CS 3710 Introduction to Cybersecurity
Term: Spring 2022

Lab Exercise 6 – Cryptography

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

Name: Hanzhang Zhao (hz9xs)

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)

```
m@rijndael-128 cbcmcrypt-sha1pSZi;|Xksha17
                                             5;v
                                                    ,xqc)b1kUA96D[0V{HꞔC#PM$Pe_c8!1tĪnJ
u]h(^w+d̲x~-]*|!pAK:aLEb&!          4$~dO;{xAAA"5SoU-g@t1
```

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 2022-03-24 [SC] [expires: 2024-03-23]
      A31099DC9761AE4A61668B84E1F49ABFD37B549E
uid                      zhaohanzhang <hz9xs@virginia.edu>
sub   rsa3072 2022-03-24 [E] [expires: 2024-03-23]
```

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)

```
uNث
Éb741ThJX'oAb3E>-Y*XV/1j-E Vt'812$;J! j"\w^v+uBf#CNDfx;euBLK2c
k$O"9ꟷG7jɴK;MG(pb#q]ıę;i;}n>@l{FbvK
3        RJ(26PDNd[,=늪/_q   !Mni/>>e\N&'T{/CRV;_
   Ov}izWhZzF_O\`tpTUFlk[y0"v>&Ɨ&@K7-/è9%
;
RwHx?}HLň   8]H,')
```

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)

mQGNBGI74gUBDADMcYWRkmFGaBvFYSBsATkHmZ4RTzGDLr/uFN/fSnIjXjLf50AI
ItIMQ3L3ASn74HTuZ7WPz44r4gKAFVnf8ZQbbclOWzEQgDjWlv0GtlLetxAxO5Ft
yF43uEUig/rKDkzhKP6G8+Qw0Dt1Qraqy9Jf5bwzytKMkuOCOaCBYGeZlv7V/Xe8
rInQ1FtJ91DFfw7fnmGVi6NZ+4tbJX4LNZu4bYCcTSfrDDKRFiPMu6E4JNIuLgPo
TKa3KnhMbQc6pnBMpcJAc0g47KE2c2qyo5os/xfpBXkDyHqfNSQheNxF8BPqH6Z6
dBLFNANSJBfwPq3rK25FFJorzNNvjyQcVZbkSJvNgPQPDtZYV4bM6oGznkgr5X3S
PK/6OrMAuIu3esvBJoYbkGkpqp2orTUUfpztINHbUMapN8midFH5/iQ/Pu2ex3+y
TaW/3YsVdMRbTVJcb49XVoXi6F7bPdA4Ukeqi7kAPXFf3pWd+WKPm2D++0GLbw4z
f5pLHjG4hXjzIeMAEQEAAbQhemhhb2hhbnpoYW5nIDxoejl4c0B2aXJnaW5pYS5l
ZHU+iQHUBBMBCgA+FiEEoxCZ3JdhrkphZouE4fSav9N7VJ4FAmI74gUCGwMFCQPC

ZwAFCwkIBwIGFQoJCAsCBBYCAwECHgECF4AACgkQ4fSav9N7VJ594gwAtC4Kh6aw
vXS2y0kwScBUJ6jPk8egCTTrVph60Rmm0w2RdHmkujX0TIsLq7pw61OM6/LOmTmj
hCurK6D3+/gVyKX2u0JTin5BMUbE7Z1eQYXCiCftrew0553VN/nc33kF+28zv9mJ
7QUydtH7uaQJll5QpOmier9fP8Y2lINsoqY8CN4oJ+7t4J9JOUMuAo/Uexx2nsbK
e0puM+Iu2nasngUaoPgwS4cqoRe98YL8UXHvJ4ttJ5t++geg4CIiACQP8GEo27Ic
10kjWcjhrHjaxaSZpcg5riBalcLWytTlymmmF7O9z5XGLUAqw9rP4Fp4SrKAJJC7
oYbGwjJrXStg/HvlQ/jj54L0YSff5ccehGwge8djOiFdPgfCUCb2smQC67VJ6fr6
WwQALqxUQ+U9AQMcpaHlAmPFh0vu9btLRPAAikzkxkFto2HF6hNjxhdAWh9NPkkz
D28O6rvVEkrgvK3A1hpD3+X1xDPjunDR4UJPAcR2j03qJQBmG9oC0bUzuQGNBGI7
4gUBDADjG4vOoNSRtffjCAP58VMjD3Qn26ORyHVZN11YxNGX5UIwCc5thn8MfsE2
+SVO9O1YLAooBe5zrNNFb7GmLHe+VTw5PiXA9ozXIsmplmVSfOEXe6KS+a6vCmVr
wJlqN+vRNMmKeteKXXH/9Cq+OuujoJzOUHN0U58KzMu3RyR8vjh/NUvknaEs2ZDT
IogOFAkhQG53mAVoqkWbW3kxFxM7ZjK18rN4cvgsMmPhGATRlCwcMZQ9rNMcwIB3
87LSGSGNbtctEIfhW87+OidoWBI7+qhpb35/Nlr3+Us4cYQsgY6lUsykkfcv98pD
TVnKCNtvpr+iPonC+CVaIZ1udGXMhqfcvR/JqDUP1Hws5ujdk4YS3lYGsxcttNAz
6WWfKXVqKJuHBLWG+KgLXq/EMYeOjVxtle1frybZypMTkaQ1PLv45XOTXWULo1Zo
ulq5PxnNHfV2lnYkbcwdWvxyZJx4yxuCN04c4uMge7dbYLewOxtwUyQ4AChQdo3B
7aVelyEAEQEAAYkBvAQYAQoAJhYhBKMQmdyXYa5KYWaLhOH0mr/Te1SeBQJiO+IF
AhsMBQkDwmcAAAoJEOH0mr/Te1SeWykMALlpHx3mlRYjpSsydi+Ntzn1Cvi31SAj
P4QQRSp+GjUCqO6enC06Lj0EfOAv9bHm2aMEaqcZy/M+3y8I0kKxzIekrDxENfs4
OHr0B9P14gvTwxLRXXoe+zPn/ba4N3Gnjl/FROYS5Qf18sfJk+Uv2mOXV0jmOfZK
8UG6nB/pEBr4FYb54sDy+yLMSxJXanrisK/GXLcFwwMM68tUQGAzMaqj9reJHfbP
f3Wgoo/KzvW1/ZryLRC74qiQnAqYwaQrk/NzRT+YO2KSs5KgA7sB7U3AMKsGF+i8
/qkPxhifoVaSt6pTV1llxQ/DFBoj4vPOTzdg5RPazQXdLCXWKxrxeIziI0tVT7hC
+SS8wBoT5MV6AVkg1Q/usjIrqqJmdX+w7ERRFCXr96vzuIR44QDuy38/8Igavo4/
4dCfxZYUaV3qsIc66fD9wMYLOpOpTNWYZDxqBQshrH8XmfUCchWFaIvc98tPC3Wl
oSbAB80Ik3m7IXdwoeI0wiEBefBHAvUXuQ==
=ZUKe

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.

n:

```
modulus:
    00:a3:b4:45:92:1b:30:34:c7:df:6e:63:1e:20:9c:
    da:f4:46:34:7b:b7:02:14:6c:a5:1d:74:7f:4a:ad:
    ad:62:99:b5:1f:e6:4d:84:0e:46:22:bf:4d:89:51:
    dc:c7:ee:e8:a3:53:17:9e:62:5a:05:cf:41:5d:ab:
    6a:32:7e:40:75:3c:ed:73:38:d3:6a:fc:7a:ee:2e:
    b7:18:fe:0b:65:c3:47:32:19:0d:6e:ca:e1:cb:1d:
    a2:12:f3:15:c2:25:3e:a3:9e:3f:85:7d:d4:b4:da:
    c3:50:4a:d8:ed:79:fd:a6:f4:56:da:4a:54:d4:6e:
    4b:96:4c:04:34:53:c6:21:35
```

integer e:

```
publicExponent: 65537 (0x10001)
```

d:

```
privateExponent:
    00:83:87:64:1e:70:19:db:4e:7c:06:85:3c:bf:97:
    47:94:dc:93:6e:93:2b:e8:9d:22:4c:f8:3d:0e:13:
    5d:2b:cb:b3:eb:5d:6f:0a:9b:2d:5e:dd:b5:be:8f:
    37:84:ac:3f:de:79:f4:90:1d:15:97:75:5e:5f:94:
    4a:4f:27:81:41:67:f3:16:03:b4:0d:f6:c2:79:86:
    1b:eb:ce:09:ce:65:23:2c:2c:a6:eb:45:eb:26:2c:
    e9:30:1b:2b:1e:4c:6f:28:59:57:0d:98:cb:d2:33:
    06:81:ce:6b:0e:5c:e9:d9:03:5f:16:92:e5:82:db:
    f4:48:71:9f:92:06:f1:ee:01
```

p:

```
prime1:
    00:d4:60:b7:cf:3d:48:3a:4a:e8:46:00:48:00:46:
    ec:01:7f:93:4b:fc:83:6c:7f:8d:ef:5a:2f:e2:d8:
    46:35:1a:c5:37:27:7d:91:d4:28:f4:72:64:75:65:
    85:7a:2f:4f:c9:17:23:c8:af:67:7c:bf:16:a4:a8:
    12:31:6a:27:d9
```

q:

```
prime2:
    00:c5:54:30:af:00:8a:a1:6f:ce:cb:20:d4:fc:e2:
    70:f8:d8:92:5b:17:7e:2b:7a:ae:f1:ff:8f:08:04:
    93:3c:d9:f2:6e:40:d5:fb:97:7a:b6:d2:b9:ca:cf:
    22:43:35:4d:89:69:20:21:fc:71:36:fa:8c:12:33:
    96:8d:10:a6:bd
```

PR:

```
-----BEGIN RSA PRIVATE KEY-----
MIICXQIBAAKBgQCjtEWSGzA0x99uYx4gnNr0RjR7twIUbKUddH9Kra1imbUf5k2E
DkYiv02JUdzH7uijUxeeYloFz0Fdq2oyfkB1PO1zONNq/HruLrcY/gtlw0cyGQ1u
yuHLHaIS8xXCJT6jnj+FfdS02sNQStjtef2m9FbaSlTUbkuWTAQ0U8YhNQIDAQAB
AoGBAIOHZB5wGdtOfAaFPL+XR5Tck26TK+idIkz4PQ4TXSvLs+tdbwqbLV7dtb6P
N4SsP9559JAdFZd1Xl+USk8ngUFn8xYDtA32wnmGG+vOCc5lIywsputF6yYs6TAb
Kx5MbyhZVw2Yy9IzBoHOaw5c6dkDXxaS5YLb9Ehxn5IG8e4BAkEA1GC3zz1IOkro
RgBIAEbsAX+TS/yDbH+N71ov4thGNRrFNyd9kdQo9HJkdWWFei9PyRcjyK9nfL8W
pKgSMWon2QJBAMVUMK8AiqFvzssg1PzicPjYklsXfit6rvH/jwgEkzzZ8m5A1fuX
erbSucrPIkM1TYlpICH8cTb6jBIzlo0Qpr0CQQDLbjmjUVg2PjpvkyVk3oqChOlb
B+37p9MVSpZD/FaD17jBNBqb7VWtkUWZDj3k5BzQylGEkj/WxlOyNSv4GW9BAkA6
KZ+T5Y/3F5KXctz0kPYFkyhIaztIK4gnUlFhZp/TftYjQz8PQk3sV15l9fm+X5a0
GuVavyLLtrvUnZoRiEftAkAN8cMZ6zfI9zkZoW86FNBaV9/F0xgpUBrobtfYrVRf
wKWB5BOrQc8AkbpuxPW2BXDdJjsdVZR/ooL9X/BfIpo4
-----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, integer e, PU) (1 point)

RSA Public-Key: (1024 bit)

n:

```
Modulus:
    00:a3:b4:45:92:1b:30:34:c7:df:6e:63:1e:20:9c:
    da:f4:46:34:7b:b7:02:14:6c:a5:1d:74:7f:4a:ad:
    ad:62:99:b5:1f:e6:4d:84:0e:46:22:bf:4d:89:51:
    dc:c7:ee:e8:a3:53:17:9e:62:5a:05:cf:41:5d:ab:
    6a:32:7e:40:75:3c:ed:73:38:d3:6a:fc:7a:ee:2e:
    b7:18:fe:0b:65:c3:47:32:19:0d:6e:ca:e1:cb:1d:
```

```
    a2:12:f3:15:c2:25:3e:a3:9e:3f:85:7d:d4:b4:da:
    c3:50:4a:d8:ed:79:fd:a6:f4:56:da:4a:54:d4:6e:
    4b:96:4c:04:34:53:c6:21:35
```

Integer e:

Exponent: 65537 (0x10001)

PU:

```
-----BEGIN PUBLIC KEY-----
MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQCjtEWSGzA0x99uYx4gnNr0RjR7
twIUbKUddH9Kra1imbUf5k2EDkYiv02JUdzH7uijUxeeYloFz0Fdq2oyfkB1PO1z
ONNq/HruLrcY/gtlw0cyGQ1uyuHLHaIS8xXCJT6jnj+FfdS02sNQStjtef2m9Fba
SlTUbkuWTAQ0U8YhNQIDAQAB
-----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)

i love the virginia cyber range

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