**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 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)

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 **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)

E<N<ʄR�����!(����mBV�!���.GQ���1U��Y��W7�gU{S��pK>Po����m���B@'�=��43��A{���27Ń:2l��֩�2~��5˨@9��Ѷ �C)��ж��B�q�+ډy�c�x���`����MRĺ��2�=֍��DF�b��.гC5

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)

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�.�� �Ө?�

M��fC]��!n� M����<�

�D>j�]�+��H��^'����α��5��Y'�B��j��;�!d�C�Xǿ�?k�U�h�kcRX(�b�0�"� ����I/���<��3Q�0X�N@u���?�8�AzgF$+��j�Qi��Ӳ׌hS���b���>�<`�-&�����~�R�Y��:�׌� &g���֧�xw'�i,�#%6q��ċ��k��B�2���.�K�Ly���oV����<�RxSQb����h�`mi�������C~�5�+z�+F����'���:�5xȻcE�K����5�j���?�ͱ��Qje]lɂ�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+Oc8mGRl60ERbdU95p0q4G0PmZ/0/b8xSeYcLuzVQvbtlC2yJQw9Ro

PzwhmskBYeptlwWVyj0YiYPu9SSmNuh+2w3wCUXoJQYU/fUyJVcsKoN+4W1NXPyv

OD6F/7RFBeLm+nSLyzS8kGq6PdmYdfuhBQSC8w+CD+FoS6UywuC/HH9zZbqCWtlP

PyKb6MJHYp7zrlSdk7eIZk+1+csFN/FVvbJjwDZvnuUysx8/moover0xdXiWbf9m

jkPiazbCVde6OjXZYEgTFgh+pnDj9u803S5OI/TAvV+stMY5FLeL//dvVflfPGRC

UdWQIo60m+bC3t4h13EX+NHqnSV48H3379Z69r+sZXLqwmIZc/DThFuLU2dFky5Z

HqA8aZGOyFwduv8AEQEAAbQnSm9zZXBoIEJhbm5vbiA8am9zZXBocGJhbm5vbkBn

bWFpbC5jb20+iQHUBBMBCgA+FiEEhZwzbEPIfjbCmpwCsdvQR6ZgHTsFAmI5JB4C

GwMFCQPCZwAFCwkIBwIGFQoJCAsCBBYCAwECHgECF4AACgkQsdvQR6ZgHTvlwgwA

q0/wAnCoO8azlsYANvwVhgIaw1NPMdPbttmZ5YUGCNZJQtfvb8GEBW6o0U7iSgGt

AvkQOvkCzLKm+3Pfk9MmU+KpoGJ4qEiUriEABa1i9MD1gHDSKd21vNn6RF8OZLJv

T7hTiLVYBhAgxcwoFNFoG2JZZtNJwo8D4H5ON0t7DmfmKhnZ6X+HPnQNNxEDP4MM

OGWBqrivdYNhwM8UMsMQx1sB5cTm70A3fscgsuRdr1H49lX4AlTAUg4WlsGYn0qE

m9eLUQoSxgqhDE9J5CgxvqvTt0YD8moIPHZfwQ17WtYX5hCRaxh7IJ48bFtf2Umk

f1pSVjHpOwGUWfZjhIXOFaiNouH5UVR0kosGOuEpFMRk8j9HUJOcw89aM+BaAD0d

yhRuBoja45deSKYLaS33xqBm6kwFP3T/dNleFz0lyGiu1kXKShlu/CdxKtSo0MEU

DiF9R1ZRkFPEy7yHjItCUKoiJmpcPDPmOZ3Ph2jzyCk4KoBsJ997/mvwrxCweh8L

uQGNBGI5JB4BDAD2o6VkMYJGSVSW/KBlgDsmVwH9VjU5zu26Q5wTz1JvJVyGBnML

PuRxzg/SnqzYTAg2IFszOuJpMJmdYf9J5IJ5cC808s2F1WMsZz/JOudmu5vTySoz

62WixUoQ7yZ2W3gC/yi+o4JipH8ide5Xv4b7AycrL8bXEPcmrDNNy3OqjsilhWUL

Icp+sdVfc68EPbtp4XhTISdvE976ZCeI4TOUpbGrCcX2iSSbDu/LWRR5TlvLHK0R

yo16EDngXPbEDvSXrUX3GKP9O4n562y4Dx+tma/CsbUcaagnp+5T40Rxx/j/lKGz

dGvoDAiTS25MXxjudwDtA3Qoa1LJHiQjiTQmJYJ4ybLTQ3ERWZ+nx73kgHrqhu5v

7HUTs8HS6ZYQCHppb+cbWBRM/LJb4quTplcQUNEn//ZJ9fkFOOoE4uvXXmc+3s2t

vMmqyYG0KmJLONEZCwoy9tiOpQB5iPG4MyQxCttTReIukQRss3lIc+lOUSNUrson

qMO0urDdCvotIn0AEQEAAYkBvAQYAQoAJhYhBIWcM2xDyH42wpqcArHb0EemYB07

BQJiOSQeAhsMBQkDwmcAAAoJELHb0EemYB0711ML/0P5hotnilrhprJu8X07TJXC

i42aNcSg2rM0rSzEBhejMzCH/itOmIer5ukjn40qFyGwuRFeHOGDeUiS1xPFYNrw

iEvGx6s3Xobr9JyEMZPbc9q4F4z3JIpoZEEJuHUQJGoRO7W1//JBgenG+Q2YTQQw

TPFXwmXa0nZiW7tf9qzaT07F9S2+dDSKZEawhBgMNTAxiXAgn/3fGyfYxSCx9oIh

tUGML4fgiM89y7XzTRjK8t7lzbfA+DiTU2kh0YeHowJMcAmnjdC62Z5RTkyj6b/l

6w+qA7/gJ3SgAS3kmPqfrthoAIIphuH47vhWwleif/BKM426us0HK6bHYvu+NeJq

RFra3LDf9IzpQ5iPGOdAJGVO48ypWcGRbzaXJewSwZPHkmvxWQK91Goxo6S++0Ba

9xvSx37fdA24atGmsneZJCeZCCpl1h03AjfwD9E9mkUm/EP8LkseDoy7aVX5VChX

F4F08D/tlXCCwlaYdpiJvNV/mhFiGXqMsIrKPKgshg==

=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:

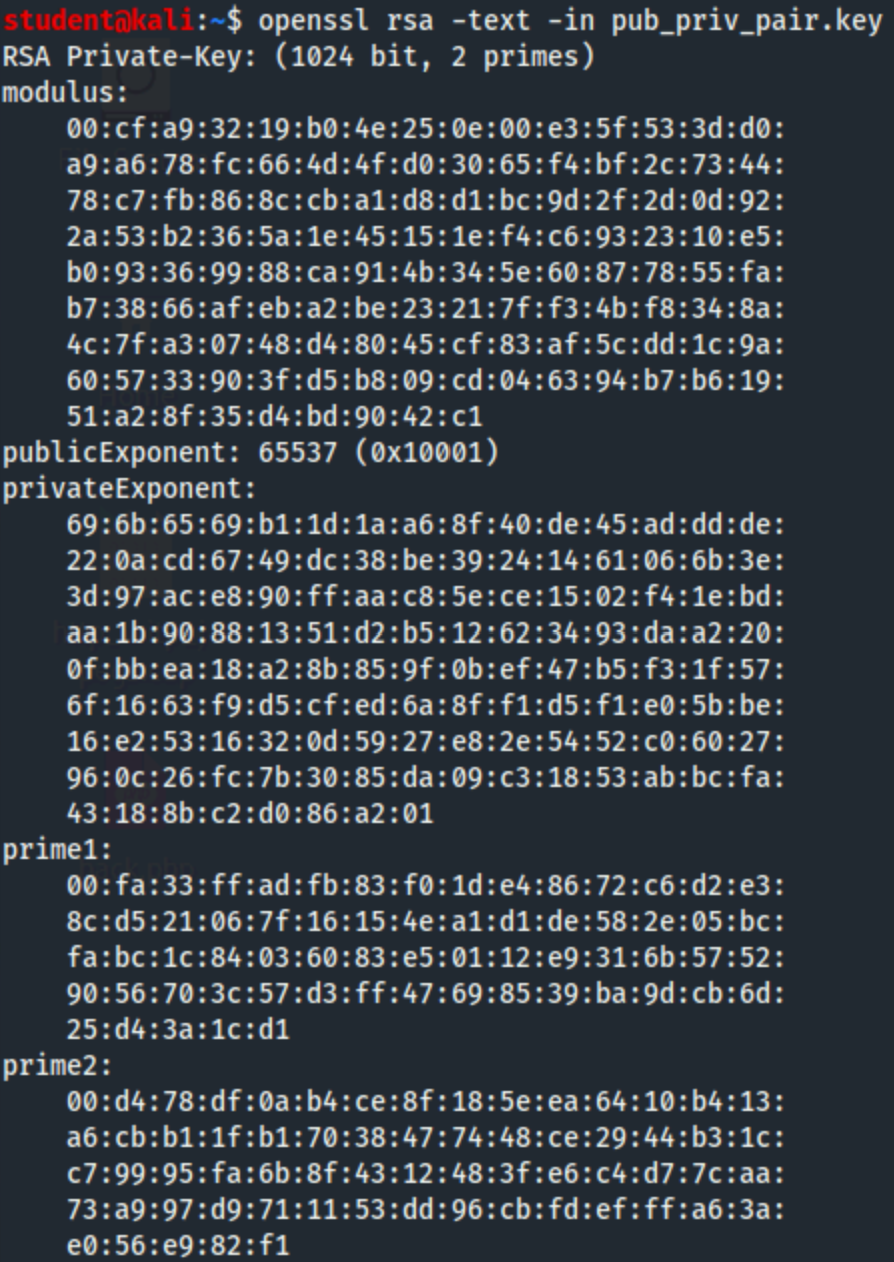
**openssl genrsa -out pub\_priv\_pair.key 1024**

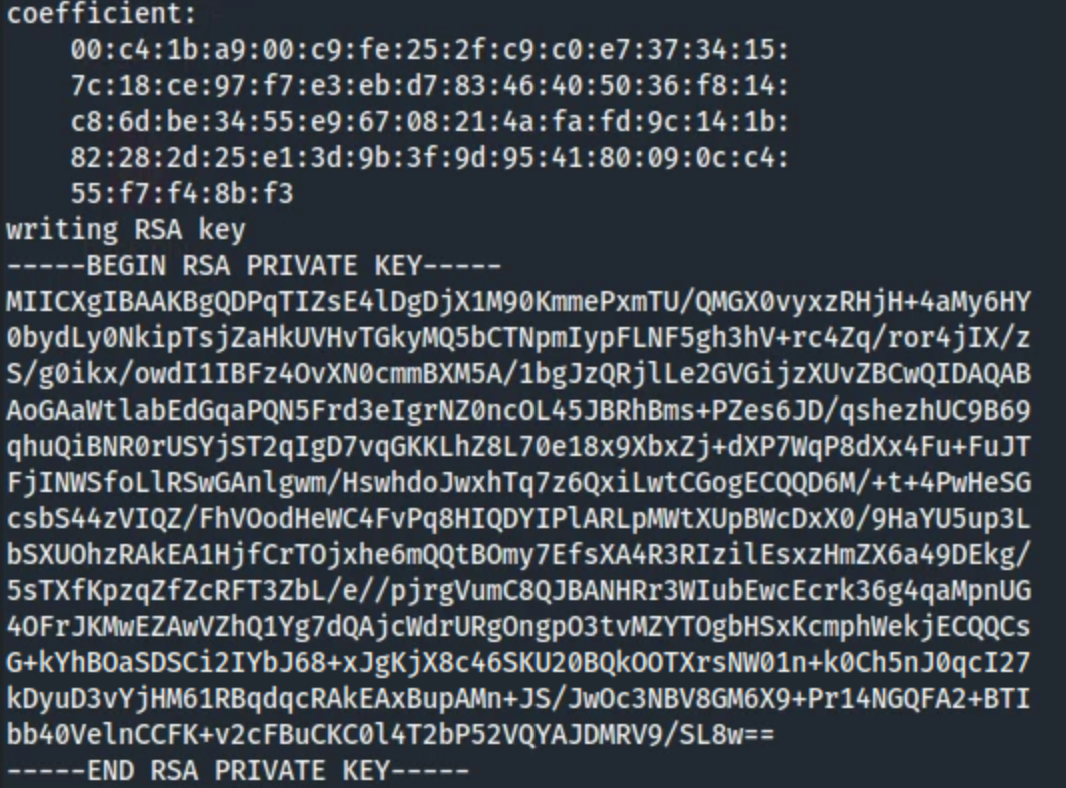
The **genrsa** flag lets **openssl** know that you want to generate an RSA key, the **-out** flag specifies the name of the output file, and the value **1024** repre­sents the length of the key. Longer keys are more secure. Remember: don’t share your private key with anyone. You can view the key pair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





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)



n is modulus. e is exponent. PU is public key.

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

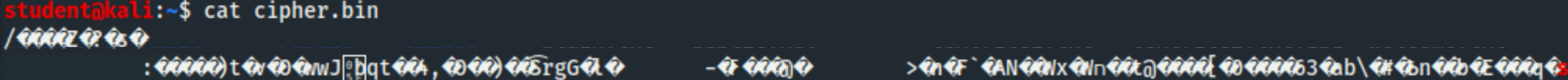
**echo "Cryptography is fun!" > plain.txt**

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**

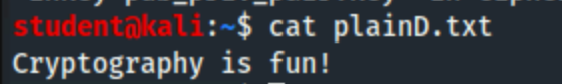
Notice that we included the **-oaep** flag. Secure implementations of RSA must also use the OAEP algorithm. Whenever you’re encrypting and decrypting files using **openssl**, be sure to apply this flag to make the operations secure.

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