CS 3710 Introduction to Cybersecurity
Term: Spring 2023

# Lab Exercise 6 – Cryptography

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

**Name:** **Sidhardh Burre**

## 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)

I couldn't get wget to work so I just inspected the file and copy-pasted manually. The file looks like:

```
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)

These are the contents of textfile1.txt.nc, the product of textfile1.txt after encryption.


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)

```
student@kali:~/Documents/CS3710/lab06$ gpg --gen-key
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: Sidhardh Burre
Email address: ssb3vk@virginia.edu
You selected this USER-ID:
    "Sidhardh Burre <ssb3vk@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 BB725E45C5745004 marked as ultimately trusted
gpg: directory '/home/student/.gnupg/openpgp-revocs.d' created
gpg: revocation certificate stored as '/home/student/.gnupg/openpgp-revocs.d/5E1799C4173EA7F7CBE3CC96BB725E45C5745004.rev'
public and secret key created and signed.

pub   rsa3072 2023-03-18 [SC] [expires: 2025-03-17]
      5E1799C4173EA7F7CBE3CC96BB725E45C5745004
uid                      Sidhardh Burre <ssb3vk@virginia.edu>
sub   rsa3072 2023-03-18 [E] [expires: 2025-03-17]
```

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)

I couldn't get wget to work so I just inspected the file and copy-pasted manually. The file looks like:

```
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)

```
student@kali:~/lab06$ cat textfile2.txt.gpg
[binary PGP data omitted]
student@kali:~/lab06$
```

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

```
$ rm textfile2.txtgpg
$ 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)

Assuming you want the contents of public.key:

```
student@kali:~/lab06$ cat public.key
-----BEGIN PGP PUBLIC KEY BLOCK-----

mQGNBGQV1acBDADq4l34U39fUOv4oDlBwdK8sV8NAGfCZ3u1b1OrTml7t6j/J+fV
TpWRLA6aJ8hbUFIGZXncVxA0Z54AH2UZRtPZOECVExcMyB7y79wWJa8FFbqgKISa
7IZ22BdeYZ23d3vDJTZdyls32Aft6YpccZqHLzlYUlGaqYKrHbBVHvl4IVpWr6gv
d3gmj8oiktIv9cUU8Llm48WAevIzQNO4Sy4vVnXhQm4yxZhNf6EQqKGYP839joo2
h411wujpN3CwdvzVoM8mx+3+nUE5x9gsr2cS+D6nbV7YP3t/NNVwT8nfmpqaN3Z3
XIe5lbQg0OmZnkhZHOMkQTalS3leVegCKu9k5GApJ9/bf9v02F8jHGn/ZNpJwl8t
fsemmD1HBBcBwY1djVrgpstIrr9bpc8jfiONyFNtPCS+UBPwC/2Dw4kHXjXenXon
Syjm87HAlMeKkf+p5j6XxHBVnGVTy+2YudntqpBC4N5K3AwfM1u3VUX6ZXycnhNR
Xo9PnOPO4oxpFLUAEQEAAbQkU2lkaGFyZGggQnVycmUgPHNzYjN2a0B2aXJnaW5p
YS5lZHU+iQHUBBMBCgA+FiEEXheZxBc+p/fL48yWu3JeRcV0UAQFAmQV1acCGwMF
CQPCZwAFCwkIBwIGFQoJCAsCBBYCAwECHgECF4AACgkQu3JeRcV0UAQgYgwAmqP/
SuE6+mlX6fNnKapPXQKIIUW57uh8xwYwpkbvDBLBipqa2eFlXZQxJJNiiIfjmT3F
oZHrRMeGVndt82WyFGQ/8VzGKhOT7SJIkAF1Pp7Jb296m5GyG1b55Jkl3fiQ+4mx
Qi6gEsTAnAZazbMFmQBMWlH/UUU2Smvn7/Fy8icb0rzXxFBnaxcIIX1ci+6BHo4/
ThyhRCTClp8ljqR1H5aiPYk5HlOTQDHL/xDJurBpBlUJk91nnYX9PtjeBq9ElnwF
Ap1GCHaEWsvoEMYwALe+ddgNWRtMfu6y9u7dlM9H/E5KowSCOmlUPeWkrgHwMwRF
o5+4TVer7SQY3BSX1cBbVq4z74BVYqcKpxeQpZQJZ5q9tBbchP7U9ByBegRHsAHT
+OjolVMyqye+G/fLkjya5HH4GquOLf/wJdCYJO/H0iwMgL2jQp4a3TVERtbOJ5ov
q+Q9dLqHNVee083XtcLtWZzK3pdOhNa6UoUkdjkzFnlaqBiQbRVLBCQYKsXUuQGN
BGQV1acBDADFUq2+K2xNsbX0mO4PfiI3yspCUagwiq3vSs/4Abqub52QQIW5Z/xG
MNLEZHshMT4ldabCpw65WlQVNyQmSft5j25j0ORzexhXOKzV3/hYFEjPODG3fgTx
s8F95plY/witZdMRCsFpxVwMd4rpDu8vTsxWxC0ugnPfN9W7y5UnBE6a+wGXLPD/
h2xvruyzzQL5simCnuQ2U7QDNJxh21/gb7JHoc9TdtUz7+B6Cj5gn1j1Z0odzRhf
8UJFa2aMWzE0raeCtnff43q0GXq4rKR/2L5aCFO+IZvVNIxl/9nBVIdQmahDvlif
Ki1GHuP593q2eBTiASB8kbkfzccDRZTvyJ+dsR7I/hSdM+hLg02W+rD9LlSxm356
+g5Y/fBPLoMjhvC0MQZQpyYqXL6Pr9Pl1CANqOkdpbqykRPJlmu60JxJkis7MZlU
Eon6kBtp7LZFdAMsp1DdRnRHj50BJmsW6DV2R1NSp09SzeHGauSDaJdAk6CPK0ZQ
ZYCHccCc560AEQEAAYkBvAQYAQoAJhYhBF4XmcQXPqf3y+PMlrtyXkXFdFAEBQJk
FdWnAhsMBQkDwmcAAAoJELtyXkXFdFAEW70L/3WvGuuOJTd5yM+xeLmXIBG1Hh3A
VqIPGu/8l6qZjT6qqJQ7LrRG6BWJTCT72xrqD5Dbjn8JaW/QEGzowGXO7oKuTH5Q
YXON5vREpDiulEMPWaoXGj8FyfwnnWVgtW0qo8Hg4Q/AU5F6+dItTyeksbC2jiV6
I0d43pcrLGL1VO/dLrwEoahBJBWrlrrFuuAlnUKaNcodrVJ7dMH64mldoXmVf/oM
qpI82LI8mJuigWsNMoyobvvwUTzmeBJ3I7zY5OGV9U7wUW11VNI1e1ixa/NzgQkv
MQjt7BGsvSGHeYnnRnqW1QWV0UAUzAcM77KB+wiIP3K+3D45fbH7aDsnXT6eCKXq
2AAXHUe9qQgJk7B7gnq/G6tuFi/T2hBOIbej+ZGaA5XIfZ0mg6PumxrjGQytDSxQ
3ucKoBuMYJVUG8zH/TYTYWhDEcPQ7aQ+Lj0LgLeVXTFlx/ovvck4NJMeZtWZT/OY
EXH1dfZ4bh39f9ChpXWIqt4E9qIFMtueXOUVqQ==
=CGnp
-----END PGP PUBLIC KEY BLOCK-----
student@kali:~/lab06$
```

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: Corresponds to n = pq
    00:ba:69:33:37:6c:b9:dd:5c:cb:60:21:3a:82:a8:
    72:8f:ca:59:3e:54:09:73:b4:2a:b7:19:2c:4a:38:
    ff:ad:e3:fb:e9:e2:a0:20:4b:6a:d9:ad:7d:9b:a4:
    9a:77:e9:8e:49:cd:ef:ea:54:83:0d:a1:e6:d0:ae:
    70:d1:a7:56:66:3c:33:26:be:30:3b:0b:72:54:e5:
    c7:74:a7:4c:9a:8e:58:39:5e:8c:00:d5:8c:f3:0a:
    5c:b7:29:23:b8:53:8d:e0:ba:37:da:75:f3:dd:aa:
    36:c0:46:c0:23:bf:aa:c9:fe:4b:c8:bc:27:03:ed:
    29:0c:14:a3:a1:21:1c:09:89
publicExponent: 65537 (0x10001) This is e.
privateExponent: This is d.
    00:b7:96:23:fa:d1:f9:bb:29:48:a2:c4:16:fb:d7:
    29:fc:b3:2c:71:56:12:79:01:57:90:3c:1a:82:7e:
    e0:50:41:d8:37:e0:1a:13:b4:32:e7:6f:15:e5:d3:
    96:cd:c2:17:80:58:71:90:36:eb:5a:e1:b8:90:8d:
    3c:4f:3e:9e:b7:1f:fc:85:4d:a0:82:df:2d:8b:0a:
    db:54:26:bd:6d:7d:08:0f:cb:d8:91:99:32:46:45:
```

16:0e:27:ed:7b:68:a0:2e:68:c2:8c:23:b9:f7:ac:
7a:b4:49:79:56:9a:6b:22:1c:2a:68:5c:18:db:39:
1a:94:fa:ad:fa:22:d8:02:81
prime1: **This is the value p (or q)**
00:f6:0a:15:8c:c3:b7:67:a2:fc:9c:46:95:af:f3:
85:1b:bb:eb:ae:dc:6c:c0:fd:c8:d7:23:2a:b1:fb:
4e:dc:3e:89:32:6b:e9:fb:dd:5d:04:e3:70:f4:a0:
a8:2d:f8:bc:bc:4f:45:c4:33:dd:b8:14:46:9a:78:
5c:c0:a1:85:27
prime2: **This is the value q (or p)**
00:c1:f5:22:aa:fd:c6:05:58:e3:0a:c7:71:6c:ce:
a8:d3:3d:30:49:dc:a5:45:68:95:3a:99:ce:37:f7:
ea:34:0f:27:c5:cb:38:cc:0e:04:5d:49:c9:a7:fe:
9c:1c:08:5f:38:4a:6c:59:cd:1b:79:88:36:bb:c6:
89:3a:75:09:cf
exponent1:
73:5b:01:05:91:91:c1:06:7a:d7:ae:84:6a:0d:8c:
00:17:d8:85:90:95:70:da:cc:3a:8a:23:6f:75:3b:
61:29:f4:db:6e:1b:33:5a:73:4d:62:71:c8:50:36:
e9:ee:f0:56:7d:f8:60:e8:4c:71:0d:18:99:3e:0b:
86:c9:74:0b
exponent2:
00:b5:6a:47:11:ee:a9:f0:48:72:9c:9b:6d:e1:1a:
ef:58:1d:ac:73:f9:b8:70:52:60:fa:5d:f0:3f:f3:
58:11:77:77:79:0f:1b:41:e9:7e:75:8f:55:da:17:
51:06:fb:61:ac:f2:17:a4:6b:2f:5e:9e:64:ab:80:
08:cd:2f:b9:8b
coefficient:
00:9d:98:d9:2a:56:ae:81:3b:5d:65:13:50:06:5c:
74:d5:93:78:88:8a:8b:4a:81:0c:c6:d2:22:5c:cb:
1c:c6:b0:dc:bf:ae:92:32:8a:99:e1:b6:75:cd:f2:
62:f3:f3:75:b5:fe:a7:5a:e1:32:7a:a2:30:d3:ec:
98:dd:49:84:d4
writing RSA key. **This is the private key PR**

-----BEGIN RSA PRIVATE KEY-----
MIICXgIBAAKBgQC6aTM3bLndXMtgITqCqHKPylk+VAlztCq3GSxKOP+t4/vp4qAg
S2rZrX2bpJp36Y5Jze/qVIMNoebQrnDRp1ZmPDMmvjA7C3JU5cd0p0yajlg5XowA
1YzzCly3KSO4U43gujfadfPdqjbARsAjv6rJ/kvIvCcD7SkMFKOhIRwJiQIDAQAB
AoGBALeWI/rR+bspSKLEFvvXKfyzLHFWEnkBV5A8GoJ+4FBB2DfgGhO0MudvFeXT
ls3CF4BYcZA261rhuJCNPE8+nrcf/IVNoILfLYsK21QmvW19CA/L2JGZMkZFFg4n
7XtooC5owowjufeserRJeVaaayIcKmhcGNs5GpT6rfoi2AKBAkEA9goVjMO3Z6L8
nEaVr/OFG7vrrtxswP3I1yMqsftO3D6JMmvp+91dBONw9KCoLfi8vE9FxDPduBRG
mnhcwKGFJwJBAMH1Iqr9xgVY4wrHcWzOqNM9MEncpUVolTqZzjf36jQPJ8XLOMwO
BF1Jyaf+nBwIXzhKbFnNG3mINrvGiTp1Cc8CQHNbAQWRkcEGeteuhGoNjAAX2IWQ
lXDazDqKI291O2Ep9NtuGzNac01icchQNunu8FZ9+GDoTHENGJk+C4bJdAsCQQC1
akcR7qnwSHKcm23hGu9YHaxz+bhwUmD6XfA/81gRd3d5DxtB6X51j1XaF1EG+2Gs
8hekay9enmSrgAjNL7mLAkEAnZjZKlaugTtdZRNQBlx01ZN4iIqLSoEMxtIiXMsc
xrDcv66SMoqZ4bZ1zfJi8/N1tf6nWuEyeqIw0+yY3UmE1A==

-----END RSA PRIVATE KEY-----

**Prime1, prime2 correspond to p, q. Modulus corresponds to n. The public exponent corresponds to e. The private exponent corresponds to d. And finally, the PR is essentially {private exponent, modulus}.**

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)

SA Public-Key: (1024 bit)
Modulus: **This is n**
00:ba:69:33:37:6c:b9:dd:5c:cb:60:21:3a:82:a8:
72:8f:ca:59:3e:54:09:73:b4:2a:b7:19:2c:4a:38:
ff:ad:e3:fb:e9:e2:a0:20:4b:6a:d9:ad:7d:9b:a4:
9a:77:e9:8e:49:cd:ef:ea:54:83:0d:a1:e6:d0:ae:
70:d1:a7:56:66:3c:33:26:be:30:3b:0b:72:54:e5:
c7:74:a7:4c:9a:8e:58:39:5e:8c:00:d5:8c:f3:0a:
5c:b7:29:23:b8:53:8d:e0:ba:37:da:75:f3:dd:aa:
36:c0:46:c0:23:bf:aa:c9:fe:4b:c8:bc:27:03:ed:
29:0c:14:a3:a1:21:1c:09:89
Exponent: 65537 (0x10001) **This is e**
writing RSA key
-----BEGIN PUBLIC KEY----- **This is the PU**
MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQC6aTM3bLndXMtgITqCqHKPylk+
VAlztCq3GSxKOP+t4/vp4qAgS2rZrX2bpJp36Y5Jze/qVIMNoebQrnDRp1ZmPDMm
vjA7C3JU5cd0p0yajlg5XowA1YzzCly3KSO4U43gujfadfPdqjbARsAjv6rJ/kvI
vCcD7SkMFKOhIRwJiQIDAQAB
-----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)

Used a little C++ for this. Came up with "ilovethevirginiacyberrange". This is a 19-character shift. So, a maps to t.

*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 hash for this sentence looks like: 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