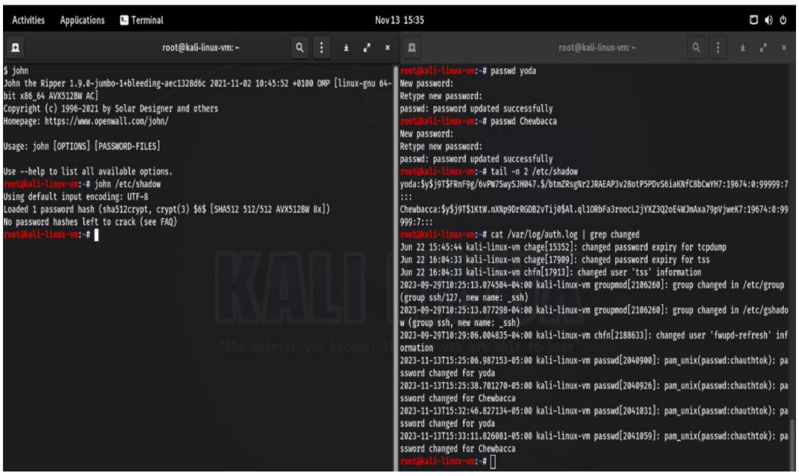
Assignment 5

1. **a. Describe at least 2 tools related to cryptography (chapter 13 with relevant screenshots, that demonstrates its working.)**

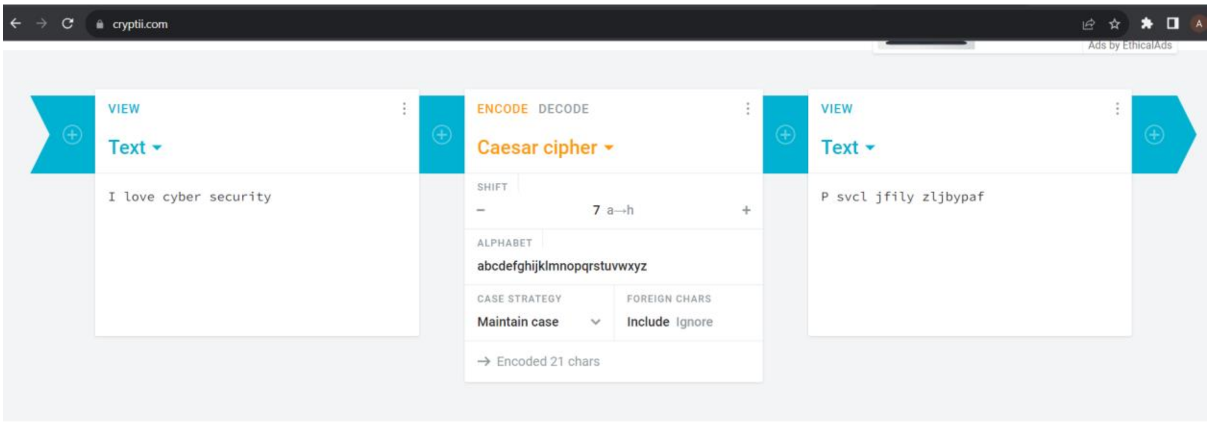


```
$ john
John the Ripper 1.9.0-jumbo-1+bleeding-aec1328d6c 2021-11-02 10:45:52 +0100 OMP [linux-gnu 64-
bit x86_64 AVX512BW AC]
Copyright (c) 1996-2021 by Solar Designer and others
Homepage: https://www.openwall.com/john/

Usage: john [OPTIONS] [PASSWORD-FILES]

Use --help to list all available options.
root@kali-linux-vm:~# john /etc/shadow
Using default input encoding: UTF-8
Loaded 1 password hash (sha512crypt, crypt(3) $6$ [SHA512 512/512 AVX512BW 8x])
No password hashes left to crack (see FAQ)
root@kali-linux-vm:~# 
```

```
root@kali-linux-vm:~# passwd yoda
New password:
Retype new password:
passwd: password updated successfully
root@kali-linux-vm:~# passwd Chewbacca
New password:
Retype new password:
passwd: password updated successfully
root@kali-linux-vm:~# tail -n 2 /etc/shadow
yoda:$y$j9T$FRnF9g/6vPW7SwySJH047.$/btmZRsgNr2JRAEAP3v28otP5PDvS6iaKNfC8bCwYH7:19674:0:99999:7
:::
Chewbacca:$y$j9T$1KtW.nXNp9DrRGDB2vTij0$Al.ql1ORbFa3roocL2jYXZ3Q2oE4WJmAxa79pVjweK7:19674:0:99
999:7:::
root@kali-linux-vm:~# cat /var/log/auth.log | grep changed
Jun 22 15:45:44 kali-linux-vm chage[15352]: changed password expiry for tcpdump
Jun 22 16:04:33 kali-linux-vm chage[17909]: changed password expiry for tss
Jun 22 16:04:33 kali-linux-vm chfn[17913]: changed user 'tss' information
2023-09-29T10:25:13.074504-04:00 kali-linux-vm groupmod[2106260]: group changed in /etc/group
(group ssh/127, new name: _ssh)
2023-09-29T10:25:13.077298-04:00 kali-linux-vm groupmod[2106260]: group changed in /etc/gshado
w (group ssh, new name: _ssh)
2023-09-29T10:29:06.004835-04:00 kali-linux-vm chfn[2188633]: changed user 'fwupd-refresh' inf
ormation
2023-11-13T15:25:06.987153-05:00 kali-linux-vm passwd[2040900]: pam_unix(passwd:chauthtok): pa
ssword changed for yoda
2023-11-13T15:25:38.701270-05:00 kali-linux-vm passwd[2040926]: pam_unix(passwd:chauthtok): pa
ssword changed for Chewbacca
2023-11-13T15:32:46.827134-05:00 kali-linux-vm passwd[2041031]: pam_unix(passwd:chauthtok): pa
ssword changed for yoda
2023-11-13T15:33:11.826081-05:00 kali-linux-vm passwd[2041059]: pam_unix(passwd:chauthtok): pa
ssword changed for Chewbacca
root@kali-linux-vm:~# 
```

Ans: John the Ripper: John the Ripper is a popular open-source password cracking application that uses a variety of attack techniques to find weak passwords. Originally created for Unix-based systems, it now supports a variety of platforms. John the Ripper is a flexible tool for cracking password hashes collected from a variety of sources since it uses tactics including dictionary assaults, brute-force attacks, and hybrid attacks. It can effectively handle password files from many operating systems and supports a variety of cryptographic hash techniques. Because of John the Ripper's speed and versatility as a command-line tool, security experts and penetration testers frequently use it to evaluate the security of passwords in a variety of settings. Its architecture is expandable and modular, and the community continues to maintain it, which adds to its efficacy in cases including penetration testing and password auditing.

Cryptii: A flexible online tool, Cryptii allows you to experiment with different ciphers and cryptographic algorithms for text encoding, decoding, and transformation. Cryptii's user-friendly interface lets users experiment with a variety of encoding techniques, from more sophisticated algorithms like RSA and AES to more traditional ciphers like Caesar and Atbash. Users may observe the instantaneous outcomes of their encoding or decoding decisions thanks to the platform's real-time feedback feature. In addition to being a useful tool for teaching and helping students grasp cryptographic ideas, Cryptii is an entertaining and engaging platform where aficionados may delve into the intriguing realm of codes and ciphers. Regardless of expertise level, anybody interested in cryptography can for studying and experimenting with cryptography, Cryptii provides an approachable and entertaining environment.



I love cyber security

SHIFT: 7 a→h

ALPHABET: abcdefghijklmnopqrstuvwxyz

CASE STRATEGY: Maintain case

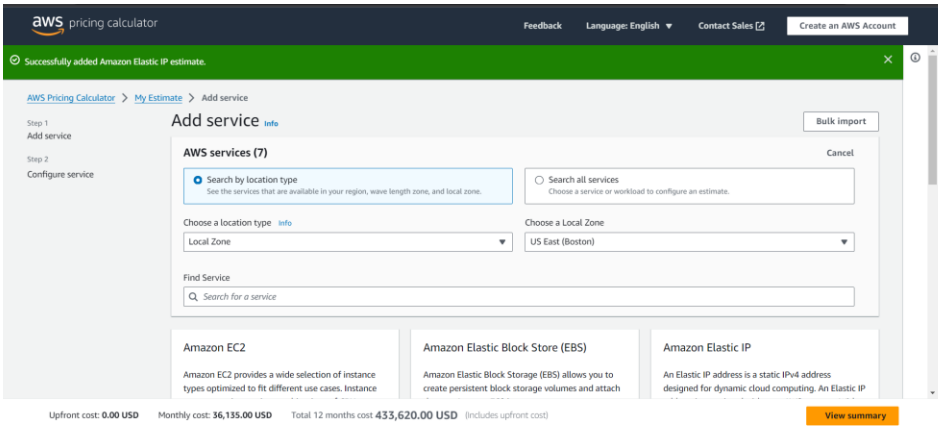
FOREIGN CHARS: Include Ignore

→ Encoded 21 chars

P svcl jfily zljbypaf

b. Describe at least 2 tools related to Cloud computing and Internet of things (chapter 13 with relevant screenshots, that demonstrates its working.)

Ans: AWS(Amazon web serives): Navigating the AWS Management Console and choosing the required service from the vast catalog of offers is the process of adding a service to Amazon Web Services (AWS). AWS offers a vast array of cloud services, such as databases, machine learning, storage, processing power, and more. Users can select the exact service they wish to add by navigating to the "Services" dropdown menu in the AWS Management Console after logging in. To include a relational database, for example, they may use Amazon RDS; alternatively, they could choose Amazon EC2 for flexible and scalable computing resources. AWS streamlines the procedure by offering user-friendly interfaces and comprehensive wizards for setting and implementing every service.



Successfully added Amazon Elastic IP estimate.

Add service Info

AWS services (7)

● Search by location type
See the services that are available in your region, wave length zone, and local zone.

○ Search all services
Choose a service or workload to configure an estimate.

Choose a location type Info: Local Zone

Choose a Local Zone: US East (Boston)

Find Service: Search for a service

Amazon EC2
Amazon EC2 provides a wide selection of instance types optimized to fit different use cases. Instance

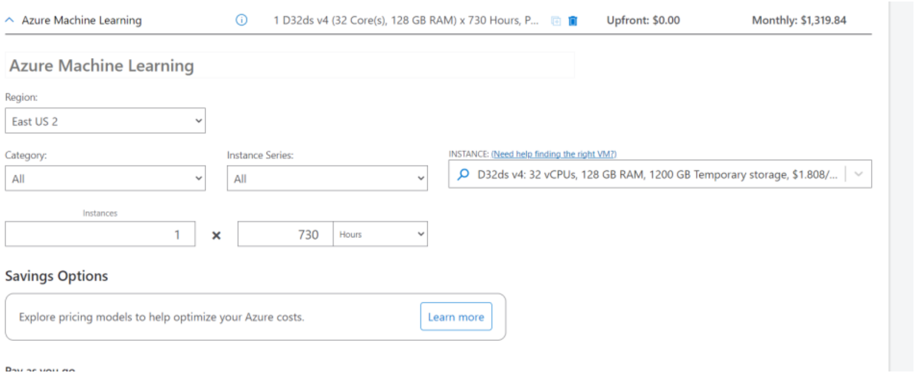
Amazon Elastic Block Store (EBS)
Amazon Elastic Block Storage (EBS) allows you to create persistent block storage volumes and attach

Amazon Elastic IP
An Elastic IP address is a static IPv4 address designed for dynamic cloud computing. An Elastic IP

Upfront cost: 0.00 USD Monthly cost: 36,135.00 USD Total 12 months cost 433,620.00 USD (Includes upfront cost)

Micorsoft Azure:

Building, deploying, and maintaining apps for organizations is made easier with Microsoft Azure, a complete cloud computing platform that provides a wide range of services via Microsoft's extensive worldwide network of data centers. Azure offers solutions in the following areas: machine learning, networking, databases, analytics, storage, and computing. Customers may take use of a variety of development tools and frameworks, pay for just the resources they use, and scale resources up or down in response to demand by utilizing Azure's infrastructure. Azure enables smooth interaction between on-premises data centers and the cloud, with a focus on hybrid cloud features. It is a flexible option that works with many operating systems, frameworks, and programming languages, making it appropriate for businesses of all kinds. A reliable cloud solution for a variety of computing purposes, Azure is well-liked due to its rich documentation, strong security measures, and user-friendly design.



Azure Machine Learning | 1 D32ds v4 (32 Core(s), 128 GB RAM) x 730 Hours, P... | Upfront: $0.00 | Monthly: $1,319.84

Azure Machine Learning

Region: East US 2

Category: All

Instance Series: All

INSTANCE: (Need help finding the right VM?) D32ds v4: 32 vCPUs, 128 GB RAM, 1200 GB Temporary storage, $1.808/...

Instances: 1 × 730 Hours

Savings Options

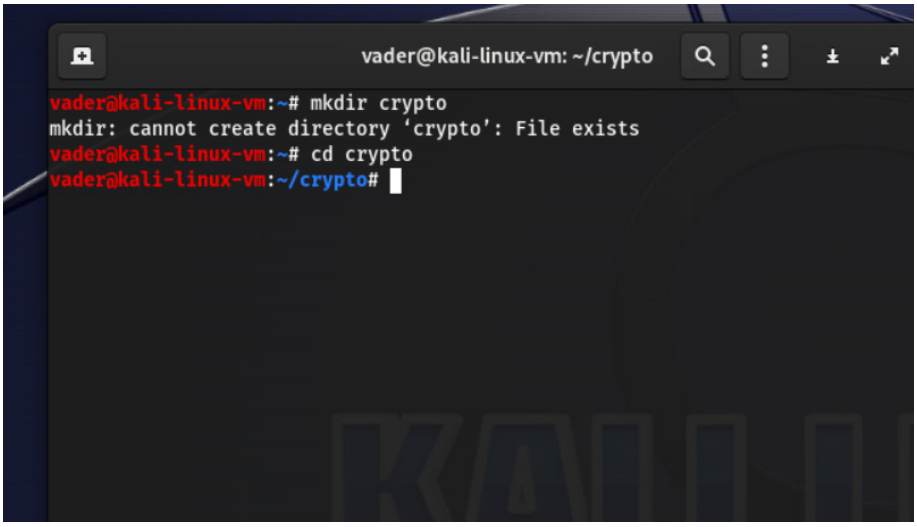
Explore pricing models to help optimize your Azure costs.

LAB 8: Cryptograhy, Encoding, and Numbering Systems: Labs

Part Zero: Hashing

Open a terminal window on the Linux SIFT workstation. Create a folder and navigate to the newly created folder:

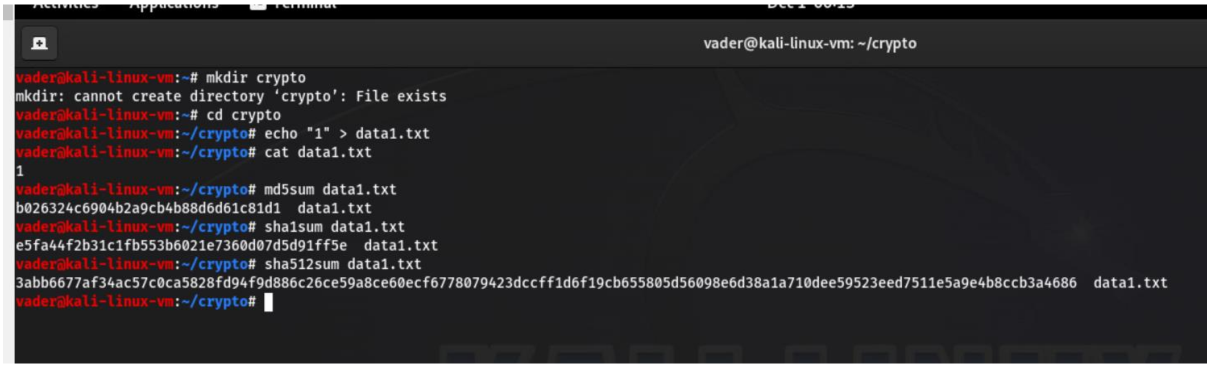
- mkdir crypto
- cd crypto



```
vader@kali-linux-vm:~# mkdir crypto
mkdir: cannot create directory ‘crypto’: File exists
vader@kali-linux-vm:~# cd crypto
vader@kali-linux-vm:~/crypto#
```

Create a file, calculate hashes using different algorithms. Compare the output hash values:

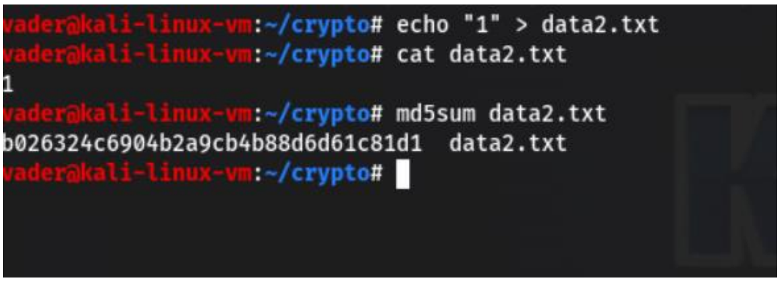
- echo “1” > data1.txt
- cat data1.txt
- md5sum data1.txt
- sha1sum data1.txt
- sha512sum data1.txt



```
vader@kali-linux-vm:~# mkdir crypto
mkdir: cannot create directory ‘crypto’: File exists
vader@kali-linux-vm:~# cd crypto
vader@kali-linux-vm:~/crypto# echo "1" > data1.txt
vader@kali-linux-vm:~/crypto# cat data1.txt
1
vader@kali-linux-vm:~/crypto# md5sum data1.txt
b026324c6904b2a9cb4b88d6d61c81d1  data1.txt
vader@kali-linux-vm:~/crypto# sha1sum data1.txt
e5fa44f2b31c1fb553b6021e7360d07d5d91ff5e  data1.txt
vader@kali-linux-vm:~/crypto# sha512sum data1.txt
3abb6677af34ac57c0ca5828fd94f9d886c26ce59a8ce60ecf6778079423dccff1d6f19cb655805d56098e6d38a1a710dee59523eed7511e5a9e4b8ccb3a4686  data1.txt
vader@kali-linux-vm:~/crypto# 
```

Create another file with the same content. Note that the filename does not change the hash value, but the file content does change:

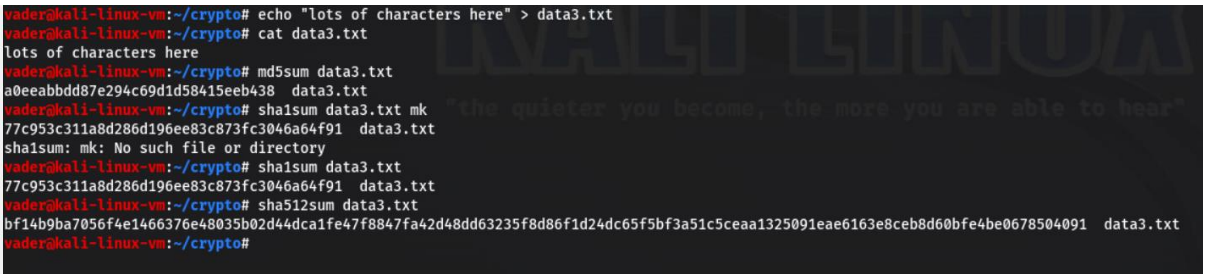
- ➢ echo “1” > data2.txt
- ➢ cat data2.txt
- ➢ md5sum data2.txt



```
vader@kali-linux-vm:~/crypto# echo "1" > data2.txt
vader@kali-linux-vm:~/crypto# cat data2.txt
1
vader@kali-linux-vm:~/crypto# md5sum data2.txt
b026324c6904b2a9cb4b88d6d61c81d1  data2.txt
vader@kali-linux-vm:~/crypto# 
```

Create a third file with different content. Note that you can copy and paste thousands of words in the third file. Calculate hash values, and note that hash length does not change depending on the input length, but the hash values change.

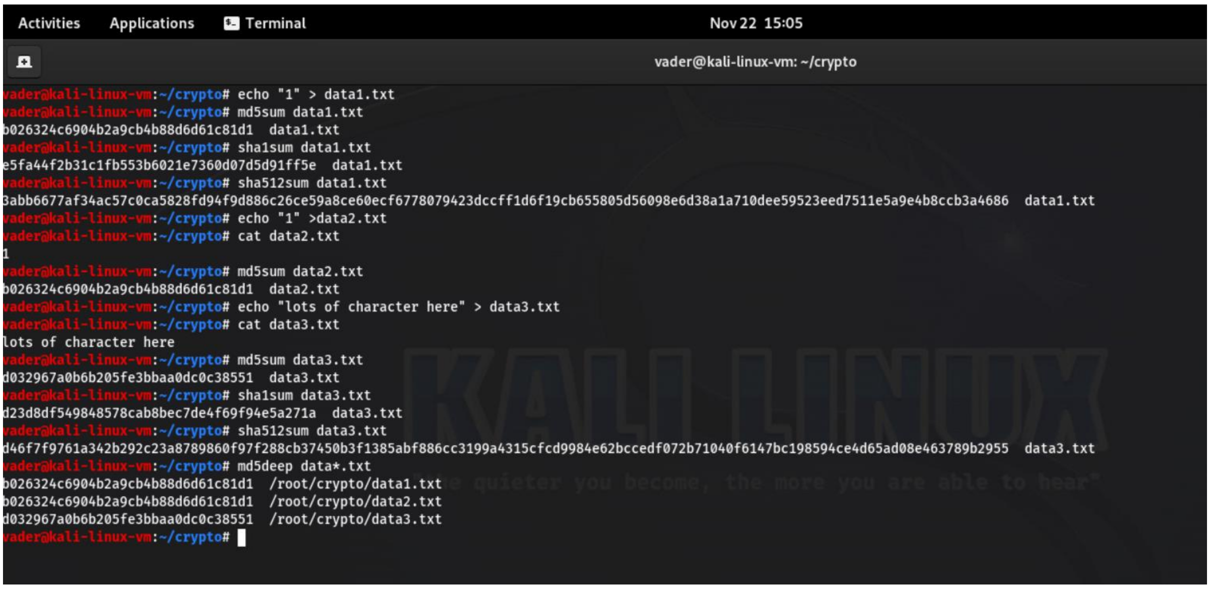
- ➢ Echo “lots of characters here” > data3.txt
- ➢ Cat data3.txt
- ➢ Md5sum data3.txt
- ➢ Sha1sum data3.txt mk
- ➢ Sha512sum data3.txt



```
vader@kali-linux-vm:~/crypto# echo "lots of characters here" > data3.txt
vader@kali-linux-vm:~/crypto# cat data3.txt
lots of characters here
vader@kali-linux-vm:~/crypto# md5sum data3.txt
a0eeabbdd87e294c69d1d58415eeb438  data3.txt
vader@kali-linux-vm:~/crypto# sha1sum data3.txt mk
77c953c311a8d286d196ee83c873fc3046a64f91  data3.txt
sha1sum: mk: No such file or directory
vader@kali-linux-vm:~/crypto# sha1sum data3.txt
77c953c311a8d286d196ee83c873fc3046a64f91  data3.txt
vader@kali-linux-vm:~/crypto# sha512sum data3.txt
bf14b9ba7056f4e1466376e48035b02d44dca1fe47f8847fa42d48dd63235f8d86f1d24dc65f5bf3a51c5ceaa1325091eae6163e8ceb8d60bfe4be0678504091  data3.txt
vader@kali-linux-vm:~/crypto#
```

Compare the hash values of created files all at once using the md5deep tool. Note that you may need to calculate many files you collected from the suspected machine as a forensics investigator.

- ➢ Md5deep data*.txt

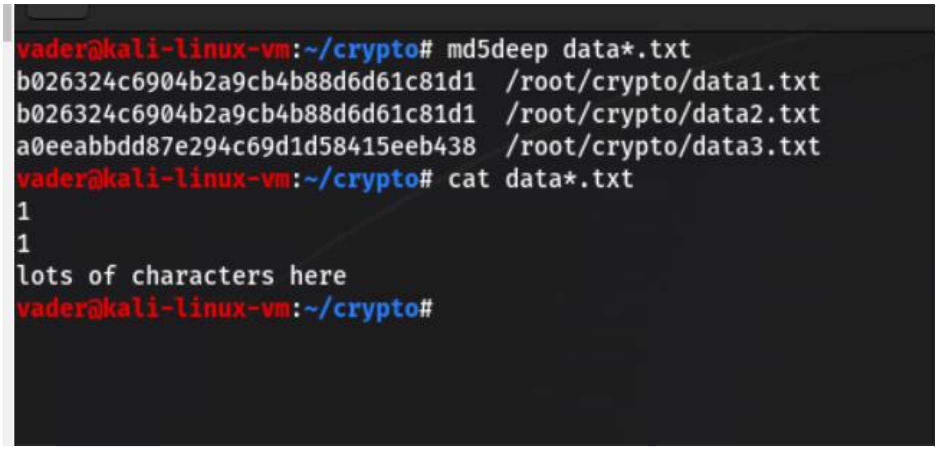


```
vader@kali-linux-vm:~/crypto# echo "1" > data1.txt
vader@kali-linux-vm:~/crypto# md5sum data1.txt
b026324c6904b2a9cb4b88d6d61c81d1  data1.txt
vader@kali-linux-vm:~/crypto# sha1sum data1.txt
e5fa44f2b31c1fb553b6021e7360d07d5d91ff5e  data1.txt
vader@kali-linux-vm:~/crypto# sha512sum data1.txt
3abb6677af34ac57c0ca5828fd94f9d886c26ce59a8ce60ecf6778079423dccff1d6f19cb655805d56098e6d38a1a710dee59523eed7511e5a9e4b8ccb3a4686  data1.txt
vader@kali-linux-vm:~/crypto# echo "1" >data2.txt
vader@kali-linux-vm:~/crypto# cat data2.txt
1
vader@kali-linux-vm:~/crypto# md5sum data2.txt
b026324c6904b2a9cb4b88d6d61c81d1  data2.txt
vader@kali-linux-vm:~/crypto# echo "lots of character here" > data3.txt
vader@kali-linux-vm:~/crypto# cat data3.txt
lots of character here
vader@kali-linux-vm:~/crypto# md5sum data3.txt
d032967a0b6b205fe3bbaa0dc0c38551  data3.txt
vader@kali-linux-vm:~/crypto# sha1sum data3.txt
d23d8df549848578cab8bec7de4f69f94e5a271a  data3.txt
vader@kali-linux-vm:~/crypto# sha512sum data3.txt
d46f7f9761a342b292c23a8789860f97f288cb37450b3f1385abf886cc3199a4315cfcd9984e62bccedf072b71040f6147bc198594ce4d65ad08e463789b2955  data3.txt
vader@kali-linux-vm:~/crypto# md5deep data*.txt
b026324c6904b2a9cb4b88d6d61c81d1  /root/crypto/data1.txt
b026324c6904b2a9cb4b88d6d61c81d1  /root/crypto/data2.txt
d032967a0b6b205fe3bbaa0dc0c38551  /root/crypto/data3.txt
vader@kali-linux-vm:~/crypto# 
```

Module Assessment

Question 1: Will the hash length change depending upon the characters of text to be encrypted? If no, prove it with the help of screenshots.

Ans: No the length of hash is independent on the characters of the text to be encrypted.



```
vader@kali-linux-vm:~/crypto# md5deep data*.txt
b026324c6904b2a9cb4b88d6d61c81d1  /root/crypto/data1.txt
b026324c6904b2a9cb4b88d6d61c81d1  /root/crypto/data2.txt
a0eeabbdd87e294c69d1d58415eeb438  /root/crypto/data3.txt
vader@kali-linux-vm:~/crypto# cat data*.txt
1
1
lots of characters here
vader@kali-linux-vm:~/crypto#
```

As we can see in the screenshot, data3.txt file contains a couple of characters when compared with data1.txt and data2.txt

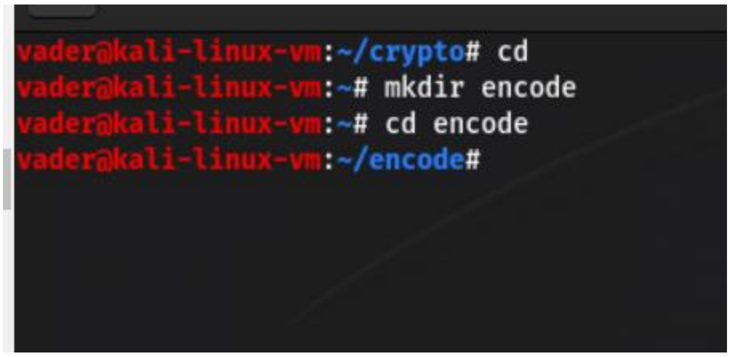
But the hashed value is of same bitsize in all the files.

Modile Activity Description:

Part one : Encoding & decoding

Open a terminal window on the Linux SIFT workstation. Create a folder and navigate to the newly created folder:

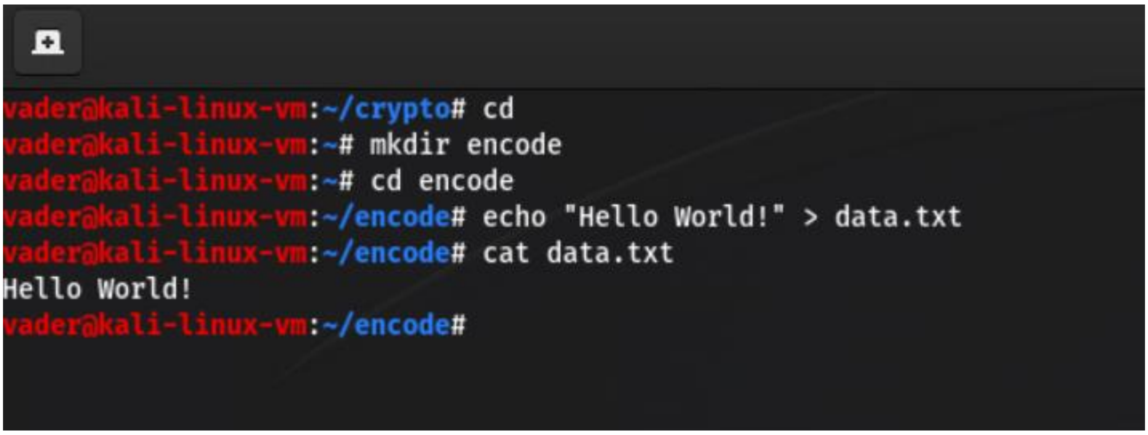
- Mkdir encode
- Cd encode



```
vader@kali-linux-vm:~/crypto# cd
vader@kali-linux-vm:~# mkdir encode
vader@kali-linux-vm:~# cd encode
vader@kali-linux-vm:~/encode#
```

Create a file and check the content:

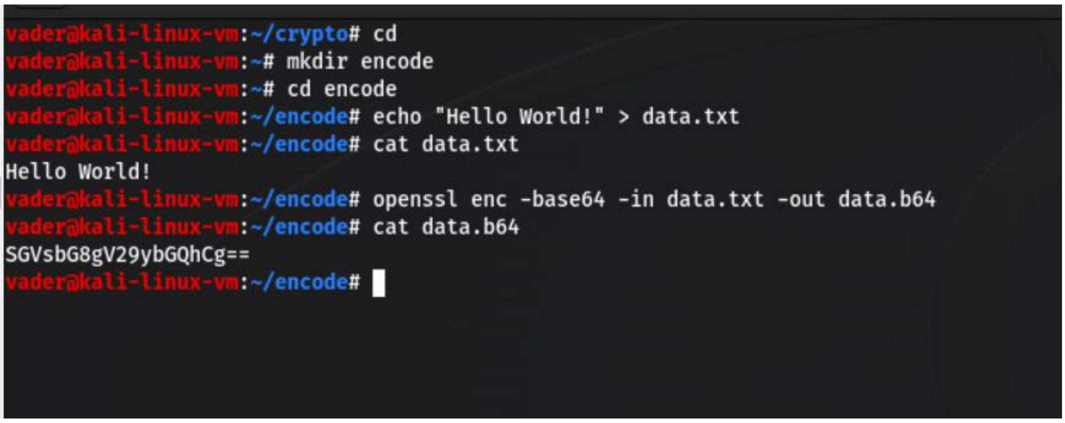
- Echo “Hello World!” > data.txt
- Cat data.txt



```
vader@kali-linux-vm:~/crypto# cd
vader@kali-linux-vm:~# mkdir encode
vader@kali-linux-vm:~# cd encode
vader@kali-linux-vm:~/encode# echo "Hello World!" > data.txt
vader@kali-linux-vm:~/encode# cat data.txt
Hello World!
vader@kali-linux-vm:~/encode#
```

Perform base64 encoding ofls the file using openssl utility and check the content:

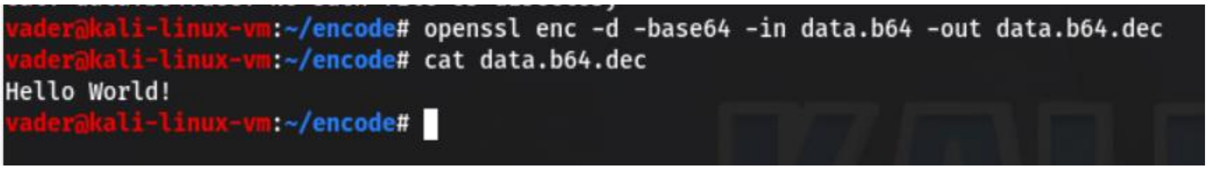
- Openssl enc –base64 –in data.txt –out data.b64
- Cat data.b64



```
vader@kali-linux-vm:~/crypto# cd
vader@kali-linux-vm:~# mkdir encode
vader@kali-linux-vm:~# cd encode
vader@kali-linux-vm:~/encode# echo "Hello World!" > data.txt
vader@kali-linux-vm:~/encode# cat data.txt
Hello World!
vader@kali-linux-vm:~/encode# openssl enc -base64 -in data.txt -out data.b64
vader@kali-linux-vm:~/encode# cat data.b64
SGVsbG8gV29ybGQhCg==
vader@kali-linux-vm:~/encode#
```

Decode the encoded file using openssl utility and check the content:

- Openssl enc –d –base64 –in data.b64 –out data.b64.dec
- Cat data.b64.dec

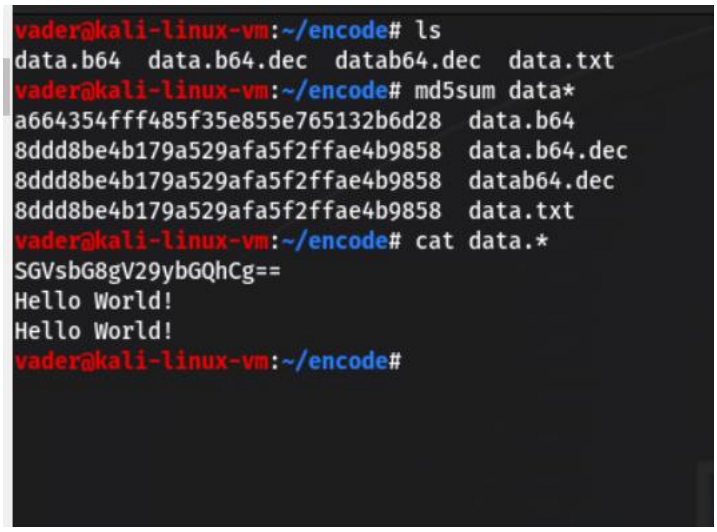


```
vader@kali-linux-vm:~/encode# openssl enc -d -base64 -in data.b64 -out data.b64.dec
vader@kali-linux-vm:~/encode# cat data.b64.dec
Hello World!
vader@kali-linux-vm:~/encode# █
```

Module Assessment

Question 2. List the files from encode folder and compare the md5 hashes. What is the conclusion from comparing md5sum with the directory.

Ans: As we can see from the above screenshot will come to know that the MD5sum comparison of all the three files results in the following conclusion. The md5sum hashed values of data.b64.dec (decoded file content) and data.txt (plain text) are one and the same. Since both the data is same. We can see the data content of both files results in Hello world!



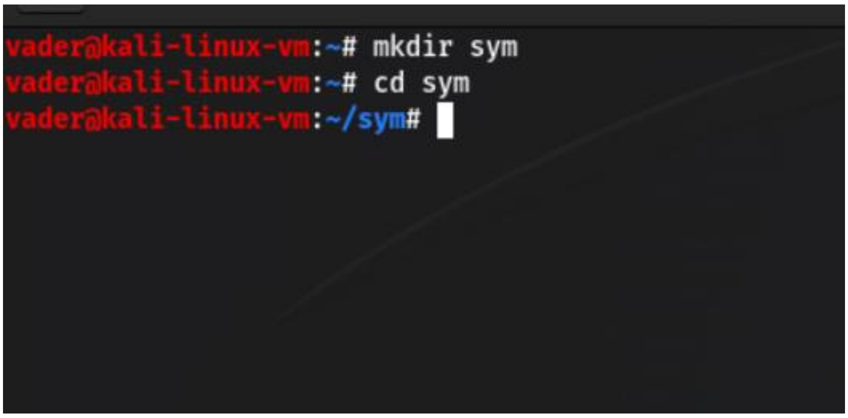
```
vader@kali-linux-vm:~/encode# ls
data.b64  data.b64.dec  datab64.dec  data.txt
vader@kali-linux-vm:~/encode# md5sum data*
a664354fff485f35e855e765132b6d28  data.b64
8ddd8be4b179a529afa5f2ffae4b9858  data.b64.dec
8ddd8be4b179a529afa5f2ffae4b9858  datab64.dec
8ddd8be4b179a529afa5f2ffae4b9858  data.txt
vader@kali-linux-vm:~/encode# cat data.*
SGVsbG8gV29ybGQhCg==
Hello World!
Hello World!
vader@kali-linux-vm:~/encode#
```

Module Activity Description:

Part Two: Symmetric Encryption

Open a terminal window on the Linux SIFT workstation. Create a folder and navigate to the newly created folder:

- Mkdir sym
- Cd sym



```
vader@kali-linux-vm:~# mkdir sym
vader@kali-linux-vm:~# cd sym
vader@kali-linux-vm:~/sym# █
```

Create a file and check the content:

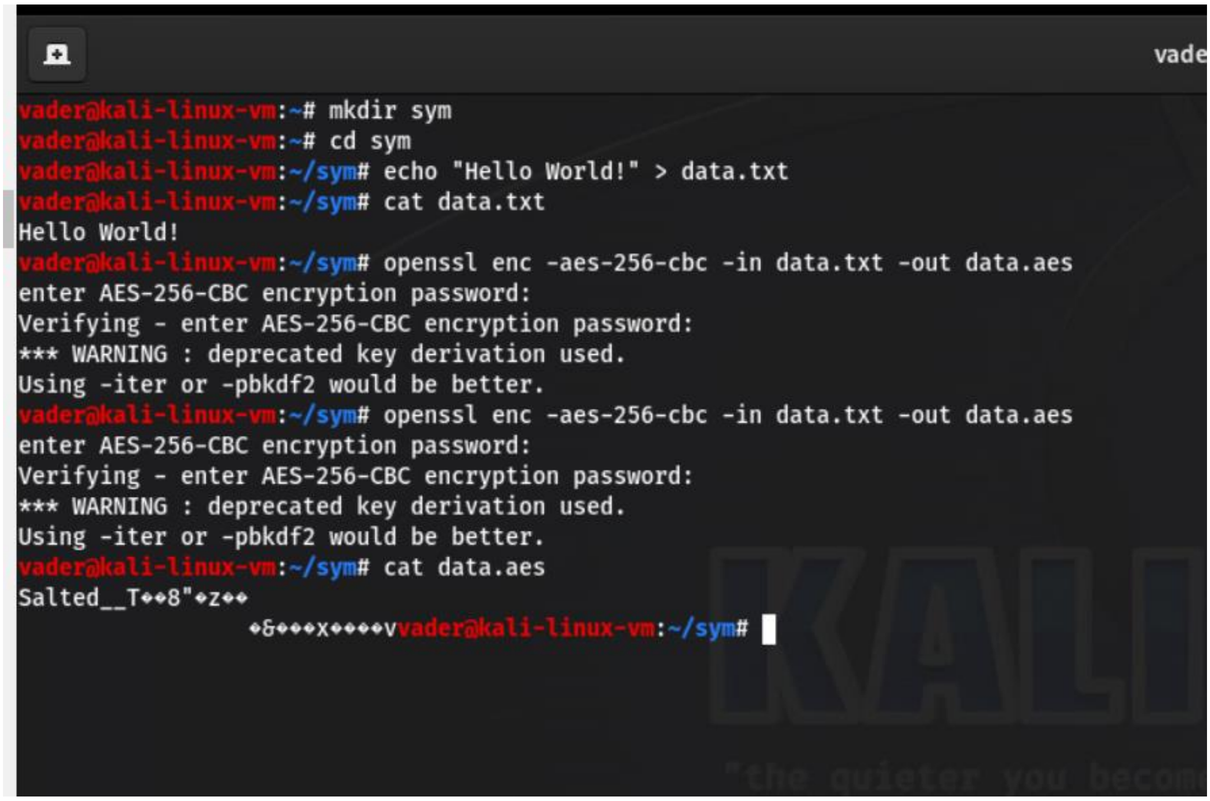
- Echo “Hello World!” > data.txt
- Cat data.txt

Encrypt(AES encryption) the file using openssl utility and check the content:

- Openssl enc –aes-256-cbc –in data.txt –out data.aes

Openssl will ask a password to protect the key. Type a password

- Cat data.aes

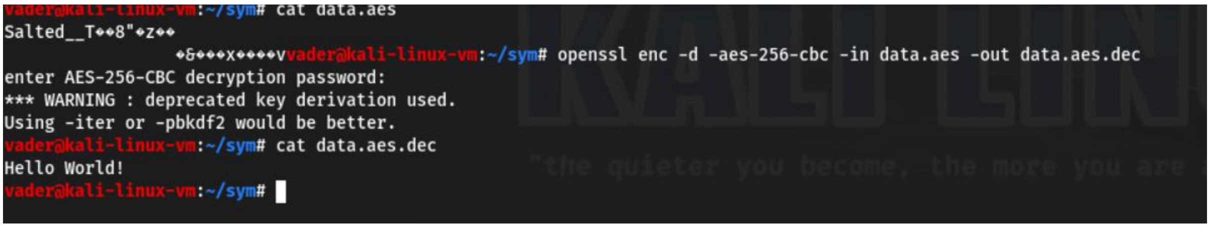


```
vader@kali-linux-vm:~# mkdir sym
vader@kali-linux-vm:~# cd sym
vader@kali-linux-vm:~/sym# echo "Hello World!" > data.txt
vader@kali-linux-vm:~/sym# cat data.txt
Hello World!
vader@kali-linux-vm:~/sym# openssl enc -aes-256-cbc -in data.txt -out data.aes
enter AES-256-CBC encryption password:
Verifying - enter AES-256-CBC encryption password:
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
vader@kali-linux-vm:~/sym# openssl enc -aes-256-cbc -in data.txt -out data.aes
enter AES-256-CBC encryption password:
Verifying - enter AES-256-CBC encryption password:
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
vader@kali-linux-vm:~/sym# cat data.aes
Salted__T◆◆8"◆z◆◆
                 ◆&◆◆◆x◆◆◆◆vvader@kali-linux-vm:~/sym# 
```

In the above screenshot, we can see that we are having a warning, deprecated key derivation used. Here I got this warning because I used the encryption password as “hello” which is the content of the file.

Decrypt the encrypted file using openssl utility and check the content:

- openssl enc -d -aes-256-cbc -in data.aes -out data.aes.dec
- cat data.aes.dec

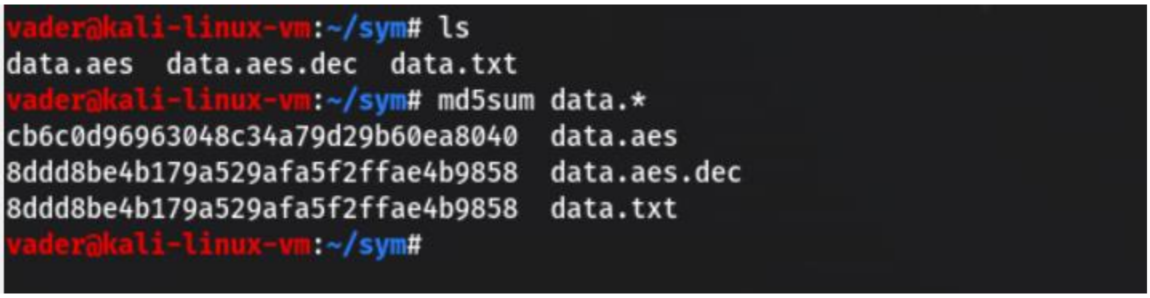


```
vader@kali-linux-vm:~/sym# cat data.aes
Salted__T◆◆8"◆z◆◆
                 ◆&◆◆◆x◆◆◆◆vvader@kali-linux-vm:~/sym# openssl enc -d -aes-256-cbc -in data.aes -out data.aes.dec
enter AES-256-CBC decryption password:
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
vader@kali-linux-vm:~/sym# cat data.aes.dec
Hello World!
vader@kali-linux-vm:~/sym# 
```

Module Assessment

Question 3 - List the files, see the filetypes, and compare the md5 hashes. What utility was used for AES encryption and explain it?

Ans:



```
vader@kali-linux-vm:~/sym# ls
data.aes  data.aes.dec  data.txt
vader@kali-linux-vm:~/sym# md5sum data.*
cb6c0d96963048c34a79d29b60ea8040  data.aes
8ddd8be4b179a529afa5f2ffae4b9858  data.aes.dec
8ddd8be4b179a529afa5f2ffae4b9858  data.txt
vader@kali-linux-vm:~/sym#
```

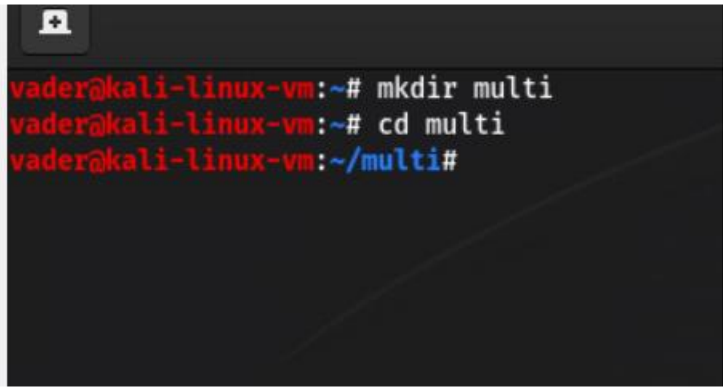
When we compare the MD5 hashes for the following files, we can see that we have three files of the types.aes,.aes.dec, and.txt. This is the output of the program that was used in the screenshot above. The tool used for the AES is SSL, as we have utilized openssl aes-256-cbc. this module's encryption.

Module Activity Description:

Part Three: Multiple Operations

Open a terminal window on the Linux SIFT workstation. Create a folder and navigate to the newly created folder:

- mkdir multi
- cd multi



```
vader@kali-linux-vm:~# mkdir multi
vader@kali-linux-vm:~# cd multi
vader@kali-linux-vm:~/multi#
```

Create a file and check the content:

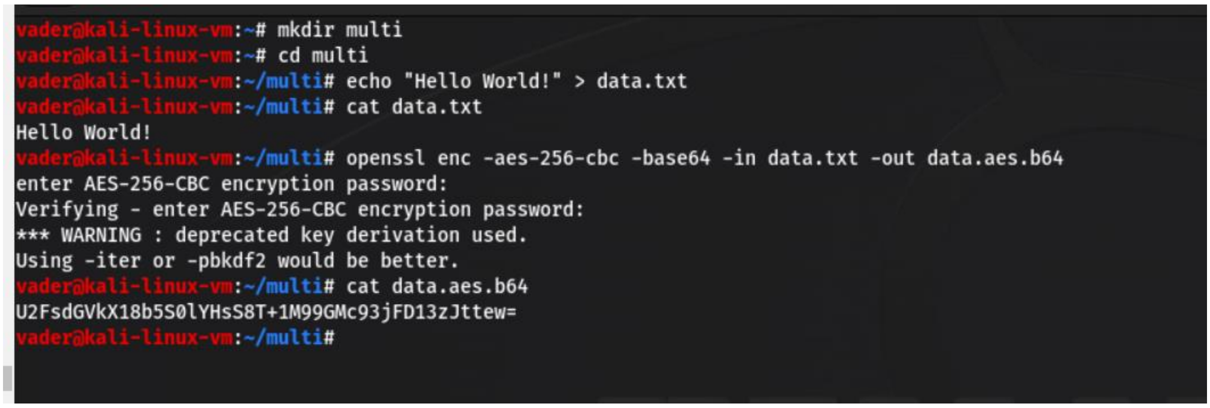
- echo “Hello world!” > data.txt
- cat data.txt

First encrypt the file (AES encryption) and then encode it (Base64) using openssl and check the content:

- openssl enc –aes-256-cbc –base64 –in data.txt –out data.aes.b64

openssl will ask for a password to protect the key. Type a password

- cat data.aes.b64

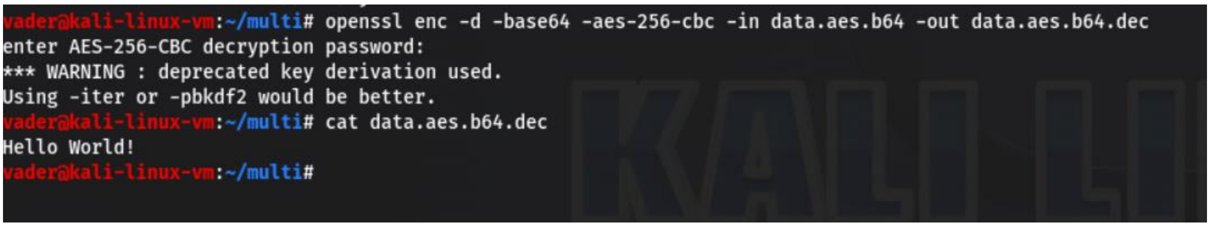


```
vader@kali-linux-vm:~# mkdir multi
vader@kali-linux-vm:~# cd multi
vader@kali-linux-vm:~/multi# echo "Hello World!" > data.txt
vader@kali-linux-vm:~/multi# cat data.txt
Hello World!
vader@kali-linux-vm:~/multi# openssl enc -aes-256-cbc -base64 -in data.txt -out data.aes.b64
enter AES-256-CBC encryption password:
Verifying - enter AES-256-CBC encryption password:
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
vader@kali-linux-vm:~/multi# cat data.aes.b64
U2FsdGVkX18b5S0lYHsS8T+1M99GMc93jFD13zJttew=
vader@kali-linux-vm:~/multi#
```

First deocde the encoded file and then decrypt the encrypted file

Using openssl and check the content:

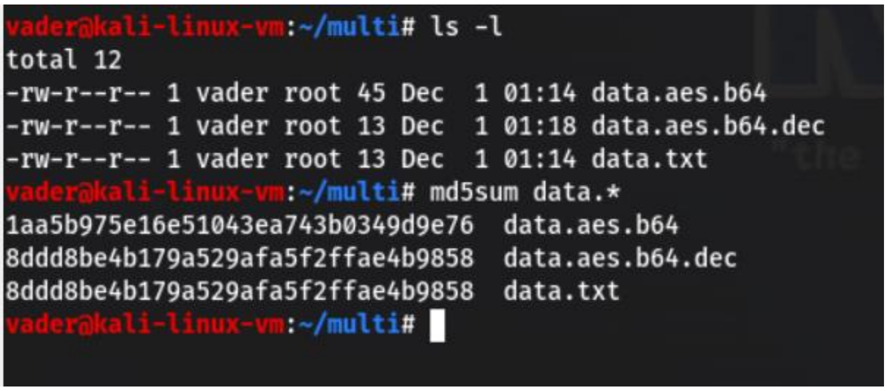
- openssl enc –d –base64 –aes-256-cbc –in data.aes.b64 –out data.aes.b64.dec
- cat data.aes.b64.dec



```
vader@kali-linux-vm:~/multi# openssl enc -d -base64 -aes-256-cbc -in data.aes.b64 -out data.aes.b64.dec
enter AES-256-CBC decryption password:
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
vader@kali-linux-vm:~/multi# cat data.aes.b64.dec
Hello World!
vader@kali-linux-vm:~/multi#
```

List the files and compare the md5 hashes:

- ls –l
- md5sum data.*



```
vader@kali-linux-vm:~/multi# ls -l
total 12
-rw-r--r-- 1 vader root 45 Dec  1 01:14 data.aes.b64
-rw-r--r-- 1 vader root 13 Dec  1 01:18 data.aes.b64.dec
-rw-r--r-- 1 vader root 13 Dec  1 01:14 data.txt
vader@kali-linux-vm:~/multi# md5sum data.*
1aa5b975e16e51043ea743b0349d9e76  data.aes.b64
8ddd8be4b179a529afa5f2ffae4b9858  data.aes.b64.dec
8ddd8be4b179a529afa5f2ffae4b9858  data.txt
vader@kali-linux-vm:~/multi# 
```

Module Assessment

Question 4 - What encoding technique was used on encryption? Compare the screenshot of encrypted file from aes encryption (part two – data.aes) and encrypted file from multiple operation (part three – data.aes.b64)? What is the difference between two files.

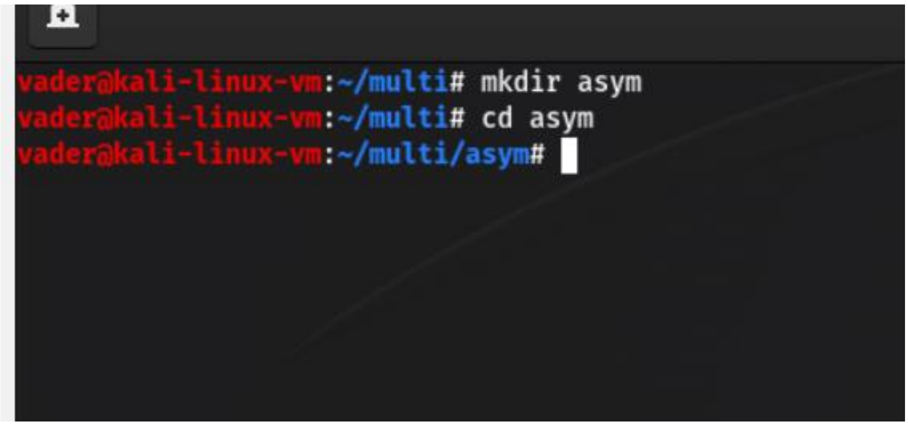
Ans: With openSSL, the contents in a text file is encrypted using the AES encryption algorithm. Screenshots of encrypted files from various processes and encrypted data from encryption are shown below.

Module Activity Description:

Part Four: Asymmetric Encryption

Open a terminal window on the Linux SIFT workstation. Create a folder and navigate to the newly created folder:

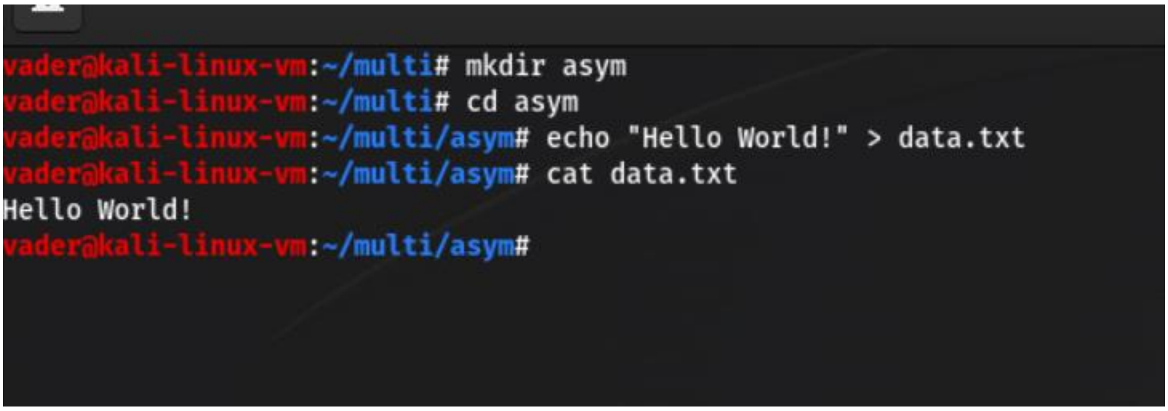
- mkdir asym
- cd asym



```
vader@kali-linux-vm:~/multi# mkdir asym
vader@kali-linux-vm:~/multi# cd asym
vader@kali-linux-vm:~/multi/asym# 
```

Create a file and check the content:

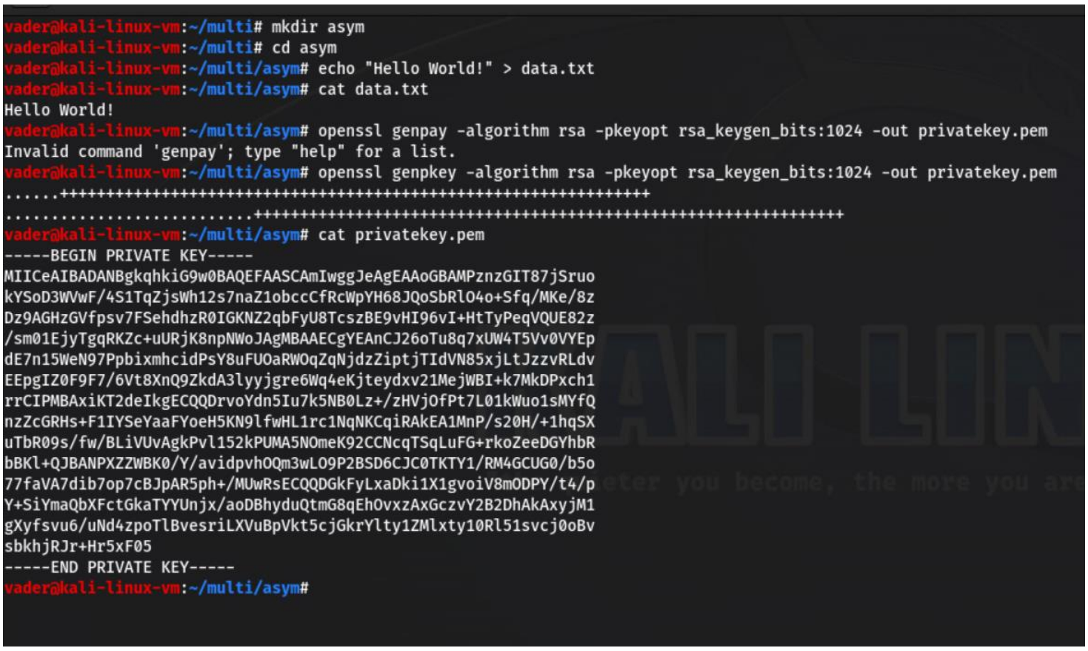
- echo “Hello World!” > data.txt
- cat data.txt



```
vader@kali-linux-vm:~/multi# mkdir asym
vader@kali-linux-vm:~/multi# cd asym
vader@kali-linux-vm:~/multi/asym# echo "Hello World!" > data.txt
vader@kali-linux-vm:~/multi/asym# cat data.txt
Hello World!
vader@kali-linux-vm:~/multi/asym#
```

Generate a private key using openssl and check the content of the key:

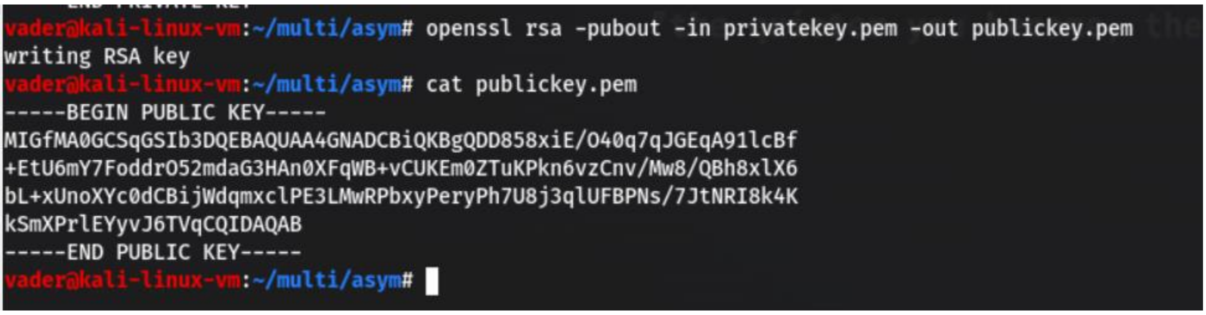
- openssl genpkey –algorithm rsa –pkeyopt rsa_keygen_bits:1024 –out privatekey.pem
- cat privatekey.pem



```
vader@kali-linux-vm:~/multi# mkdir asym
vader@kali-linux-vm:~/multi# cd asym
vader@kali-linux-vm:~/multi/asym# echo "Hello World!" > data.txt
vader@kali-linux-vm:~/multi/asym# cat data.txt
Hello World!
vader@kali-linux-vm:~/multi/asym# openssl genpay -algorithm rsa -pkeyopt rsa_keygen_bits:1024 -out privatekey.pem
Invalid command 'genpay'; type "help" for a list.
vader@kali-linux-vm:~/multi/asym# openssl genpkey -algorithm rsa -pkeyopt rsa_keygen_bits:1024 -out privatekey.pem
......++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
...........................++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
vader@kali-linux-vm:~/multi/asym# cat privatekey.pem
-----BEGIN PRIVATE KEY-----
MIICeAIBADANBgkqhkiG9w0BAQEFAASCAmIwggJeAgEAAoGBAMPznzGIT87jSruo
kYSoD3WVwF/4S1TqZjsWh12s7naZ1obccCfRcWpYH68JQoSbRlO4o+Sfq/MKe/8z
Dz9AGHzGVfpsv7FSehdhzR0IGKNZ2qbFyU8TcszBE9vHI96vI+HtTyPeqVQUE82z
/sm01EjyTgqRKZc+uURjK8npNWoJAgMBAAECgYEAnCJ26oTu8q7xUW4T5Vv0VYEp
dE7n15WeN97PpbixmhcidPsY8uFUOaRWOqZqNjdzZiptjTIdVN85xjLtJzzvRLdv
EEpgIZ0F9F7/6Vt8XnQ9ZkdA3lyyjgre6Wq4eKjteydxv21MejWBI+k7MkDPxch1
rrCIPMBAxiKT2deIkgECQQDrvoYdn5Iu7k5NB0Lz+/zHVjOfPt7L01kWuo1sMYfQ
nzZcGRHs+F1IYSeYaaFYoeH5KN9lfwHL1rc1NqNKCqiRAkEA1MnP/s20H/+1hqSX
uTbR09s/fw/BLiVUvAgkPvl152kPUMA5NOmeK92CCNcqTSqLuFG+rkoZeeDGYhbR
bBKl+QJBANPXZZWBK0/Y/avidpvhOQm3wLO9P2BSD6CJC0TKTY1/RM4GCUG0/b5o
77faVA7dib7op7cBJpAR5ph+/MUwRsECQQDGkFyLxaDki1X1gvoiV8mODPY/t4/p
Y+SiYmaQbXFctGkaTYYUnjx/aoDBhyduQtmG8qEhOvxzAxGczvY2B2DhAkAxyjM1
gXyfsvu6/uNd4zpoTlBvesriLXVuBpVkt5cjGkrYlty1ZMlxty10Rl51svcj0oBv
sbkhjRJr+Hr5xF05
-----END PRIVATE KEY-----
vader@kali-linux-vm:~/multi/asym#
```

Generate the public key associated with the private key and check the content of the key:

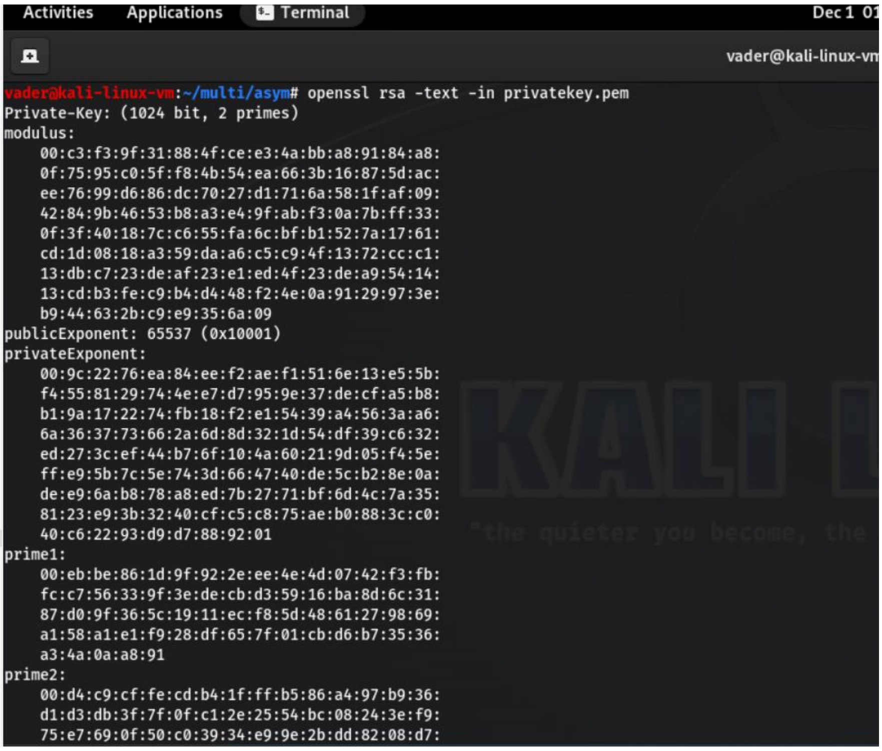
- openssl rsa –pubout –in privatekey.pem –out publickey.pem
- cat publickey.pem



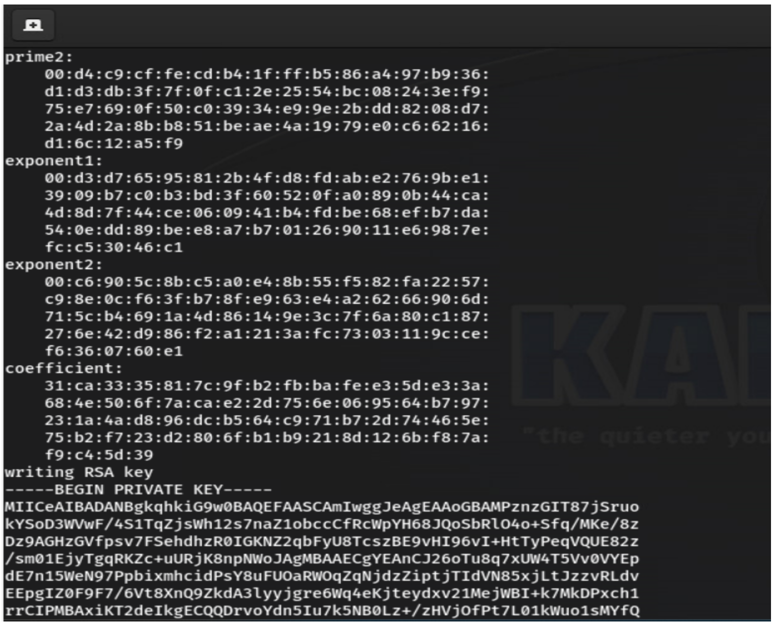
```
vader@kali-linux-vm:~/multi/asym# openssl rsa -pubout -in privatekey.pem -out publickey.pem
writing RSA key
vader@kali-linux-vm:~/multi/asym# cat publickey.pem
-----BEGIN PUBLIC KEY-----
MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQDD858xiE/O40q7qJGEqA91lcBf
+EtU6mY7FoddrO52mdaG3HAn0XFqWB+vCUKEm0ZTuKPkn6vzCnv/Mw8/QBh8xlX6
bL+xUnoXYc0dCBijWdqmxclPE3LMwRPbxyPeryPh7U8j3qlUFBPNs/7JtNRI8k4K
kSmXPrlEYyvJ6TVqCQIDAQAB
-----END PUBLIC KEY-----
vader@kali-linux-vm:~/multi/asym#
```

Preview the private key:

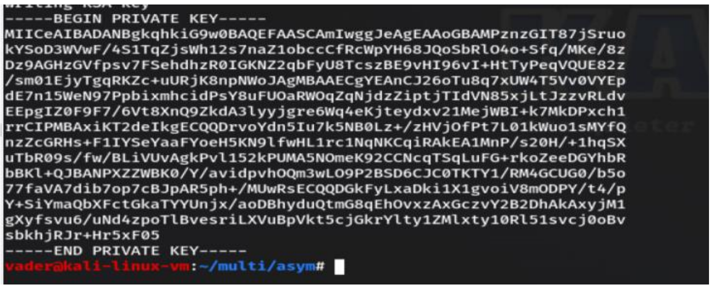
- openssl rsa –text –in privatekey.pem



```
vader@kali-linux-vm:~/multi/asym# openssl rsa -text -in privatekey.pem
Private-Key: (1024 bit, 2 primes)
modulus:
    00:c3:f3:9f:31:88:4f:ce:e3:4a:bb:a8:91:84:a8:
    0f:75:95:c0:5f:f8:4b:54:ea:66:3b:16:87:5d:ac:
    ee:76:99:d6:86:dc:70:27:d1:71:6a:58:1f:af:09:
    42:84:9b:46:53:b8:a3:e4:9f:ab:f3:0a:7b:ff:33:
    0f:3f:40:18:7c:c6:55:fa:6c:bf:b1:52:7a:17:61:
    cd:1d:08:18:a3:59:da:a6:c5:c9:4f:13:72:cc:c1:
    13:db:c7:23:de:af:23:e1:ed:4f:23:de:a9:54:14:
    13:cd:b3:fe:c9:b4:d4:48:f2:4e:0a:91:29:97:3e:
    b9:44:63:2b:c9:e9:35:6a:09
publicExponent: 65537 (0x10001)
privateExponent:
    00:9c:22:76:ea:84:ee:f2:ae:f1:51:6e:13:e5:5b:
    f4:55:81:29:74:4e:e7:d7:95:9e:37:de:cf:a5:b8:
    b1:9a:17:22:74:fb:18:f2:e1:54:39:a4:56:3a:a6:
    6a:36:37:73:66:2a:6d:8d:32:1d:54:df:39:c6:32:
    ed:27:3c:ef:44:b7:6f:10:4a:60:21:9d:05:f4:5e:
    ff:e9:5b:7c:5e:74:3d:66:47:40:de:5c:b2:8e:0a:
    de:e9:6a:b8:78:a8:ed:7b:27:71:bf:6d:4c:7a:35:
    81:23:e9:3b:32:40:cf:c5:c8:75:ae:b0:88:3c:c0:
    40:c6:22:93:d9:d7:88:92:01
prime1:
    00:eb:be:86:1d:9f:92:2e:ee:4e:4d:07:42:f3:fb:
    fc:c7:56:33:9f:3e:de:cb:d3:59:16:ba:8d:6c:31:
    87:d0:9f:36:5c:19:11:ec:f8:5d:48:61:27:98:69:
    a1:58:a1:e1:f9:28:df:65:7f:01:cb:d6:b7:35:36:
    a3:4a:0a:a8:91
prime2:
    00:d4:c9:cf:fe:cd:b4:1f:ff:b5:86:a4:97:b9:36:
    d1:d3:db:3f:7f:0f:c1:2e:25:54:bc:08:24:3e:f9:
    75:e7:69:0f:50:c0:39:34:e9:9e:2b:dd:82:08:d7:
```



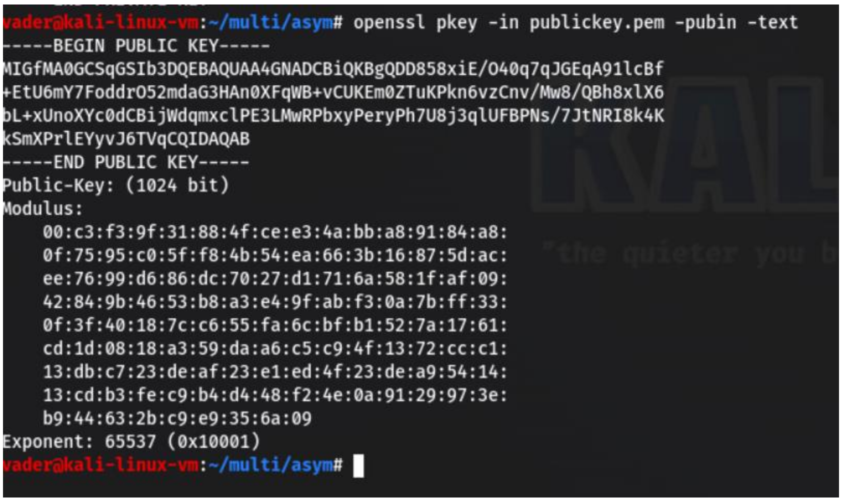
```
prime2:
    00:d4:c9:cf:fe:cd:b4:1f:ff:b5:86:a4:97:b9:36:
    d1:d3:db:3f:7f:0f:c1:2e:25:54:bc:08:24:3e:f9:
    75:e7:69:0f:50:c0:39:34:e9:9e:2b:dd:82:08:d7:
    2a:4d:2a:8b:b8:51:be:ae:4a:19:79:e0:c6:62:16:
    d1:6c:12:a5:f9
exponent1:
    00:d3:d7:65:95:81:2b:4f:d8:fd:ab:e2:76:9b:e1:
    39:09:b7:c0:b3:bd:3f:60:52:0f:a0:89:0b:44:ca:
    4d:8d:7f:44:ce:06:09:41:b4:fd:be:68:ef:b7:da:
    54:0e:dd:89:be:e8:a7:b7:01:26:90:11:e6:98:7e:
    fc:c5:30:46:c1
exponent2:
    00:c6:90:5c:8b:c5:a0:e4:8b:55:f5:82:fa:22:57:
    c9:8e:0c:f6:3f:b7:8f:e9:63:e4:a2:62:66:90:6d:
    71:5c:b4:69:1a:4d:86:14:9e:3c:7f:6a:80:c1:87:
    27:6e:42:d9:86:f2:a1:21:3a:fc:73:03:11:9c:ce:
    f6:36:07:60:e1
coefficient:
    31:ca:33:35:81:7c:9f:b2:fb:ba:fe:e3:5d:e3:3a:
    68:4e:50:6f:7a:ca:e2:2d:75:6e:06:95:64:b7:97:
    23:1a:4a:d8:96:dc:b5:64:c9:71:b7:2d:74:46:5e:
    75:b2:f7:23:d2:80:6f:b1:b9:21:8d:12:6b:f8:7a:
    f9:c4:5d:39
writing RSA key
-----BEGIN PRIVATE KEY-----
MIICeAIBADANBgkqhkiG9w0BAQEFAASCAmIwggJeAgEAAoGBAMPznzGIT87jSruo
kYSoD3WVwF/4S1TqZjsWh12s7naZ1obccCfRcWpYH68JQoSbRlO4o+Sfq/MKe/8z
Dz9AGHzGVfpsv7FSehdhzR0IGKNZ2qbFyU8TcszBE9vHI96vI+HtTyPeqVQUE82z
/sm01EjyTgqRKZc+uURjK8npNWoJAgMBAAECgYEAnCJ26oTu8q7xUW4T5Vv0VYEp
dE7n15WeN97PpbixmhcidPsY8uFUOaRWOqZqNjdzZiptjTIdVN85xjLtJzzvRLdv
EEpgIZ0F9F7/6Vt8XnQ9ZkdA3lyyjgre6Wq4eKjteydxv21MejWBI+k7MkDPxch1
rrCIPMBAxiKT2deIkgECQQDrvoYdn5Iu7k5NB0Lz+/zHVjOfPt7L01kWuo1sMYfQ
```



```
writing RSA key
-----BEGIN PRIVATE KEY-----
MIICeAIBADANBgkqhkiG9w0BAQEFAASCAmIwggJeAgEAAoGBAMPznzGIT87jSruo
kYSoD3WVwF/4S1TqZjsWh12s7naZ1obccCfRcWpYH68JQoSbRlO4o+Sfq/MKe/8z
Dz9AGHzGVfpsv7FSehdhzR0IGKNZ2qbFyU8TcszBE9vHI96vI+HtTyPeqVQUE82z
/sm01EjyTgqRKZc+uURjK8npNWoJAgMBAAECgYEAnCJ26oTu8q7xUW4T5Vv0VYEp
dE7n15WeN97PpbixmhcidPsY8uFUOaRWOqZqNjdzZiptjTIdVN85xjLtJzzvRLdv
EEpgIZ0F9F7/6Vt8XnQ9ZkdA3lyyjgre6Wq4eKjteydxv21MejWBI+k7MkDPxch1
rrCIPMBAxiKT2deIkgECQQDrvoYdn5Iu7k5NB0Lz+/zHVjOfPt7L01kWuo1sMYfQ
nzZcGRHs+F1IYSeYaaFYoeH5KN9lfwHL1rc1NqNKCqiRAkEA1MnP/s20H/+1hqSX
uTbR09s/fw/BLiVUvAgkPvl152kPUMA5NOmeK92CCNcqTSqLuFG+rkoZeeDGYhbR
bBKl+QJBANPXZZWBK0/Y/avidpvhOQm3wLO9P2BSD6CJC0TKTY1/RM4GCUG0/b5o
77faVA7dib7op7cBJpAR5ph+/MUwRsECQQDGkFyLxaDki1X1gvoiV8mODPY/t4/p
Y+SiYmaQbXFctGkaTYYUnjx/aoDBhyduQtmG8qEhOvxzAxGczvY2B2DhAkAxyjM1
gXyfsvu6/uNd4zpoTlBvesriLXVuBpVkt5cjGkrYlty1ZMlxty10Rl51svcj0oBv
sbkhjRJr+Hr5xF05
-----END PRIVATE KEY-----
vader@kali-linux-vm:~/multi/asym#
```

Preview the public key:

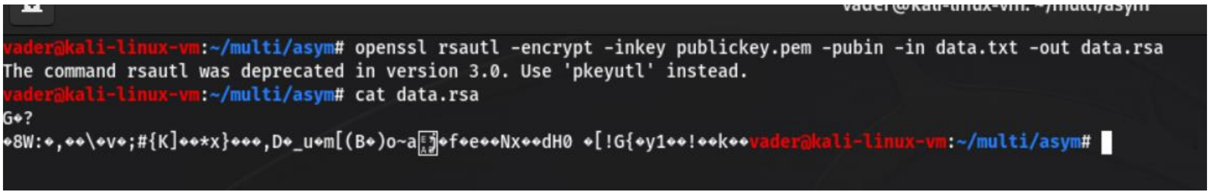
- openssl pkey –in publickey.pem –pubin –text



```
vader@kali-linux-vm:~/multi/asym# openssl pkey -in publickey.pem -pubin -text
-----BEGIN PUBLIC KEY-----
MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQDD858xiE/O40q7qJGEqA91lcBf
+EtU6mY7FoddrO52mdaG3HAn0XFqWB+vCUKEm0ZTuKPkn6vzCnv/Mw8/QBh8xlX6
bL+xUnoXYc0dCBijWdqmxclPE3LMwRPbxyPeryPh7U8j3qlUFBPNs/7JtNRI8k4K
kSmXPrlEYyvJ6TVqCQIDAQAB
-----END PUBLIC KEY-----
Public-Key: (1024 bit)
Modulus:
    00:c3:f3:9f:31:88:4f:ce:e3:4a:bb:a8:91:84:a8:
    0f:75:95:c0:5f:f8:4b:54:ea:66:3b:16:87:5d:ac:
    ee:76:99:d6:86:dc:70:27:d1:71:6a:58:1f:af:09:
    42:84:9b:46:53:b8:a3:e4:9f:ab:f3:0a:7b:ff:33:
    0f:3f:40:18:7c:c6:55:fa:6c:bf:b1:52:7a:17:61:
    cd:1d:08:18:a3:59:da:a6:c5:c9:4f:13:72:cc:c1:
    13:db:c7:23:de:af:23:e1:ed:4f:23:de:a9:54:14:
    13:cd:b3:fe:c9:b4:d4:48:f2:4e:0a:91:29:97:3e:
    b9:44:63:2b:c9:e9:35:6a:09
Exponent: 65537 (0x10001)
vader@kali-linux-vm:~/multi/asym#
```

Encrypt the file you created and check the content:

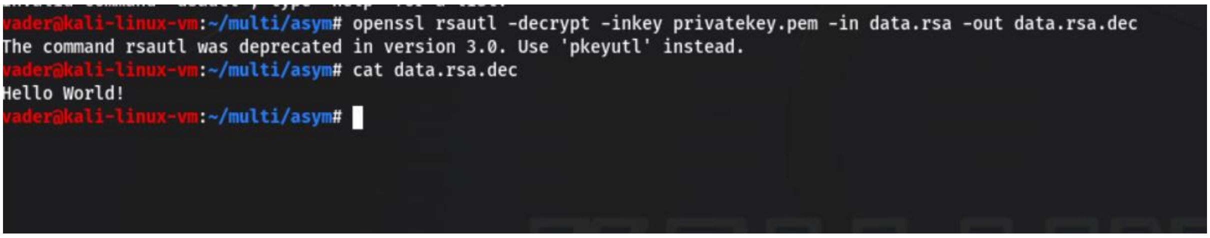
- ➢ openssl rsautl –encrypt –inkey publickey.pem –pubin –in data.txt –out data.rsa
- ➢ cata data.rsa



```
vader@kali-linux-vm:~/multi/asym# openssl rsautl -encrypt -inkey publickey.pem -pubin -in data.txt -out data.rsa
The command rsautl was deprecated in version 3.0. Use 'pkeyutl' instead.
vader@kali-linux-vm:~/multi/asym# cat data.rsa
G�?
�8W:�,��\�v�;#{K]���x}���,D�_u�m[(B�)o~a□�f�e��Nx��dH0 �[!G{�y1��!��k��vader@kali-linux-vm:~/multi/asym#
```

Decrypt the encrypted file and check the content:

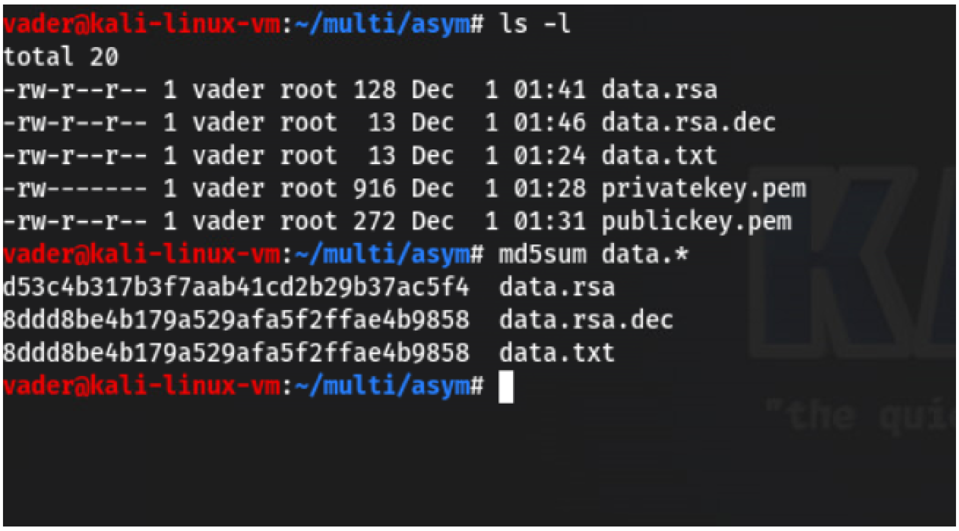
- ➢ openssl rsautl –decrypt –inkey privateket,pem –in data.rsa –out data.rsa.dec
- ➢ cat data.rsa.dec



```
vader@kali-linux-vm:~/multi/asym# openssl rsautl -decrypt -inkey privatekey.pem -in data.rsa -out data.rsa.dec
The command rsautl was deprecated in version 3.0. Use 'pkeyutl' instead.
vader@kali-linux-vm:~/multi/asym# cat data.rsa.dec
Hello World!
vader@kali-linux-vm:~/multi/asym#
```

List the files and compare the md6 hashes:

- ➢ ls –l
- ➢ md5sum data.*



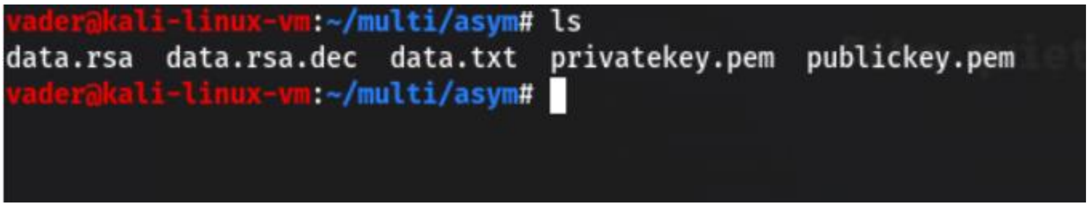
```
vader@kali-linux-vm:~/multi/asym# ls -l
total 20
-rw-r--r-- 1 vader root 128 Dec  1 01:41 data.rsa
-rw-r--r-- 1 vader root  13 Dec  1 01:46 data.rsa.dec
-rw-r--r-- 1 vader root  13 Dec  1 01:24 data.txt
-rw------- 1 vader root 916 Dec  1 01:28 privatekey.pem
-rw-r--r-- 1 vader root 272 Dec  1 01:31 publickey.pem
vader@kali-linux-vm:~/multi/asym# md5sum data.*
d53c4b317b3f7aab41cd2b29b37ac5f4  data.rsa
8ddd8be4b179a529afa5f2ffae4b9858  data.rsa.dec
8ddd8be4b179a529afa5f2ffae4b9858  data.txt
vader@kali-linux-vm:~/multi/asym# 
```

Keep the private key and public key for the next lab.

Module Assessment

Question 5 - What is a private key and what is a public key? What are the name of files used to encrypt and decrypt data.txt in this section?

Ans: The public key is used for encryption and is shared openly or publicly. It can be freely distributed and is used by anyone who wants to send an encrypted message to the owner of the public key. Where as The private key is kept secret and known only to the owner. It is used for decrypting messages that were encrypted with the corresponding public key.



```
vader@kali-linux-vm:~/multi/asym# ls
data.rsa  data.rsa.dec  data.txt  privatekey.pem  publickey.pem
vader@kali-linux-vm:~/multi/asym# 
```

The above files are used to encrypt and decrypt in this module.

The file names are as follows.

Data.txt: plain text

Data.rsa: encrypted

Data.rsa.dec: decrypted

Privatekey.pem: private key

Publickey.pem: public key

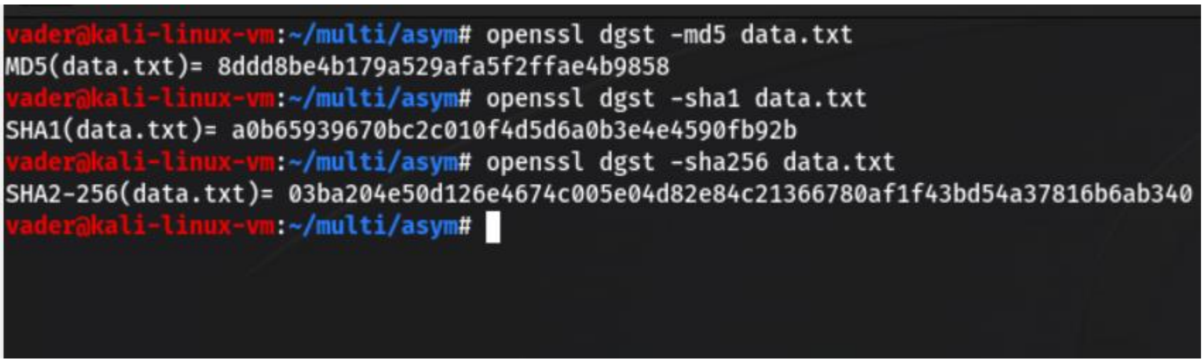
Module Activity Description:

Part five: Digital signatures

In this lab, you will use the key pair created in Lab 5. Please stay in the terminal window within “asym” folder

1. Paste the screenshot of different hash values of the data file(data.txt)
   - Openssl dgst –md5 data.txt

- Openssl dgst –sha1 data.txt
- Openssl dgst –sha256 data.txt



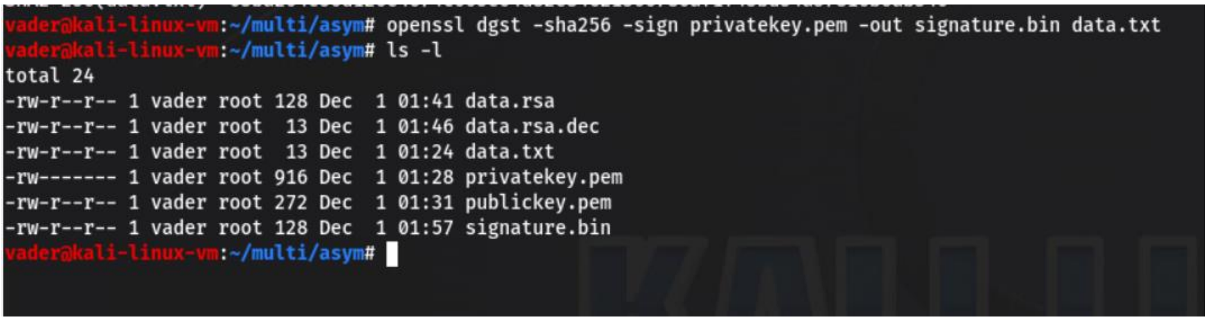
```
vader@kali-linux-vm:~/multi/asym# openssl dgst -md5 data.txt
MD5(data.txt)= 8ddd8be4b179a529afa5f2ffae4b9858
vader@kali-linux-vm:~/multi/asym# openssl dgst -sha1 data.txt
SHA1(data.txt)= a0b65939670bc2c010f4d5d6a0b3e4e4590fb92b
vader@kali-linux-vm:~/multi/asym# openssl dgst -sha256 data.txt
SHA2-256(data.txt)= 03ba204e50d126e4674c005e04d82e84c21366780af1f43bd54a37816b6ab340
vader@kali-linux-vm:~/multi/asym# 
```

Sign the file and check the content of the signature:

- Openssl dgst –sha256 –sign privatekey.pem –out signature.bin data.txt
- ls –l

Description of the sign operation: Hash data.txt with sha256 algorithm and then process (sign) the hash with the private key; the output is the signature.

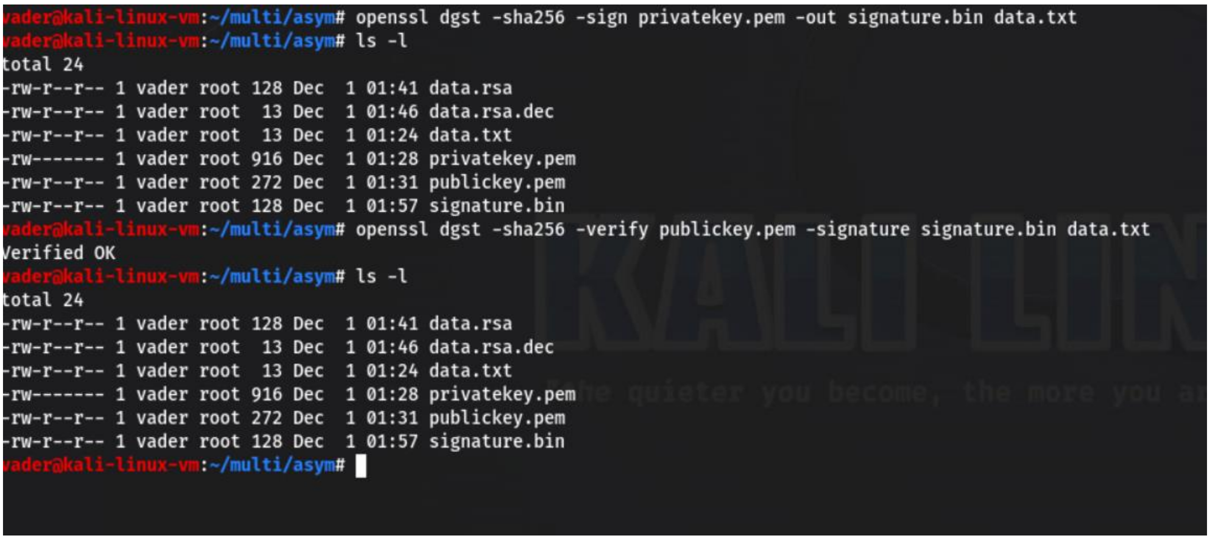
In a real-world scenario, you can send the data.txt and signature.bin to the receiver.



```
vader@kali-linux-vm:~/multi/asym# openssl dgst -sha256 -sign privatekey.pem -out signature.bin data.txt
vader@kali-linux-vm:~/multi/asym# ls -l
total 24
-rw-r--r-- 1 vader root 128 Dec  1 01:41 data.rsa
-rw-r--r-- 1 vader root  13 Dec  1 01:46 data.rsa.dec
-rw-r--r-- 1 vader root  13 Dec  1 01:24 data.txt
-rw------- 1 vader root 916 Dec  1 01:28 privatekey.pem
-rw-r--r-- 1 vader root 272 Dec  1 01:31 publickey.pem
-rw-r--r-- 1 vader root 128 Dec  1 01:57 signature.bin
vader@kali-linux-vm:~/multi/asym# 
```

Receiver to verify the signature:

- Openssl dgst –sha256 –verify publickey.pem –signature signature.bin data.txt
- ls –l

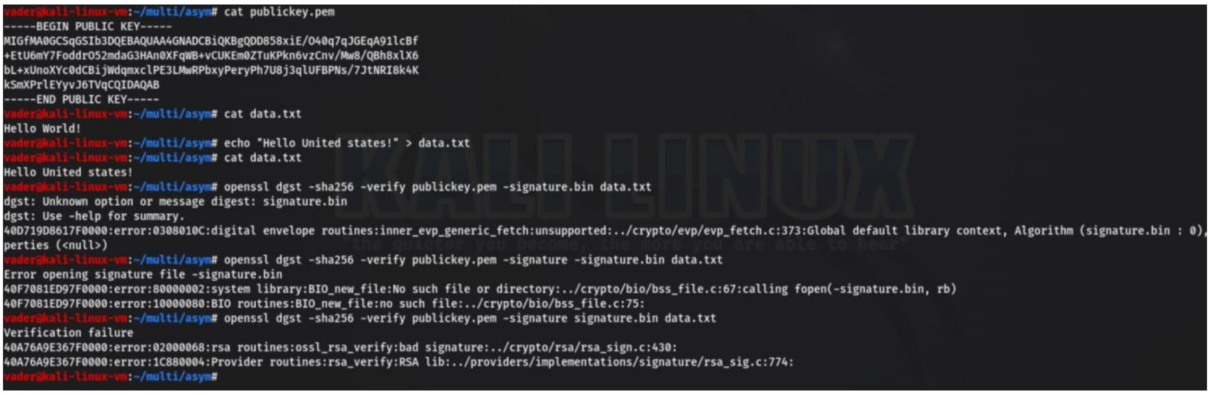


```
ader@kali-linux-vm:~/multi/asym# openssl dgst -sha256 -sign privatekey.pem -out signature.bin data.txt
ader@kali-linux-vm:~/multi/asym# ls -l
otal 24
-rw-r--r-- 1 vader root 128 Dec  1 01:41 data.rsa
-rw-r--r-- 1 vader root  13 Dec  1 01:46 data.rsa.dec
-rw-r--r-- 1 vader root  13 Dec  1 01:24 data.txt
-rw------- 1 vader root 916 Dec  1 01:28 privatekey.pem
-rw-r--r-- 1 vader root 272 Dec  1 01:31 publickey.pem
-rw-r--r-- 1 vader root 128 Dec  1 01:57 signature.bin
ader@kali-linux-vm:~/multi/asym# openssl dgst -sha256 -verify publickey.pem -signature signature.bin data.txt
erified OK
ader@kali-linux-vm:~/multi/asym# ls -l
otal 24
-rw-r--r-- 1 vader root 128 Dec  1 01:41 data.rsa
-rw-r--r-- 1 vader root  13 Dec  1 01:46 data.rsa.dec
-rw-r--r-- 1 vader root  13 Dec  1 01:24 data.txt
-rw------- 1 vader root 916 Dec  1 01:28 privatekey.pem
-rw-r--r-- 1 vader root 272 Dec  1 01:31 publickey.pem
-rw-r--r-- 1 vader root 128 Dec  1 01:57 signature.bin
ader@kali-linux-vm:~/multi/asym# 
```

Module Assessment:

Question 6 - What was used to verify the signature of data.txt. Will verification fail if any of the file – signature.bin, publickey.pem, data.txt changes?

Ans: Since we already know that the public key may be shared with everyone, anybody can check the document, we utilized the public.pem on the receiver side to confirm the signature of the data.txt file.The recipient side of the verification process will fail if any changes are made to the indicated files signature.bin, publickey.pem, or data.txt.



```
vader@kali-linux-vm:~/multi/asym# cat publickey.pem
-----BEGIN PUBLIC KEY-----
MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQDD858xiE/O40q7qJGEqA91lcBf
+EtU6mY7FoddrO52mdaG3HAn0XFqWB+vCUKEm0ZTuKPkn6vzCnv/Mw8/QBh8xlX6
bL+xUnoXYc0dCBijWdqmxclPE3LMwRPbxyPeryPh7U8j3qlUFBPNs/7JtNRI8k4K
kSmXPrlEYyvJ6TVqCQIDAQAB
-----END PUBLIC KEY-----
vader@kali-linux-vm:~/multi/asym# cat data.txt
Hello World!
vader@kali-linux-vm:~/multi/asym# echo "Hello United states!" > data.txt
vader@kali-linux-vm:~/multi/asym# cat data.txt
Hello United states!
vader@kali-linux-vm:~/multi/asym# openssl dgst -sha256 -verify publickey.pem -signature.bin data.txt
dgst: Unknown option or message digest: signature.bin
dgst: Use -help for summary.
40D719D8617F0000:error:0308010C:digital envelope routines:inner_evp_generic_fetch:unsupported:../crypto/evp/evp_fetch.c:373:Global default library context, Algorithm (signature.bin : 0),
perties (<null>)
vader@kali-linux-vm:~/multi/asym# openssl dgst -sha256 -verify publickey.pem -signature -signature.bin data.txt
Error opening signature file -signature.bin
40F7081ED97F0000:error:80000002:system library:BIO_new_file:No such file or directory:../crypto/bio/bss_file.c:67:calling fopen(-signature.bin, rb)
40F7081ED97F0000:error:10000080:BIO routines:BIO_new_file:no such file:../crypto/bio/bss_file.c:75:
vader@kali-linux-vm:~/multi/asym# openssl dgst -sha256 -verify publickey.pem -signature signature.bin data.txt
Verification failure
40A76A9E367F0000:error:02000068:rsa routines:ossl_rsa_verify:bad signature:../crypto/rsa/rsa_sign.c:430:
40A76A9E367F0000:error:1C880004:Provider routines:rsa_verify:RSA lib:../providers/implementations/signature/rsa_sig.c:774:
vader@kali-linux-vm:~/multi/asym#
```

Above picture we have the changed the data.txt file and we can see that verification has failed.