f4:c6:93:23:10:e5:
 b0:93:36:99:88:ca:91:4b:34:5e:60:87:78:55:fa:
 b7:38:66:af:eb:a2:be:23:21:7f:f3:4b:f8:34:8a:
 4c:7f:a3:07:48:d4:80:45:cf:83:af:5c:dd:1c:9a:
 60:57:33:90:3f:d5:b8:09:cd:04:63:94:b7:b6:19:
 51:a2:8f:35:d4:bd:90:42:c1
publicExponent: 65537 (0x10001)
privateExponent:
 69:6b:65:69:b1:1d:1a:a6:8f:40:de:45:ad:dd:de:
 22:0a:cd:67:49:dc:38:be:39:24:14:61:06:6b:3e:
 3d:97:ac:e8:90:ff:aa:c8:5e:ce:15:02:f4:1e:bd:
 aa:1b:90:88:13:51:d2:b5:12:62:34:93:da:a2:20:
 0f:bb:ea:18:a2:8b:85:9f:0b:ef:47:b5:f3:1f:57:
 6f:16:63:f9:d5:cf:ed:6a:8f:f1:d5:f1:e0:5b:be:
 16:e2:53:16:32:0d:59:27:e8:2e:54:52:c0:60:27:
 96:0c:26:fc:7b:30:85:da:09:c3:18:53:ab:bc:fa:
 43:18:8b:c2:d0:86:a2:01
prime1:
 00:fa:33:ff:ad:fb:83:f0:1d:e4:86:72:c6:d2:e3:
 8c:d5:21:06:7f:16:15:4e:a1:d1:de:58:2e:05:bc:
 fa:bc:1c:84:03:60:83:e5:01:12:e9:31:6b:57:52:
 90:56:70:3c:57:d3:ff:47:69:85:39:ba:9d:cb:6d:
 25:d4:3a:1c:d1
prime2:
 00:d4:78:df:0a:b4:ce:8f:18:5e:ea:64:10:b4:13:
 a6:cb:b1:1f:b1:70:38:47:74:48:ce:29:44:b3:1c:
 c7:99:95:fa:6b:8f:43:12:48:3f:e6:c4:d7:7c:aa:
 73:a9:97:d9:71:11:53:dd:96:cb:fd:ef:ff:a6:3a:
 e0:56:e9:82:f1
```



```
coefficient:
 00:c4:1b:a9:00:c9:fe:25:2f:c9:c0:e7:37:34:15:
 7c:18:ce:97:f7:e3:eb:d7:83:46:40:50:36:f8:14:
 c8:6d:be:34:55:e9:67:08:21:4a:fa:fd:9c:14:1b:
 82:28:2d:25:e1:3d:9b:3f:9d:95:41:80:09:0c:c4:
 55:f7:f4:8b:f3
writing RSA key
-----BEGIN RSA PRIVATE KEY-----
MIICXgIBAAKBgQDPqTIZsE4lDgDjX1M90KmmePxmTU/QMGX0vyxzRHjH+4aMy6HY
0bydLy0NkipTsjZaHkUVHvTGkyMQ5bCTNpmIypFLNF5gh3hV+rc4Zq/ror4jIX/z
S/g0ikx/owdI1IBFz40vXN0cmmBXM5A/1bgJzQRjLe2GVGijzXUvZBCwQIDAQAB
AoGAaWtlabEdGqaPQN5Frd3eIgrNZ0ncOL45JBRhBms+PZes6JD/qshezhUC9B69
qhuQiBNR0rUSYjST2qIgD7vqGKKLhZ8L70e18x9XbxZj+dXP7WqP8dXx4Fu+FuJT
FjINWSfoLLRSwGAnlgwm/HswldoJwxhTq7z6QxiLwtCGogECQQD6M/+t+4PwHeSG
csbS44zVIQZ/FhV0odHeWC4FvPq8HIQDYIPLARLpMwtXUpBWcDxX0/9HaYU5up3L
bSXU0hzRAkEA1HjfCrTOjxhe6mMQtB0my7EfsXA4R3RizilEsxzHmZX6a49DEkg/
5sTXfKpzqZfZcRFT3ZbL/e//pjrgVumC8QJBANHRr3WIubEwcEcrk36g4qaMpnUG
40FrJKMwEZAawVZhQ1Yg7dQAjcwdrURgOngpO3tvMZYTOgbHSxKcmphWekjECQQCs
G+kYhB0aSDSCi2IYbJ68+xJgKjX8c46SKU20BQk00TXrsNW01n+k0Ch5nJ0qcI27
kDyuD3vYjHM61RBqdqcRAkEAXBupAMn+JS/JwOc3NBV8GM6X9+Pr14NGQFA2+BTI
bb40VeInCCFK+v2cFBuCKC0l4T2bP52VQYAJDMRV9/SL8w==
-----END RSA PRIVATE KEY-----
```

Note:

p is prime1. q is prime2. n is modulus. e is public exponent. d is private exponent. PR is 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, integer e, PU) (1 point)

```
student@kali:~$ openssl rsa -text -pubin -in public_key.key
RSA Public-Key: (1024 bit)
Modulus:
 00:cf:a9:32:19:b0:4e:25:0e:00:e3:5f:53:3d:d0:
 a9:a6:78:fc:66:4d:4f:d0:30:65:f4:bf:2c:73:44:
 78:c7:fb:86:8c:cb:a1:d8:d1:bc:9d:2f:2d:0d:92:
 2a:53:b2:36:5a:1e:45:15:1e:f4:c6:93:23:10:e5:
 b0:93:36:99:88:ca:91:4b:34:5e:60:87:78:55:fa:
 b7:38:66:af:eb:a2:be:23:21:7f:f3:4b:f8:34:8a:
 4c:7f:a3:07:48:d4:80:45:cf:83:af:5c:dd:1c:9a:
 60:57:33:90:3f:d5:b8:09:cd:04:63:94:b7:b6:19:
 51:a2:8f:35:d4:bd:90:42:c1
Exponent: 65537 (0x10001)
writing RSA key
-----BEGIN PUBLIC KEY-----
MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQDPqTIZsE4lDgDjX1M90KmmePxm
TU/QMGX0vyxzRHjH+4aMy6HY0bydLy0NkipTsJZaHkUVHvTGkyMQ5bCTNpmIypFL
NF5gh3hV+rc4Zq/ror4jIX/zS/g0ikx/owdI1IBFz4OvXN0cmmBXM5A/1bgJzQRj
lLe2GVGijzXUvZBCwQIDAQAB
-----END PUBLIC KEY-----
```

n is modulus. e is exponent. PU is 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.

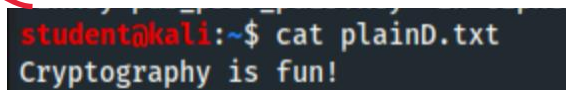
Next, decrypt the binary using the following command:





```
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)

The plain text decryption is: ilovethevirginiacyberrange

The shift was 19 to decrypt (7 to encrypt).

**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)

The MD5 has of the sentence is: 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

