Name: Amit S Terdal
CSU ID: 2887869

3. Exercise 3 - Complete Lab_Cryptography in the Ohio Cyber Range Infrastructure accessed using

Part Zero: Hashing

Open a terminal on your Linux SIFT workstation.
Create a folder and navigate to it

```
mkdir crypto
cd crypto
```

Create a file and calculate its hashes:

```
echo "1" > data1.txt
cat data1.txt
md5sum data1.txt
sha1sum data1.txt
sha512sum data1.txt
```

Create a second file with the same content:

```
echo "1" > data2.txt
cat data2.txt
md5sum data2.txt
```

Create a third file with different content:

```
echo "lots of characters here" > data3.txt
cat data3.txt
md5sum data3.txt
sha1sum data3.txt
sha512sum data3.txt
```

Compare the hash values of all files:

```
md5deep data*.txt
```

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

Answer:

No, the hash length does not change based on the input size or content. Cryptographic hash algorithms are designed to produce fixed-length outputs, regardless of the size or type of data being hashed. This is a fundamental property of hash functions, ensuring consistency and predictability in cryptographic operations.

1) Fixed-Length Outputs:

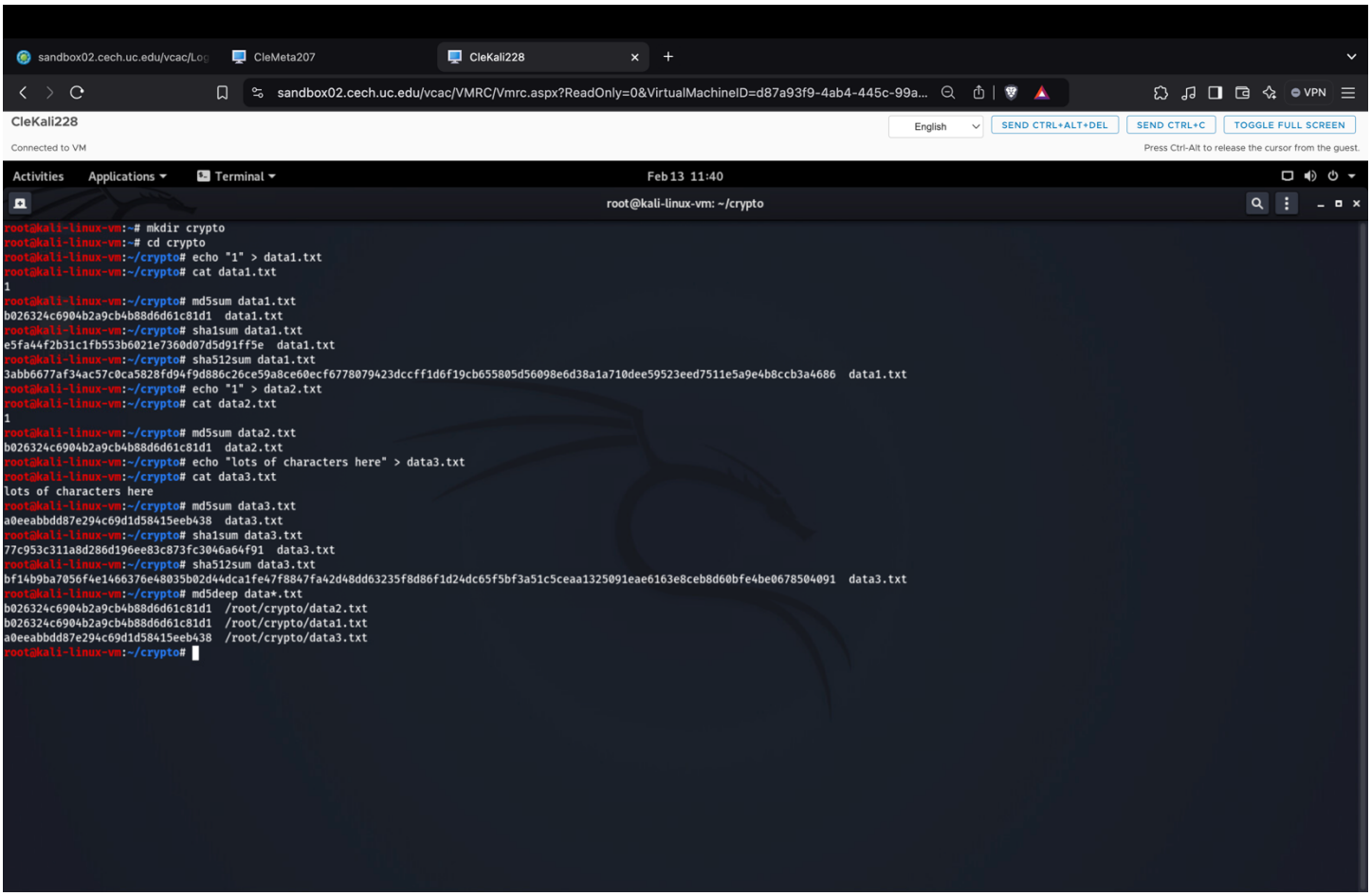
MD5: Produces a 32-character (128-bit) hash.

SHA1: Produces a 40-character (160-bit) hash.

SHA512: Produces a 128-character (512-bit) hash.

2) Proof:

When you create files with different content (e.g., data1.txt with "1" and data3.txt with "lots of characters here") and calculate their hashes using md5sum, sha1sum, and sha512sum, you will observe that the hash lengths remain constant for each algorithm. Even though the input sizes differ, the output hash lengths do not change. This fixed-length property is crucial for applications like data integrity checks, digital signatures, and password storage, where consistent hash lengths are required.



Part One: Encoding & Decoding

Steps:

Create a folder and navigate to it:

```
mkdir encode
cd encode
```

Create a file and check its content:

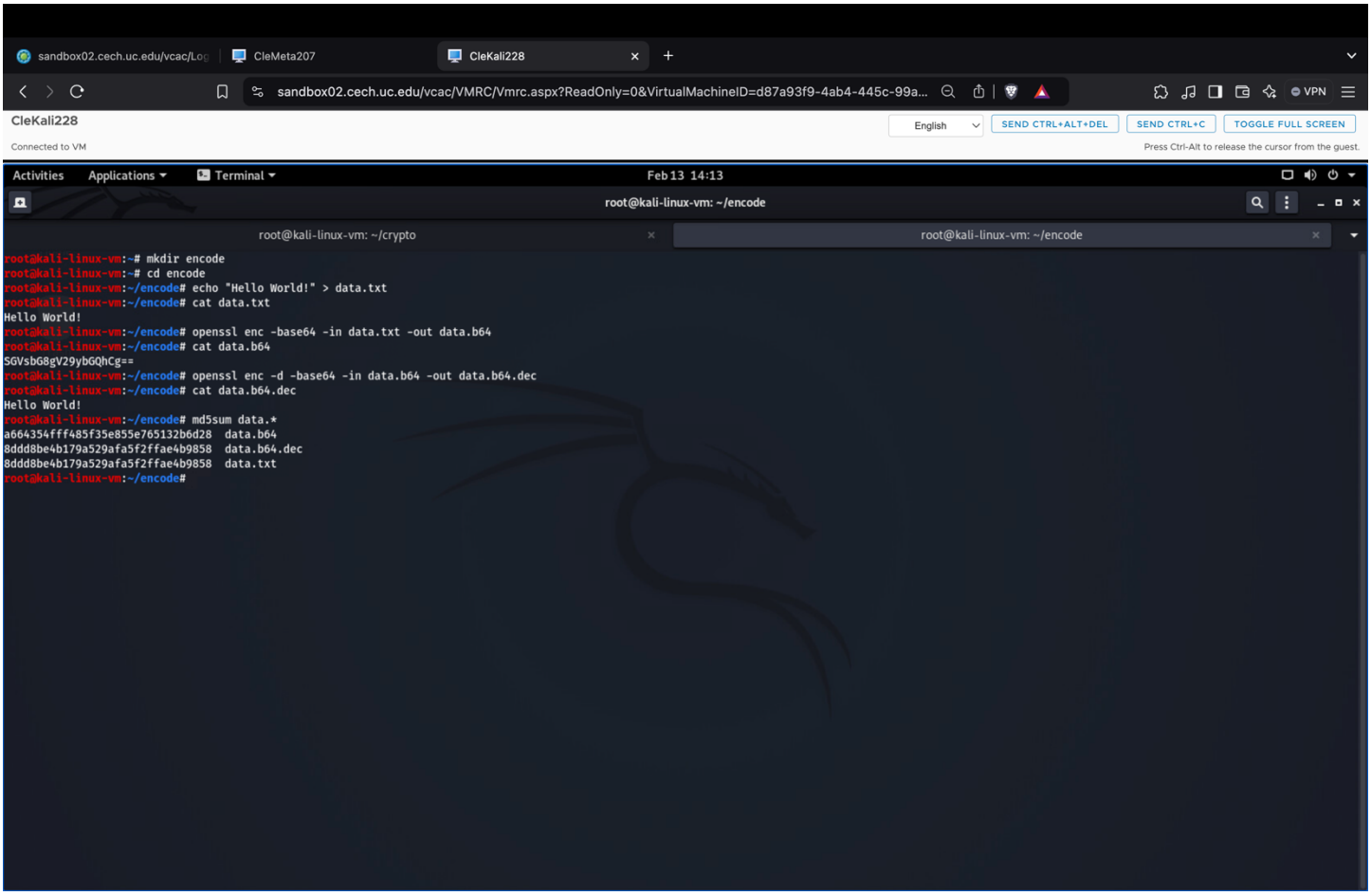
```
echo "Hello World!" > data.txt
cat data.txt
```

Perform Base64 encoding:

```
openssl enc -base64 -in data.txt -out data.b64
cat data.b64
```

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.cd



In the encode folder, three files are created:

data.txt: Contains the original text "Hello World!".

data.b64: Contains the Base64-encoded version of data.txt.

data.b64.dec: Contains the decoded (original) content of data.b64.

MD5 Hash Comparison:

data.txt and data.b64.dec will have the same MD5 hash because they contain identical content ("Hello World!").

data.b64 will have a different MD5 hash because it contains the Base64-encoded version of the original data, which is not the same as the original text.

Conclusion:

The MD5 hash of a file depends on its content. If two files have the same content, their MD5 hashes will match. However, if the content differs (even due to encoding), the hashes will differ.

Part Two: Symmetric Encryption

Steps:

Create a folder and navigate to it:

```
mkdir sym
cd sym
```

Create a file and check its content:

```
echo "Hello World!" > data.txt
cat data.txt
```

Encrypt the file using AES-256-CBC:

```
openssl enc -aes-256-cbc -in data.txt -out data.aes
```

Enter a password when prompted.

Check the encrypted content:

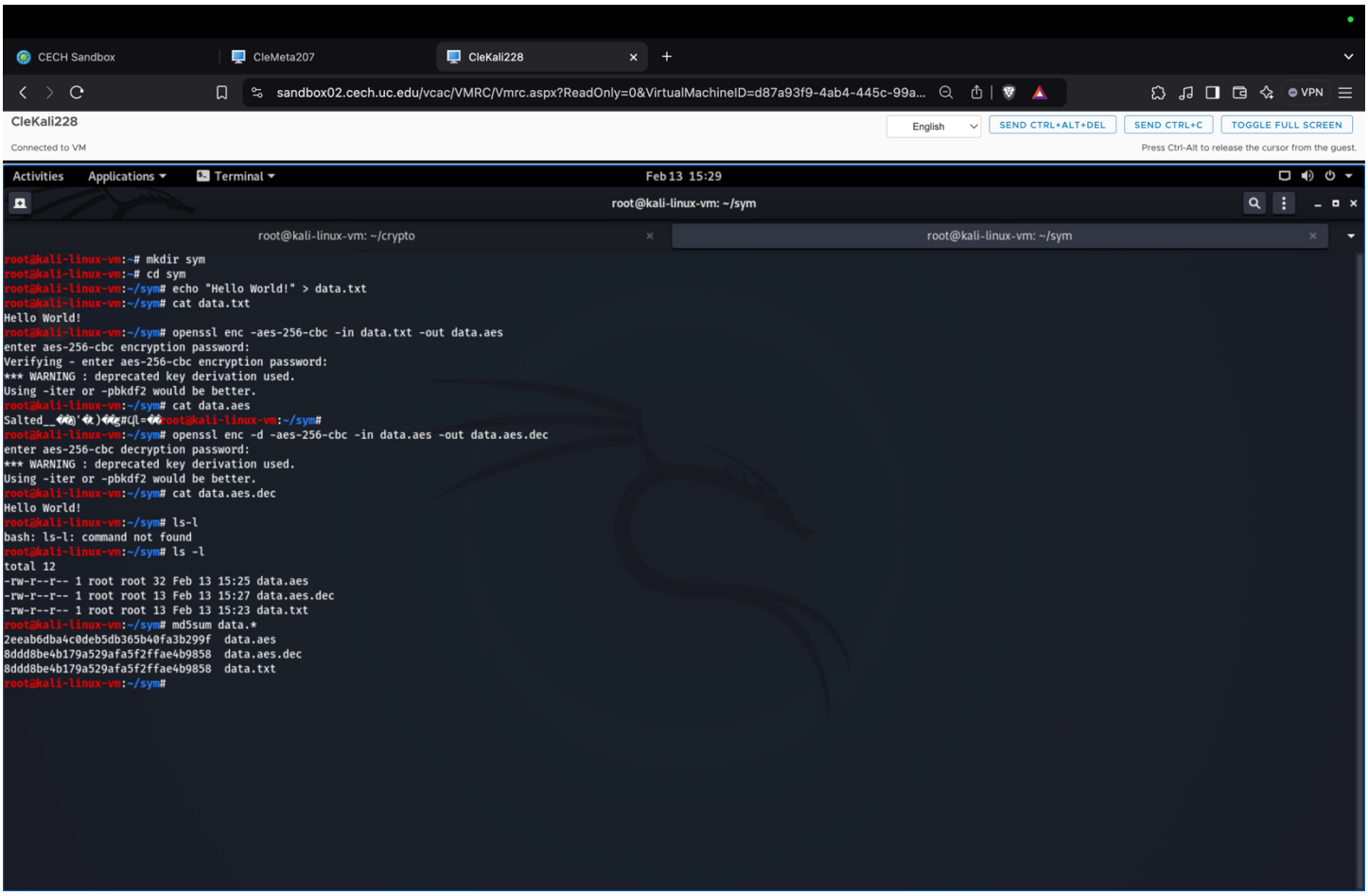
```
cat data.aes
```

Decrypt the file:

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

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?



In the sym folder, the following files are created:

data.txt: Contains the original text "Hello World!".

data.aes: Contains the AES-encrypted version of data.txt.

data.aes.dec: Contains the decrypted version of data.aes.

MD5 Hash Comparison:

data.txt and data.aes.dec will have the same MD5 hash because they contain identical content.

data.aes will have a different MD5 hash because it contains encrypted binary data, which is not the same as the original text.

Utility Used:

The openssl utility was used for AES encryption. AES (Advanced Encryption Standard) is a symmetric encryption algorithm, meaning the same key is used for both encryption and decryption. The openssl command-line tool provides a simple way to perform AES encryption and decryption.

Explanation of AES:

AES is a widely used encryption standard that operates on fixed block sizes (128 bits) and supports key lengths of 128, 192, or 256 bits. It is secure, efficient, and suitable for encrypting sensitive data.

Part Three: Multiple Operations

Steps:

Create a folder and navigate to it:

```
mkdir multi
cd multi
```

Create a file and check its content:

```
echo "Hello World!" > data.txt
cat data.txt
```

Encrypt the file using AES-256-CBC and encode it in Base64:

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

Enter a password when prompted.

Check the encoded content:

```
cat data.aes.b64
```

Decode and decrypt the file:

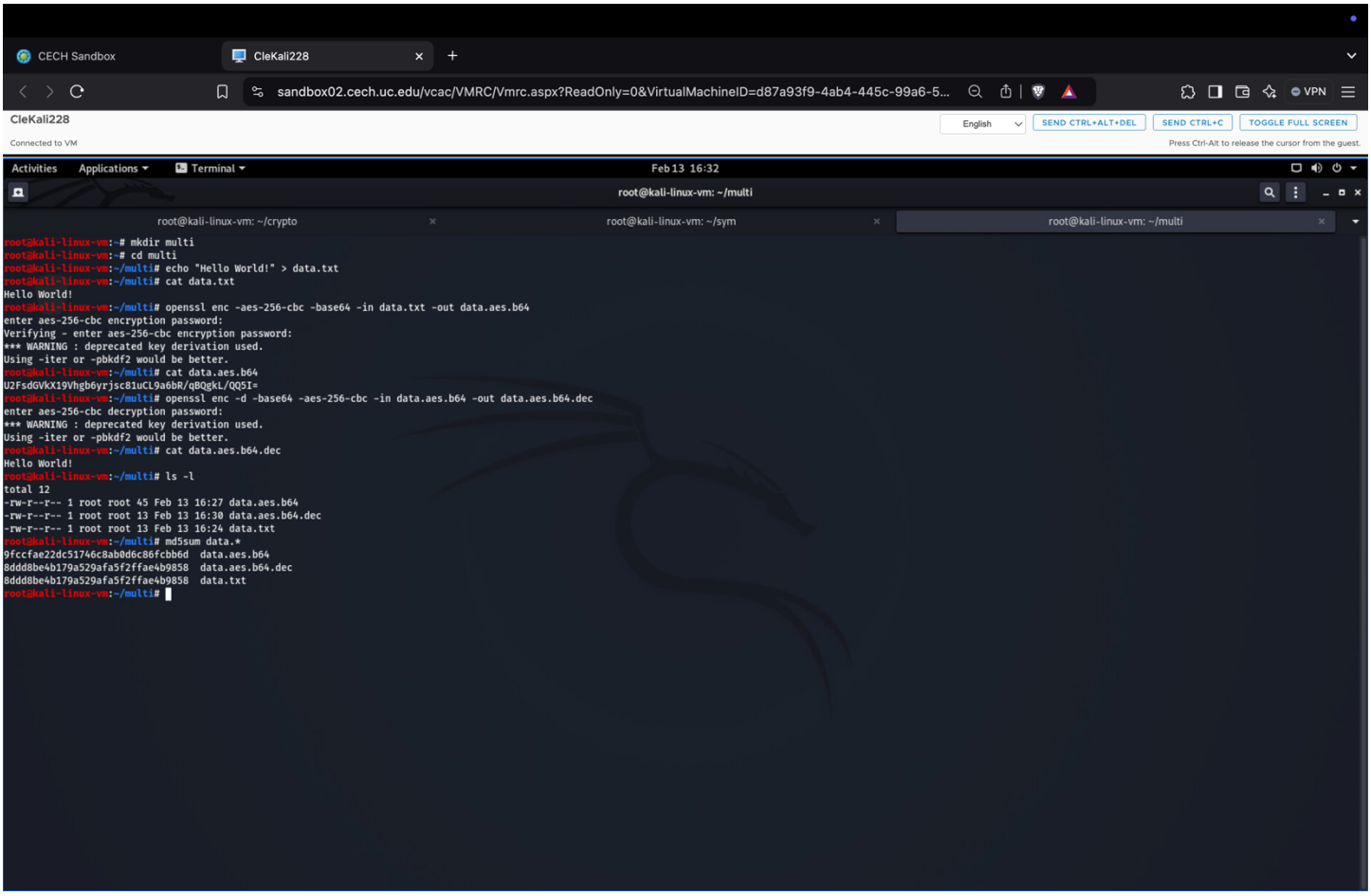
```
openssl enc -d -base64 -aes-256-cbc -in data.aes.b64 -out data.aes.b64.dec
cat data.aes.b64.dec
```

Compare the MD5 hashes:

```
ls -l
md5sum data.*
```

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.



1) Encoding Technique:

Base64 encoding was used on the AES-encrypted file in the multiple operations step.

2) Comparison of Files:

data.aes (from Part Two): Contains raw binary data resulting from AES encryption. It is not human-readable.

3) data.aes.b64 (from Part Three):

Contains the Base64-encoded version of the AES-encrypted data. Base64 encoding converts binary data into a text format, making it human-readable.

4) Difference:

The key difference is that data.aes is in binary format, while data.aes.b64 is in a text-based format due to Base64 encoding. Base64 encoding is often used to safely transmit binary data over text-based protocols (e.g., email).

Part Four: Asymmetric Encryption

Steps:

Create a folder and navigate to it:

```
mkdir asym
cd asym
```

Create a file and check its content:

```
echo "Hello World!" > data.txt
cat data.txt
```

Generate a private key:

```
openssl genpkey -algorithm RSA -pkeyopt rsa_keygen_bits:1024 -out privatekey.pem
cat privatekey.pem
```

Generate the public key:

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

Encrypt the file using the public key:

```
openssl rsautl -encrypt -inkey publickey.pem -pubin -in data.txt -out data.rsa
cat data.rsa
```

Decrypt the file using the private key:

```
openssl rsautl -decrypt -inkey privatekey.pem -in data.rsa -out data.rsa.dec
cat data.rsa.dec
```

Compare the MD5 hashes:

```
ls -l
md5sum data.txt data.rsa data.rsa.dec
```

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?

1) Private Key:
   A private key is a cryptographic key used to decrypt data that has been encrypted with the corresponding public key. It must be kept secret and secure, as anyone with access to the private key can decrypt the data.
2) Public Key:
   A public key is a cryptographic key used to encrypt data. It can be shared openly, as it cannot be used to decrypt the data. Only the corresponding private key can decrypt the data.
3) Files Used:
   Encryption: The public key stored in publickey.pem was used to encrypt data.txt.
   Decryption: The private key stored in privatekey.pem was used to decrypt the encrypted file (data.rsa).

sandbox02.cech.uc.edu/vcac/Log | CleKali228

sandbox02.cech.uc.edu/vcac/VMRC/Vmrc.aspx?ReadOnly=0&VirtualMachineID=d87a93f9-4ab4-445c-99a...

CleKali228

Connected to VM

English | SEND CTRL+ALT+DEL | SEND CTRL+C | TOGGLE FULL SCREEN

Press Ctrl-Alt to release the cursor from the guest.

Activities | Applications | Terminal | Feb 13 17:56

root@kali-linux-vm: ~/asym

root@kali-linux-vm: ~/crypto | root@kali-linux-vm: ~/sym | root@kali-linux-vm: ~/multi | root@kali-linux-vm: ~/asym

```
root@kali-linux-vm:~# mkdir asym
root@kali-linux-vm:~# cd asym
root@kali-linux-vm:~/asym# echo "Hello World!" > data.txt
root@kali-linux-vm:~/asym# cat data.txt
Hello World!
root@kali-linux-vm:~/asym# openssl genpkey -algorithm rsa -pkeyopt rsa_keygen_bits:1024 -out privatekey.pem
.............................................+++++
...+++++
root@kali-linux-vm:~/asym# cat privatekey.pem
-----BEGIN PRIVATE KEY-----
MIICeAIBADANBgkqhkiG9w0BAQEFAASCAmIwggJeAgEAAoGBAL3BidJkvHHKC5s2
+8ClJoCaZq8AuSs/rBiVgUnJP/GpGY6quJSq95iuGZkvZbpGRMA08sykbqvwxumx
jqPPVer9VTMt0+6p+xlUT5b6iqGcYya/j9YBOaKFSlYnciBk8o5o4MYTLhMmOXrO
mypM8TQHFx2IwCju9LEO0wUUSb2NAgMBAAECgYEAnsxhh1XQMjXQorNP2bcb0pHk
Dt4/GAdwAR7UlKg7wAQwpCkQ0TGGhRplx8FXk5yMmoF1E3ckOvYDsLxrz4CKt+pH
a3fTTey30RYdT+/jRIKQnC3HS0UU8+wjPk7ghJ4YZvkPZ7TU4bwjlQ42qar5qNKf
jskW3d6x5B50E83XbYECQQD5qc7iTMpVCvtImQlxHSVoPWfiChksKysohCHvgtgo
Egjum/iyKsuhQ9k/1QbmQcPpoOejV3xO7h663XN33AKhAkEAwpJ7P+gJAvqhQ7NG
oi8nz/32xfGTRPCzR/9eAgJzOqe144rUA+wQcyMBQE1Kxly+9LTP0xNmee3UOGQS
Gpc/bQJBANORe9k3+8o/iRxHW8H4UgfY6ZyatCRC3Mrz1hIyzlx+lAUoNYM+Uttx
NLwsGTGtD4l/Kmy4KYZ1saLnIP+0eKECQECEV8/p/cr6Fi0frgUrQ7VJDeG259FN
SvVNrPtXmqPRZpOLvh+K0C2NZuXOe0PpDvyPU/Y2kZ1Oijckn4gnrMUCQQDwOMKW
t1QUIUig2tPlxpGbwIj892bGy8/0LfaD19jbBSviJSbcTJAFyomKutbeYvfWYFbf
gH3DzXSO6PG21Xs7
-----END PRIVATE KEY-----
root@kali-linux-vm:~/asym# openssl rsa -pubout -in privatekey.pem -out publickey.pem
writing RSA key
root@kali-linux-vm:~/asym# cat publickey.pem
-----BEGIN PUBLIC KEY-----
MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQC9wYnSZLxxygubNvvApSaAmmav
ALkrP6wYlYFJyT/xqRmOqriUqveYrhmZL2W6RkTANPLMpG6r8MbpsY6jz1Xq/VUz
LdPuqfsZVE+W+oqhnGMmv4/WATmihUpWJ3IgZPKOaODGEy4TJjl6zpsqTPE0Bxcd
iMAo7vSxDtMFFEm9jQIDAQAB
-----END PUBLIC KEY-----
root@kali-linux-vm:~/asym# openssl rsautl -encrypt -inkey publickey.pem -pubin -in data.txt -out data.rsa
root@kali-linux-vm:~/asym# cat data.rsa
h[illegible]|0[illegible]
.[illegible]mh[illegible]?R!2U[illegible]SS>`[illegible]Wr[illegible]I[illegible]t[illegible]Iroot@kali-linux-vm:~/asym#
root@kali-linux-vm:~/asym# openssl rsautl -decrypt -inkey privatekey.pem -in data.rsa -out data.rsa.dec
root@kali-linux-vm:~/asym# cat data.rsa.dec
Hello World!
root@kali-linux-vm:~/asym# ls -l
total 20
-rw-r--r-- 1 root root 128 Feb 13 17:38 data.rsa
-rw-r--r-- 1 root root  13 Feb 13 17:44 data.rsa.dec
-rw-r--r-- 1 root root  13 Feb 13 17:16 data.txt
-rw------- 1 root root 916 Feb 13 17:26 privatekey.pem
-rw-r--r-- 1 root root 272 Feb 13 17:32 publickey.pem
root@kali-linux-vm:~/asym# md5sum data.*
46defd04bee5633ca87e54a5521c115b  data.rsa
```

sandbox02.cech.uc.edu/vcac/Log

CleKali228

sandbox02.cech.uc.edu/vcac/VMRC/Vmrc.aspx?ReadOnly=0&VirtualMachineID=d87a93f9-4ab4-445c-99a...

VPN

CleKali228

Connected to VM

English | SEND CTRL+ALT+DEL | SEND CTRL+C | TOGGLE FULL SCREEN

Press Ctrl-Alt to release the cursor from the guest.

Activities | Applications | Terminal | Feb 13 18:01

root@kali-linux-vm: ~/asym

root@kali-linux-vm: ~/crypto | root@kali-linux-vm: ~/sym | root@kali-linux-vm: ~/multi | root@kali-linux-vm: ~/asym

```
root@kali-linux-vm:~/asym# cat data.txt
Hello World!
root@kali-linux-vm:~/asym# openssl genpkey -algorithm rsa -pkeyopt rsa_keygen_bits:1024 -out privatekey.pem
.............................................+++++
...+++++
root@kali-linux-vm:~/asym# cat privatekey.pem
-----BEGIN PRIVATE KEY-----
MIICeAIBADANBgkqhkiG9w0BAQEFAASCAmIwggJeAgEAAoGBAL3BidJkvHHKC5s2
+8ClJoCaZq8AuSs/rBiVgUnJP/GpGY6quJSq95iuGZkvZbpGRMA08sykbqvwxumx
jqPPVer9VTMt0+6p+xlUT5b6iqGcYya/j9YBOaKFSlYnciBk8o5o4MYTLhMmOXrO
mypM8TQHFx2IwCju9LEO0wUUSb2NAgMBAAECgYEAnsxhh1XQMjXQorNP2bcb0pHk
Dt4/GAdwAR7UlKg7wAQwpCkQ0TGGhRplx8FXk5yMmoF1E3ckOvYDsLxrz4CKt+pH
a3fTTey30RYdT+/jRIKQnC3HS0UU8+wjPk7ghJ4YZvkPZ7TU4bwjlQ42qar5qNKf
jskW3d6x5B50E83XbYECQQD5qc7iTMpVCvtImQlxHSVoPWfiChksKysohCHvgtgo
Egjum/iyKsuhQ9k/1QbmQcPpoOejV3xO7h663XN33AKhAkEAwpJ7P+gJAvqhQ7NG
oi8nz/32xfGTRPCzR/9eAgJzOqe144rUA+wQcyMBQE1Kxly+9LTP0xNmee3UOGQS
Gpc/bQJBANORe9k3+8o/iRxHW8H4UgfY6ZyatCRC3Mrz1hIyzlx+lAUoNYM+Uttx
NLwsGTGtD4l/Kmy4KYZ1saLnIP+0eKECQECEV8/p/cr6Fi0frgUrQ7VJDeG259FN
SvVNrPtXmqPRZpOLvh+K0C2NZuXOe0PpDvyPU/Y2kZ1Oijckn4gnrMUCQQDwOMKW
t1QUIUig2tPlxpGbwIj892bGy8/0LfaD19jbBSviJSbcTJAFyomKutbeYvfWYFbf
gH3DzXSO6PG21Xs7
-----END PRIVATE KEY-----
root@kali-linux-vm:~/asym# openssl rsa -pubout -in privatekey.pem -out publickey.pem
writing RSA key
root@kali-linux-vm:~/asym# cat publickey.pem
-----BEGIN PUBLIC KEY-----
MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQC9wYnSZLxxygubNvvApSaAmmav
ALkrP6wYlYFJyT/xqRmOqriUqveYrhmZL2W6RkTANPLMpG6r8MbpsY6jz1Xq/VUz
LdPuqfsZVE+W+oqhnGMmv4/WATmihUpWJ3IgZPKOaODGEy4TJjl6zpsqTPE0Bxcd
iMAo7vSxDtMFFEm9jQIDAQAB
-----END PUBLIC KEY-----
root@kali-linux-vm:~/asym# openssl rsautl -encrypt -inkey publickey.pem -pubin -in data.txt -out data.rsa
root@kali-linux-vm:~/asym# cat data.rsa
h[illegible]
.[illegible]
root@kali-linux-vm:~/asym#
root@kali-linux-vm:~/asym# openssl rsautl -decrypt -inkey privatekey.pem -in data.rsa -out data.rsa.dec
root@kali-linux-vm:~/asym# cat data.rsa.dec
Hello World!
root@kali-linux-vm:~/asym# ls -l
total 20
-rw-r--r-- 1 root root 128 Feb 13 17:38 data.rsa
-rw-r--r-- 1 root root  13 Feb 13 17:44 data.rsa.dec
-rw-r--r-- 1 root root  13 Feb 13 17:16 data.txt
-rw------- 1 root root 916 Feb 13 17:26 privatekey.pem
-rw-r--r-- 1 root root 272 Feb 13 17:32 publickey.pem
root@kali-linux-vm:~/asym# md5sum data.*
46defd04bee5633ca87e54a5521c115b  data.rsa
8ddd8be4b179a529afa5f2ffae4b9858  data.rsa.dec
8ddd8be4b179a529afa5f2ffae4b9858  data.txt
root@kali-linux-vm:~/asym#
```

Part Five: Digital Signatures

Steps:

Calculate hash values of data.txt:

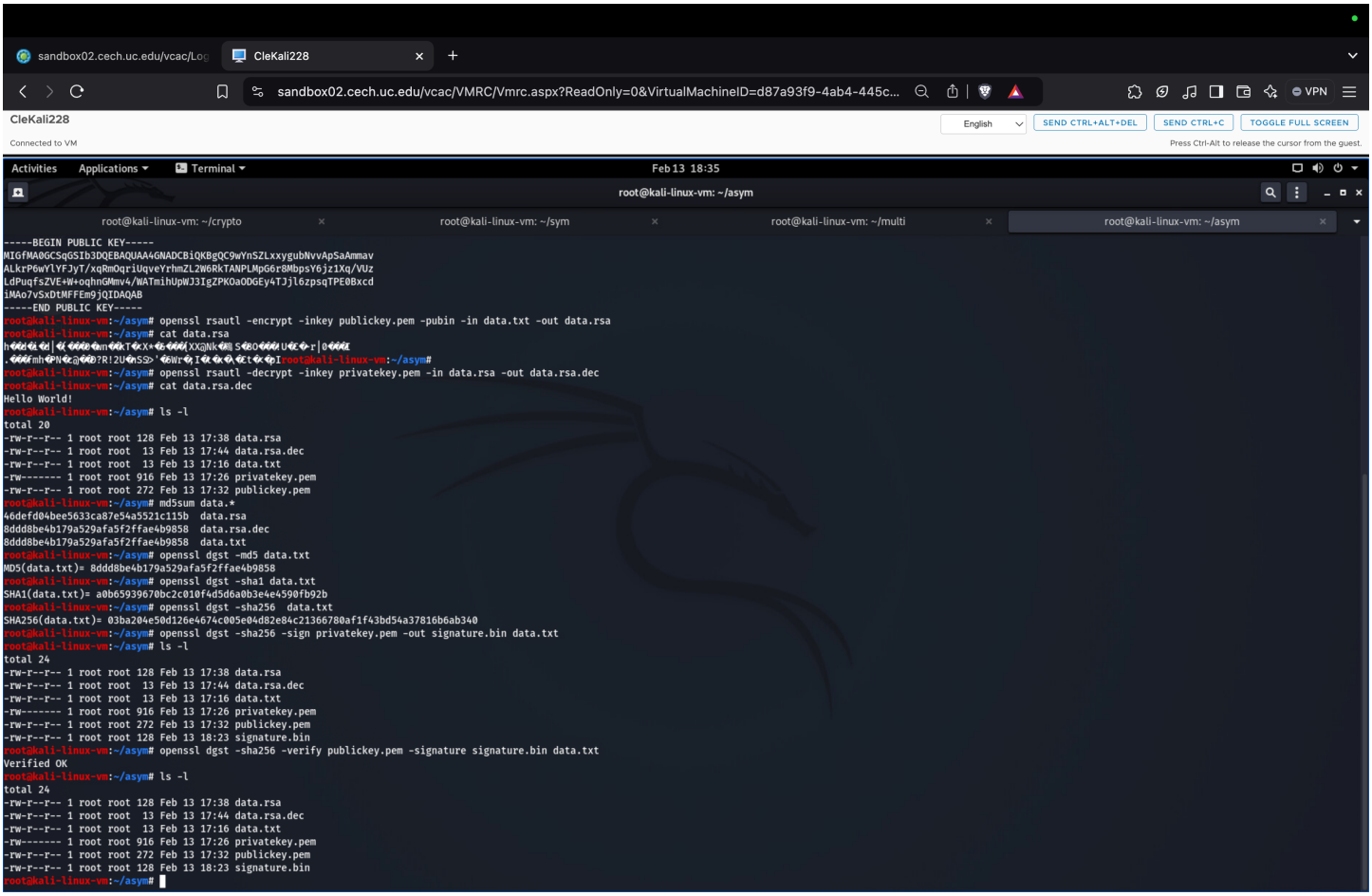
```
openssl dgst -md5 data.txt
openssl dgst -sha1 data.txt
openssl dgst -sha256 data.txt
```

Sign the file using the private key:

```
openssl dgst -sha256 -sign privatekey.pem -out signature.bin data.txt
ls -l
```

Verify the signature using the public key:

```
openssl dgst -sha256 -verify publickey.pem -signature signature.bin data.txt
ls -l
```



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?

1) Verification Tool:

The public key (publickey.pem) was used to verify the digital signature of data.txt. The openssl dgst -verify command was used for this purpose.

2) Verification Failure:

Yes, the verification will fail if any of the following files are modified:

3) signature.bin:

The signature file contains the hash of the original data signed with the private key. If this file changes, the signature will no longer match the data.

4) publickey.pem:

The public key is required to verify the signature. If it changes, the verification process will fail because the key pair (public and private) will no longer match.

5) data.txt:

If the content of the original file changes, its hash will also change, causing the signature verification to fail.

6) Conclusion:

Digital signatures ensure data integrity and authenticity. Any alteration to the data, signature, or public key will result in a failed verification, indicating potential tampering or corruption.