Al-Najah National University
Department of Engineering and Information technology
Computer Network and Information Security

Hash Function and Message Authentication Code (MAC)

Mohammed Adnan
Instructor: Dr. Ahmed Awad

20/2/2021

1 Abstract

This experiment aims to gain the basic knowledge of cryptographic hash functions alongside with Message Authentication Code (MAC) with proper Linux tools.

2 Introduction

A cryptographic hash function **(CHF)** is a mathematical algorithm that maps data of arbitrary size (often called the ”message”) to a bit array of a fixed size (the ”hash value”, ”hash”, or ”message digest”). It is a one-way function, that is, a function which is practically infeasible to invert. Ideally, the only way to find a message that produces a given hash is to attempt a brute-force search of possible inputs to see if they produce a match, or use a rainbow table of matched hashes. Cryptographic hash functions are a basic tool of modern cryptography . Cryptographic hash functions have many information-security applications, notably in digital signatures, message authentication codes **(MACs)**, and other forms of authentication. They can also be used as ordinary hash functions, to index data in hash tables, for fingerprinting, to detect duplicate data or uniquely identify files, and as checksums to detect accidental data corruption. Indeed, in information-security contexts, cryptographic hash values are sometimes called (digital) fingerprints, checksums, or just hash values, even though all these terms stand for more general functions with rather different properties and purposes. A message authentication code (MAC), sometimes known as a tag, is a short piece of information used to authenticate a message—in other words, to confirm that the message came from the stated sender (its authenticity) and has not been changed. The MAC value protects a message's data integrity, as well as its authenticity, by allowing verifiers (who also possess the secret key) to detect any changes to the message content. While MAC functions are similar to cryptographic hash functions, they possess different security requirements. To be considered secure, a MAC function must resist existential forgery under chosen-plaintext attacks. This means that even if an attacker has access to an oracle which possesses the secret key and generates MACs for messages of the attacker's choosing, the attacker cannot guess the MAC for other messages (which were not used to query the oracle) without performing infeasible amounts of computation. MACs differ from digital signatures as MAC values are both generated and verified using the same secret key. This implies that the sender and receiver of a message must agree on the same key before initiating communications, as is the case with symmetric encryption.

3 Procedure

3.1 Message Digest Generation

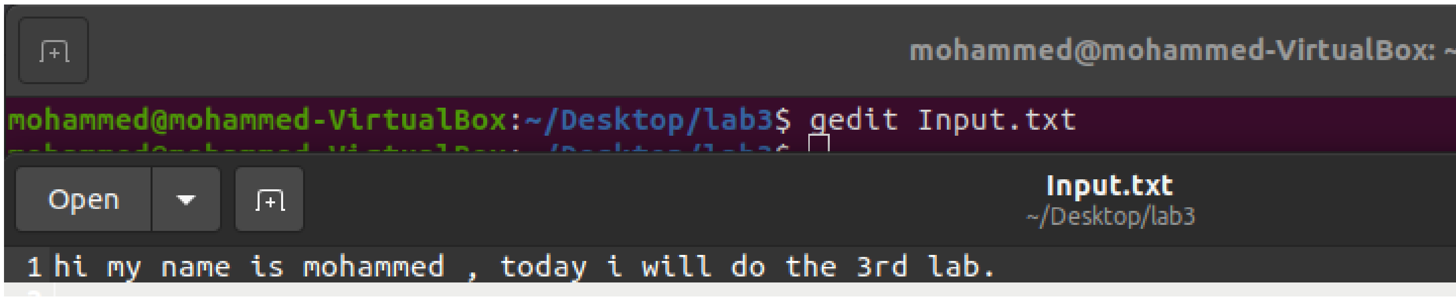
1. **Question:** What is the purpose of hash functions?

   The purpose of the hash function is to achieve integrity and to ensure that message doesn't change.

2. **Question :** What do we mean by Message Authentication Code (MAC) ?

   The message authentication code is a short piece of information used to authenticate a message, in other words, to confirm that the message came from the stated sender (its authenticity) and has not been changed. The MAC value protects a message's data integrity, as well as its authenticity, by allowing verifiers (who also possess the secret key) to detect any changes to the message content.

3. We create file text **input.txt** and fill it with some arbitrary text, as figure 1 shows.

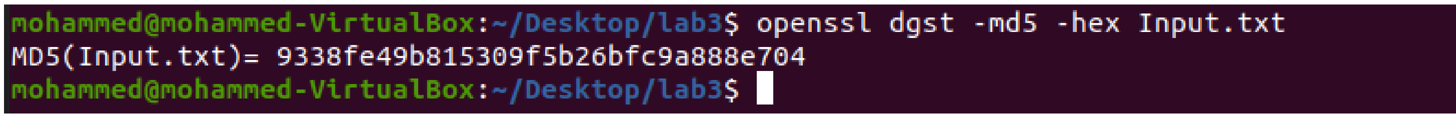


```
mohammed@mohammed-VirtualBox:~/Desktop/lab3$ gedit Input.txt
```

```
hi my name is mohammed , today i will do the 3rd lab.
```

Figure 1: Creating file text.

4. We Generated a hash code (message digest) for our input file using the md5 algorithm, as shown in figure 2.

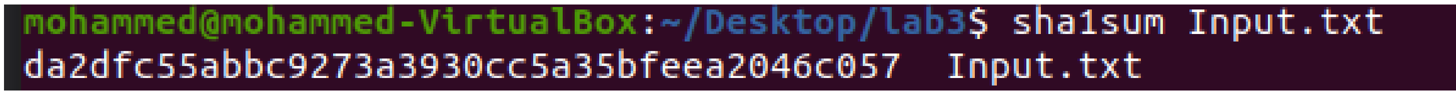


```
mohammed@mohammed-VirtualBox:~/Desktop/lab3$ openssl dgst -md5 -hex Input.txt
MD5(Input.txt)= 9338fe49b815309f5b26bfc9a888e704
mohammed@mohammed-VirtualBox:~/Desktop/lab3$
```

Figure 2: Md5 hash code .

   Question:What is the hash code size? 128 bits **(32 bytes)**.

5. We then Generated the hash code of the input file by using the sha1 algorithm, as shown in figure 3.

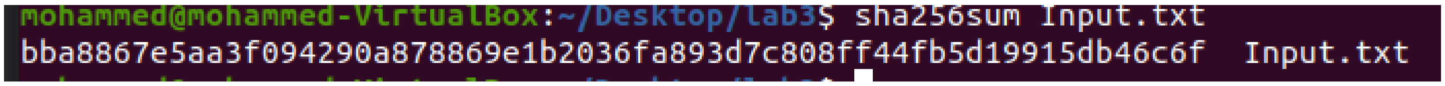


```
mohammed@mohammed-VirtualBox:~/Desktop/lab3$ sha1sum Input.txt
da2dfc55abbc9273a3930cc5a35bfeea2046c057  Input.txt
```

Figure 3: Sha 1 hash code.

   Question:What is the hash code size? 160 bits **(40 bytes)**.

6. We also Generated the hash code of the input file using sha256 algorithm, as shown in figure 4.



```
mohammed@mohammed-VirtualBox:~/Desktop/lab3$ sha256sum Input.txt
bba8867e5aa3f094290a878869e1b2036fa893d7c808ff44fb5d19915db46c6f  Input.txt
```

Figure 4: Sha 256 hash code.

Question:What is the hash code size? 256 bits **(64 byts)**.

3.2 Message Authentication Code (MAC) Keyed Hash

1. **Question :** What is the purpose of MAC (Keyed hash)? Explain it with a simple block diagram .

   MACs are cryptographic checksums. They are used to detect when an attacker has tampered with a message, the message generates a hash code then encrypts it with a secret key, after this the new value will append to the message. Figure 5 shows how it's done .

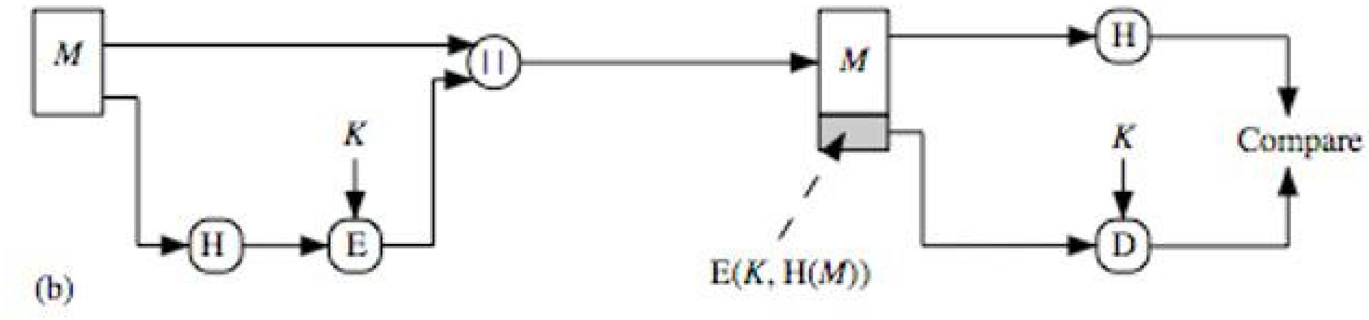
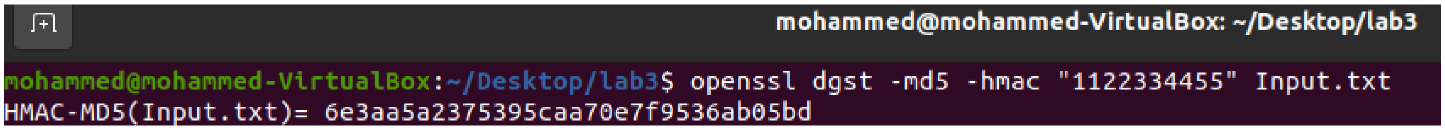


Figure 5: MAC block diagram .

2. We Generated a keyed hash using MD5 for the input file that have we made , as shown in figure 6

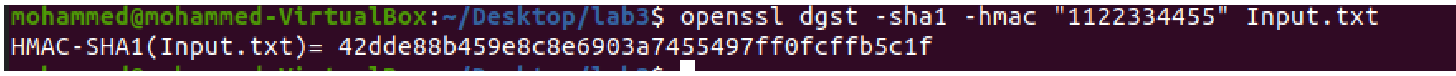


```
mohammed@mohammed-VirtualBox: ~/Desktop/lab3
mohammed@mohammed-VirtualBox:~/Desktop/lab3$ openssl dgst -md5 -hmac "1122334455" Input.txt
HMAC-MD5(Input.txt)= 6e3aa5a2375395caa70e7f9536ab05bd
```

Figure 6: Mac using MD5 .

Question: How will you do that? By using the dgst hmac function in the openssl tool.

3. Then We Generated a keyed hash using Sh1 for the input file that have we made , as shown in figure 7

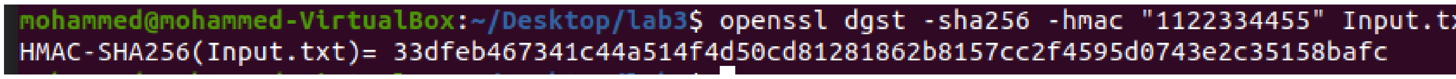


```
mohammed@mohammed-VirtualBox:~/Desktop/lab3$ openssl dgst -sha1 -hmac "1122334455" Input.txt
HMAC-SHA1(Input.txt)= 42dde88b459e8c8e6903a7455497ff0fcffb5c1f
```

Figure 7: Mac using Sha 1 .

Question: How will you do that? By using the dgst hmac function in the openssl tool.

4. Also We Generated a keyed hash using Sh1 for the input file that have we made , as shown in figure 8



```
mohammed@mohammed-VirtualBox:~/Desktop/lab3$ openssl dgst -sha256 -hmac "1122334455" Input.t
HMAC-SHA256(Input.txt)= 33dfeb467341c44a514f4d50cd81281862b8157cc2f4595d0743e2c35158bafc
```

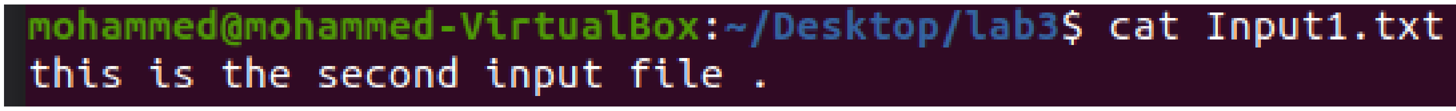
Figure 8: Mac using Sha 256 .

Question: How will you do that? By using the dgst hmac function in the openssl tool.

5. **Question :** What is the relation between the key size, the text file size and the hash code size?
   Hash code size and **Key** size are fixed lengths, depending on the algorithms, but the **message** has a variable length.

3.3 One-Way Hash Functions and Randomness

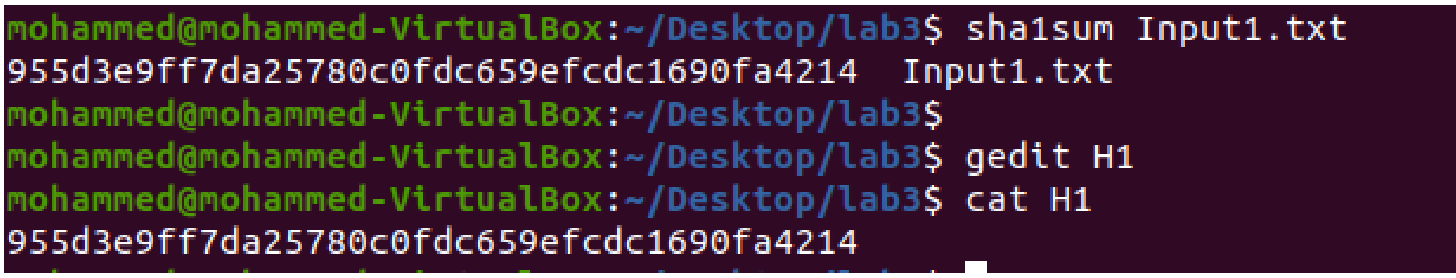
1. We create file text **input1.txt** and fill it with some arbitrary text, as figure 9 shows.



```
mohammed@mohammed-VirtualBox:~/Desktop/lab3$ cat Input1.txt
this is the second input file .
```

Figure 9: Creating file text.

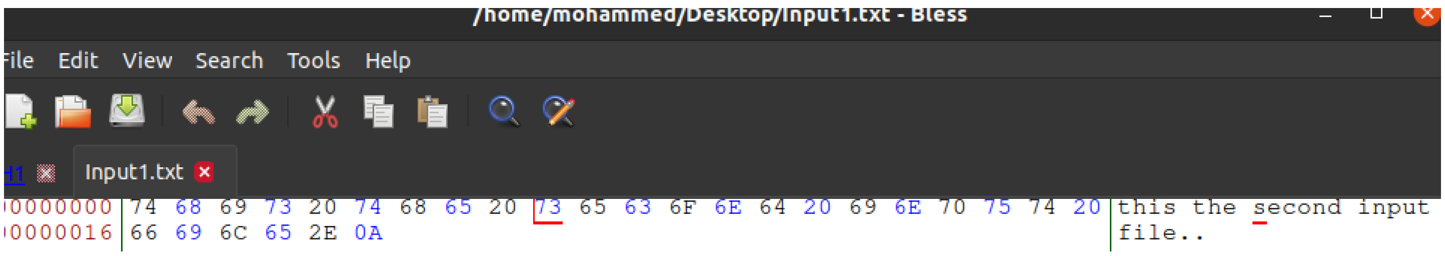
2. We Generated a hash code (H1) for the text file and saved it in a file named H1, as figure 10 shows.



```
mohammed@mohammed-VirtualBox:~/Desktop/lab3$ sha1sum Input1.txt
955d3e9ff7da25780c0fdc659efcdc1690fa4214  Input1.txt
mohammed@mohammed-VirtualBox:~/Desktop/lab3$
mohammed@mohammed-VirtualBox:~/Desktop/lab3$ gedit H1
mohammed@mohammed-VirtualBox:~/Desktop/lab3$ cat H1
955d3e9ff7da25780c0fdc659efcdc1690fa4214
```

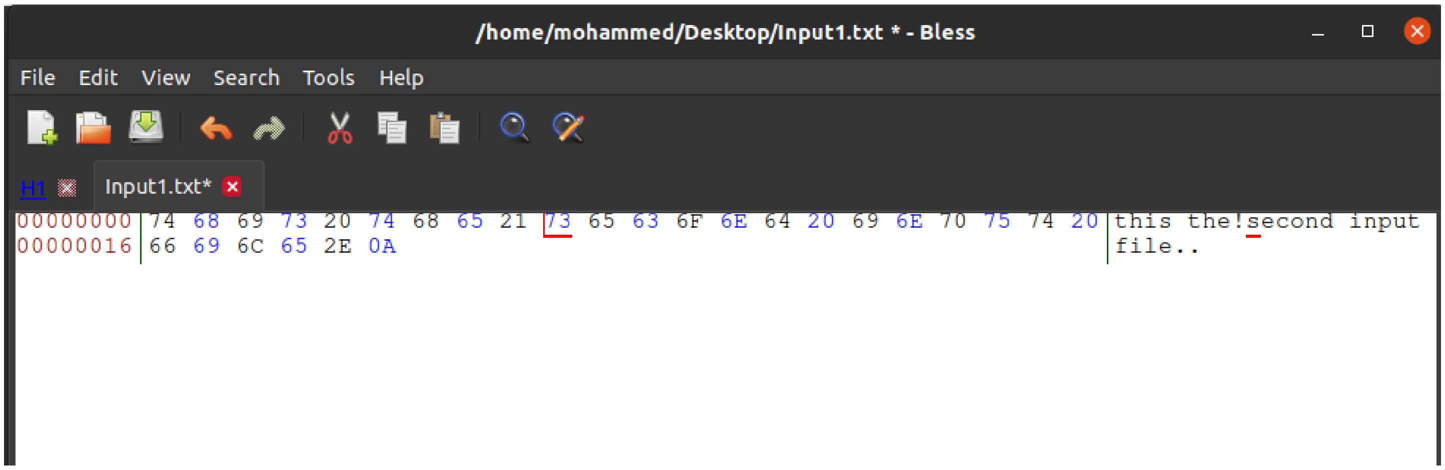
Figure 10: Saved hash value.

3. Then we Flipped one bit of the input file using Bless editor,**20** become **21** , as shown in figures 11 , 12


/home/mohammed/Desktop/Input1.txt - Bless
File Edit View Search Tools Help
H1
Input1.txt

```
00000000 74 68 69 73 20 74 68 65 20 73 65 63 6F 6E 64 20 69 6E 70 75 74 20 this the second input
00000016 66 69 6C 65 2E 0A                                                 file..
```

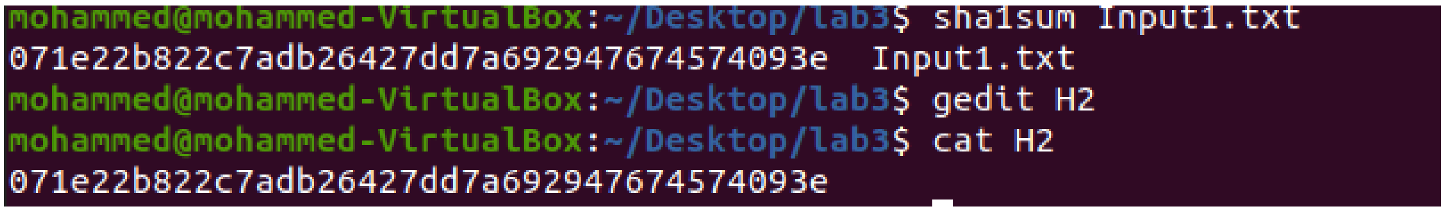
Figure 11: Before flip.


/home/mohammed/Desktop/Input1.txt * - Bless
File Edit View Search Tools Help
H1
Input1.txt*

```
00000000 74 68 69 73 20 74 68 65 21 73 65 63 6F 6E 64 20 69 6E 70 75 74 20 this the!second input
00000016 66 69 6C 65 2E 0A                                                 file..
```

Figure 12: After flip.

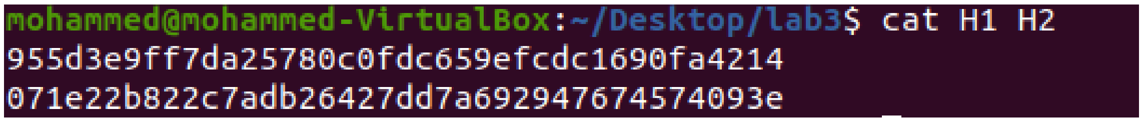
4. After this we Generate a hash code (H2) for the modified text using the same hashing algorithm and we saved it in a file named H2 ,as figure 13 shows.



```
mohammed@mohammed-VirtualBox:~/Desktop/lab3$ sha1sum Input1.txt
071e22b822c7adb26427dd7a692947674574093e  Input1.txt
mohammed@mohammed-VirtualBox:~/Desktop/lab3$ gedit H2
mohammed@mohammed-VirtualBox:~/Desktop/lab3$ cat H2
071e22b822c7adb26427dd7a692947674574093e
```

Figure 13: Hashing of modifying text .

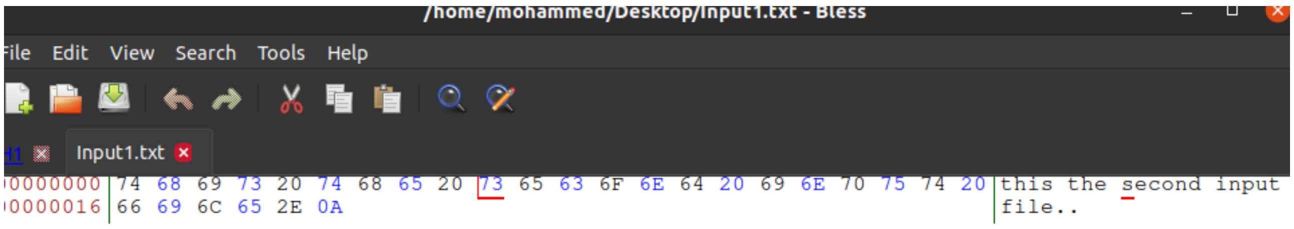
5. After Counting the number of bit differences between the hash codes H1 and H2, we conclude that changing one bit will change all the hash codes, as shown in figure 14



```
mohammed@mohammed-VirtualBox:~/Desktop/lab3$ cat H1 H2
955d3e9ff7da25780c0fdc659efcdc1690fa4214
071e22b822c7adb26427dd7a692947674574093e
```

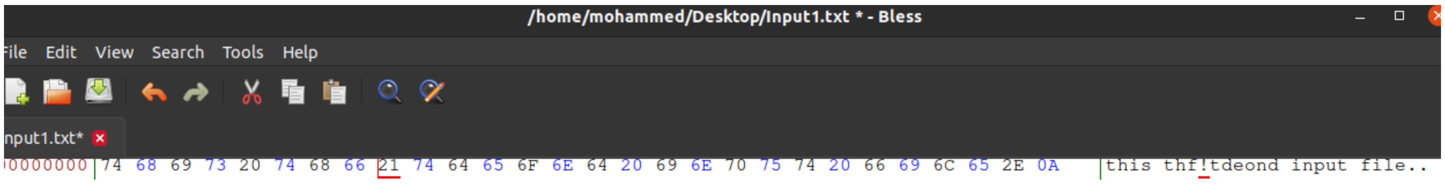
Figure 14: Comparing between H1 and H2.

6. We re-perform the above experiment by changing 5 bits in the original input file , as figures 15,16and 17, We conclude that changing one bit or five bit will give the same effect .


/home/mohammed/Desktop/Input1.txt - Bless
File Edit View Search Tools Help
Input1.txt

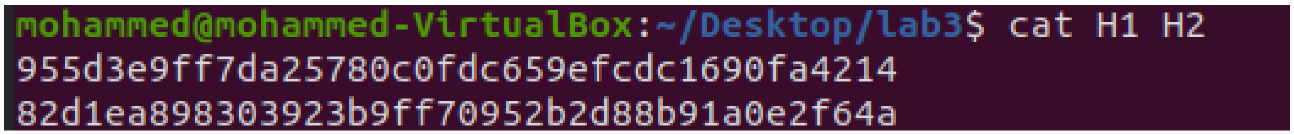
```
00000000 74 68 69 73 20 74 68 65 20 73 65 63 6F 6E 64 20 69 6E 70 75 74 20 this the second input
00000016 66 69 6C 65 2E 0A                                                   file..
```

Figure 15: before flip.


/home/mohammed/Desktop/Input1.txt * - Bless
File Edit View Search Tools Help
nput1.txt*

```
00000000 74 68 69 73 20 74 68 66 21 74 64 65 6F 6E 64 20 69 6E 70 75 74 20 66 69 6C 65 2E 0A    this thf!tdeond input file..
```

Figure 16: after flip.

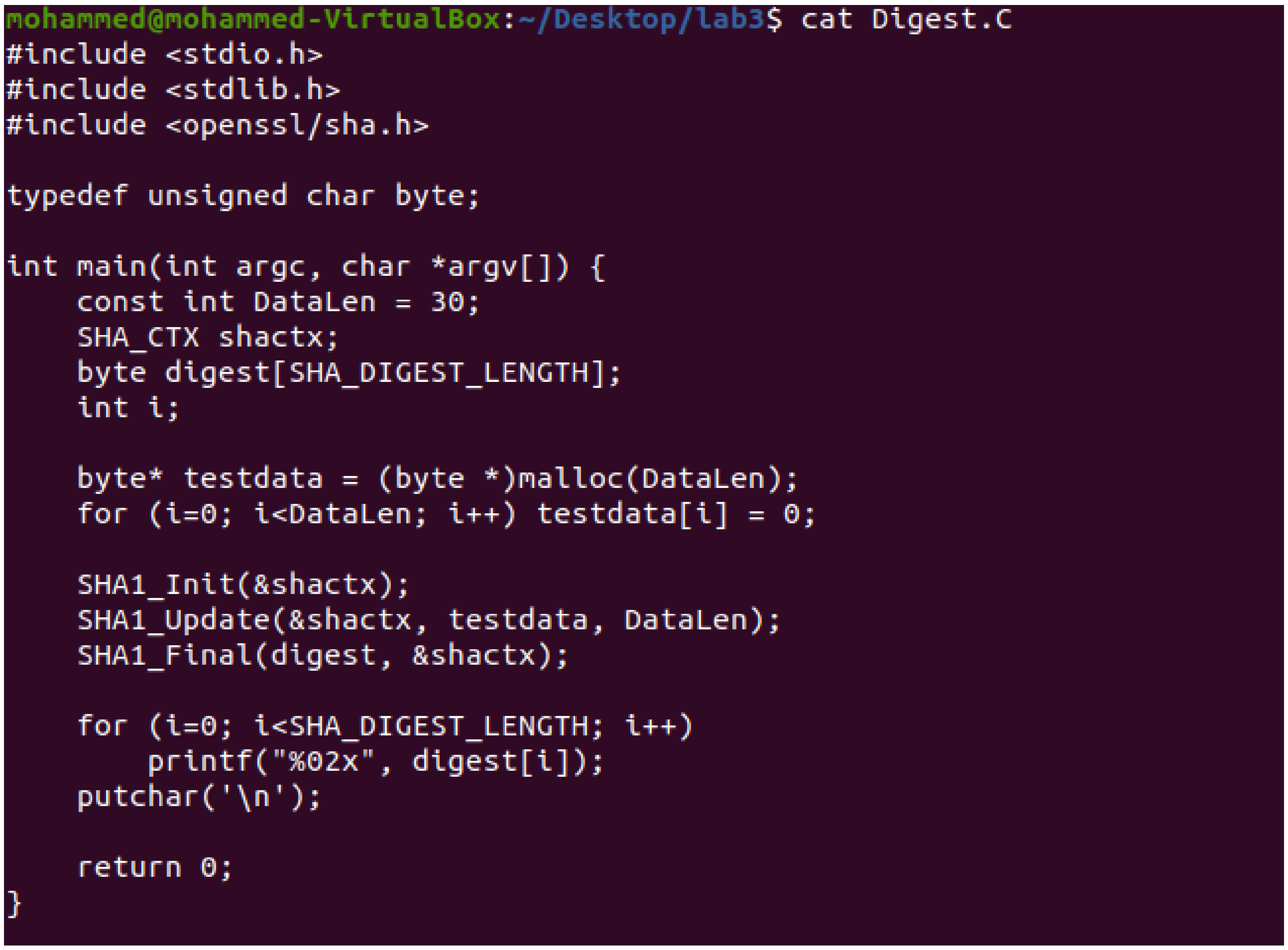


```
mohammed@mohammed-VirtualBox:~/Desktop/lab3$ cat H1 H2
955d3e9ff7da25780c0fdc659efcdc1690fa4214
82d1ea898303923b9ff70952b2d88b91a0e2f64a
```

Figure 17: Comparing.

3.4 C Code Compilation for Digest Generation using SHA

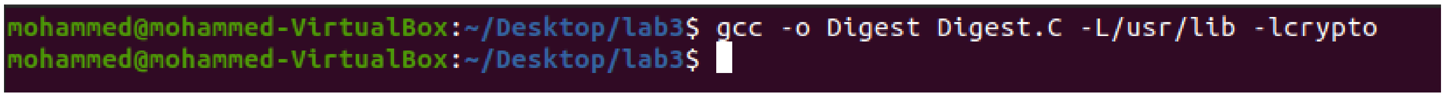
1. We downloaded the C code from moodel, as figure 18 shows.



```
mohammed@mohammed-VirtualBox:~/Desktop/lab3$ cat Digest.C
#include <stdio.h>
#include <stdlib.h>
#include <openssl/sha.h>

typedef unsigned char byte;

int main(int argc, char *argv[]) {
    const int DataLen = 30;
    SHA_CTX shactx;
    byte digest[SHA_DIGEST_LENGTH];
    int i;

    byte* testdata = (byte *)malloc(DataLen);
    for (i=0; i<DataLen; i++) testdata[i] = 0;

    SHA1_Init(&shactx);
    SHA1_Update(&shactx, testdata, DataLen);
    SHA1_Final(digest, &shactx);

    for (i=0; i<SHA_DIGEST_LENGTH; i++)
        printf("%02x", digest[i]);
    putchar('\n');

    return 0;
}
```

Figure 18: C code.

2. After that, we compile that code, as shown in figure 19.

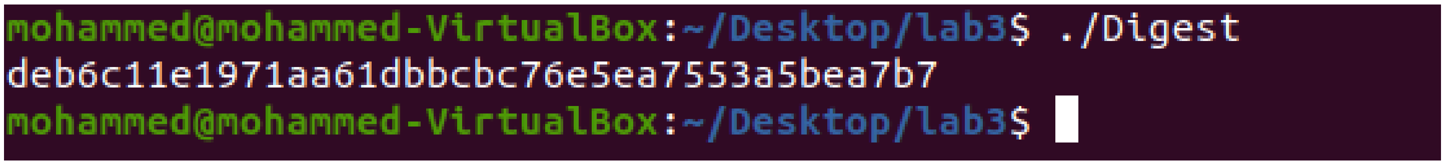


```
mohammed@mohammed-VirtualBox:~/Desktop/lab3$ gcc -o Digest Digest.C -L/usr/lib -lcrypto
mohammed@mohammed-VirtualBox:~/Desktop/lab3$ 
```

Figure 19: Compile C code.

3. We understood that this code will take the message which is all 0, and the length of it is DataLen =30, then it will generate a hash code for this message by sha1 algorithm.

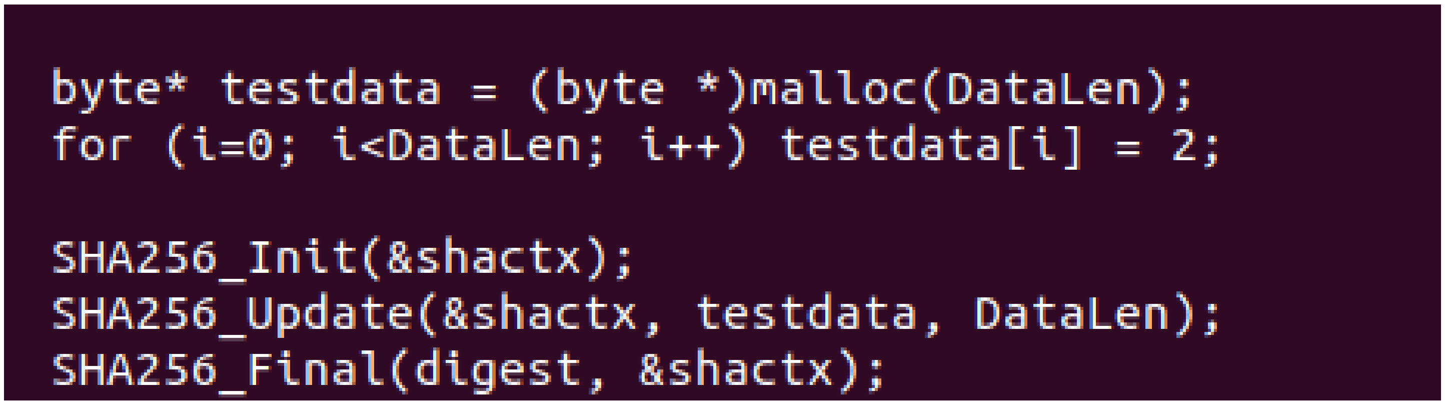
4. We run that code, and as we see in figure 20, that this code generates a hash code.



```
mohammed@mohammed-VirtualBox:~/Desktop/lab3$ ./Digest
deb6c11e1971aa61dbbcbc76e5ea7553a5bea7b7
mohammed@mohammed-VirtualBox:~/Desktop/lab3$
```

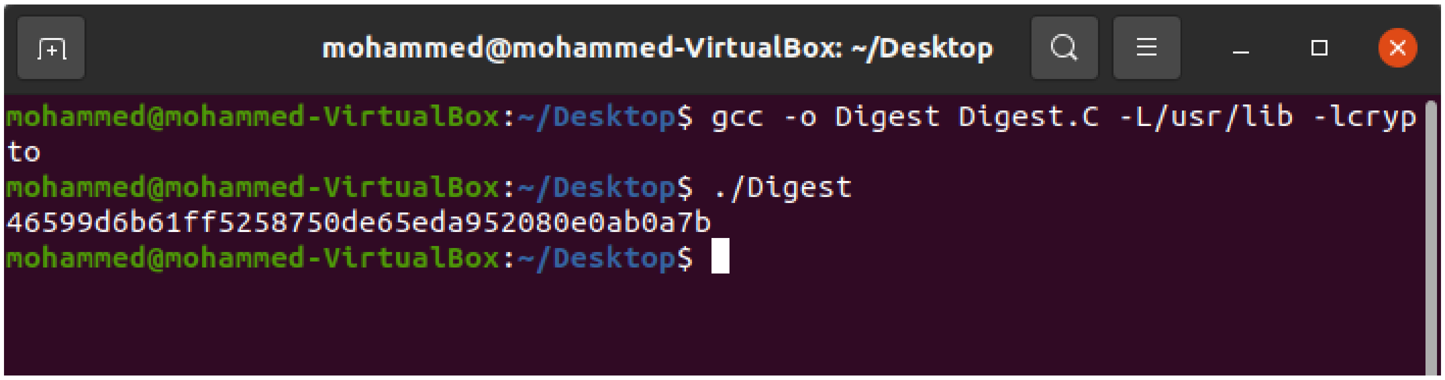
Figure 20: Running C code

5. We change the message code and replace 0 to 2, as figure 21 shows ,Then run it again as figure 22 shows ,we conclude that the hash code will change.



```
byte* testdata = (byte *)malloc(DataLen);
for (i=0; i<DataLen; i++) testdata[i] = 2;

SHA256_Init(&shactx);
SHA256_Update(&shactx, testdata, DataLen);
SHA256_Final(digest, &shactx);
```

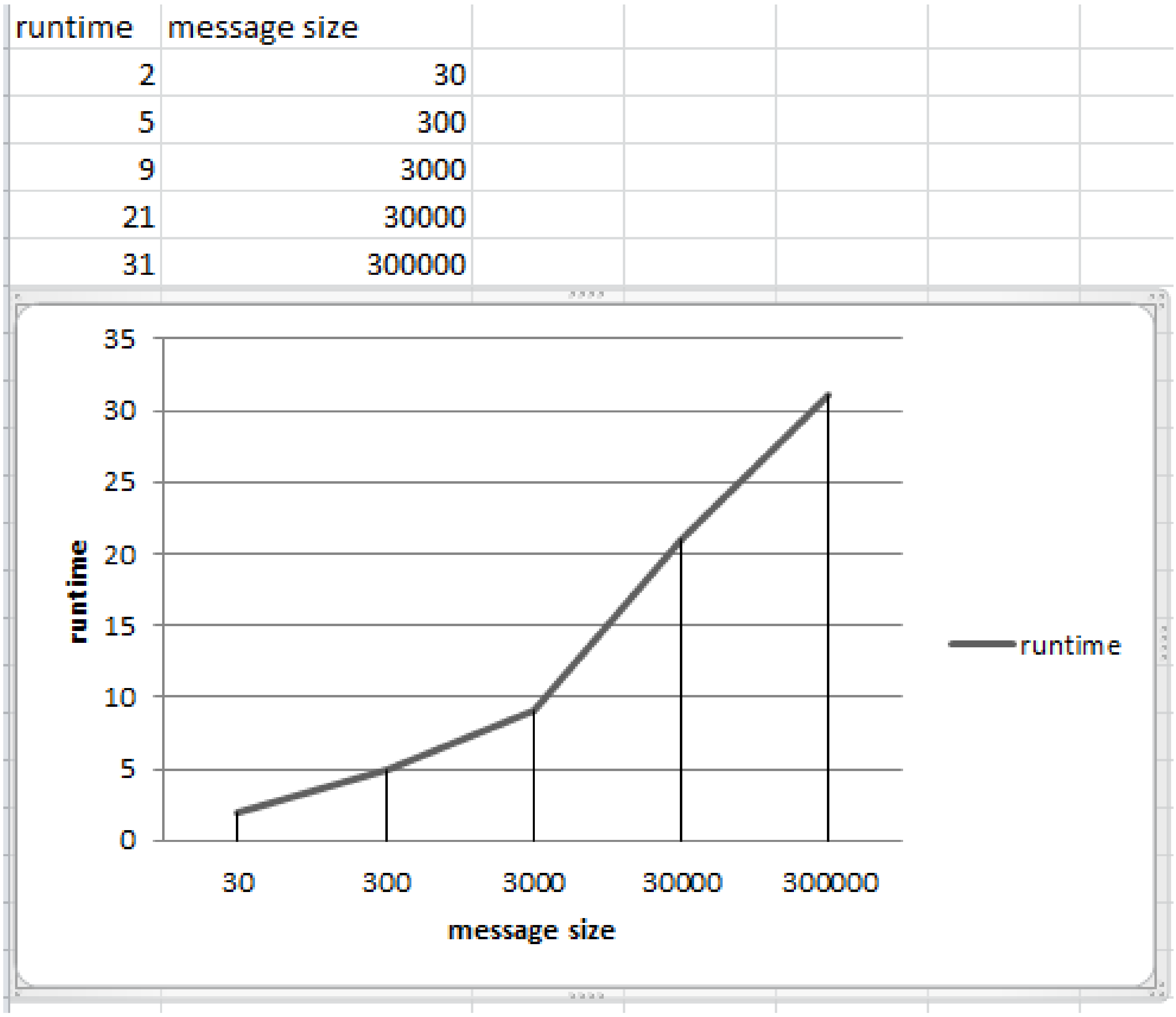
Figure 21: Change message code



```
mohammed@mohammed-VirtualBox:~/Desktop$ gcc -o Digest Digest.C -L/usr/lib -lcryp
to
mohammed@mohammed-VirtualBox:~/Desktop$ ./Digest
46599d6b61ff5258750de65eda952080e0ab0a7b
mohammed@mohammed-VirtualBox:~/Desktop$
```

Figure 22: Run the code

6. We try to run the code many times using different message sizes, then we calculate the run time**(ms)** for each time using **time ./Digest** command. After that, we draw a plot using excel, as figure 23 shows, so we conclude that when the message time increase, the runtime also will be increased.

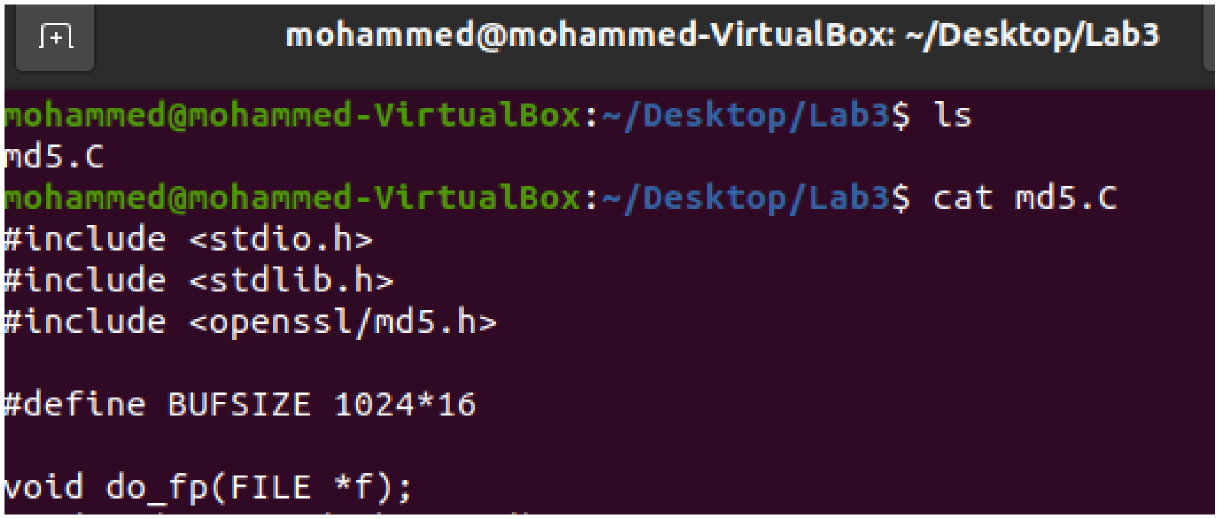


runtime	message size
2	30
5	300
9	3000
21	30000
31	300000

Figure 23: Runtime versus message size.

3.5 C Code Compilation for Digest Generation using MD5

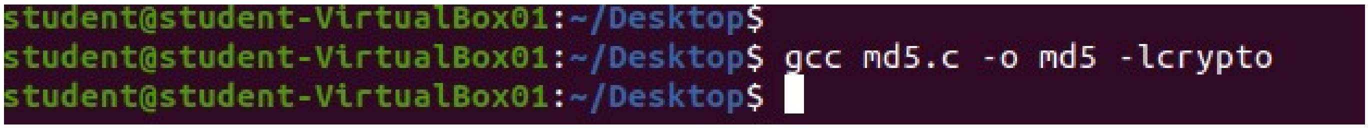
1. We downloaded the C code from moodel, as figure 24 shows.



```
mohammed@mohammed-VirtualBox: ~/Desktop/Lab3
mohammed@mohammed-VirtualBox:~/Desktop/Lab3$ ls
md5.C
mohammed@mohammed-VirtualBox:~/Desktop/Lab3$ cat md5.C
#include <stdio.h>
#include <stdlib.h>
#include <openssl/md5.h>

#define BUFSIZE 1024*16

void do_fp(FILE *f);
```

Figure 24: C code.

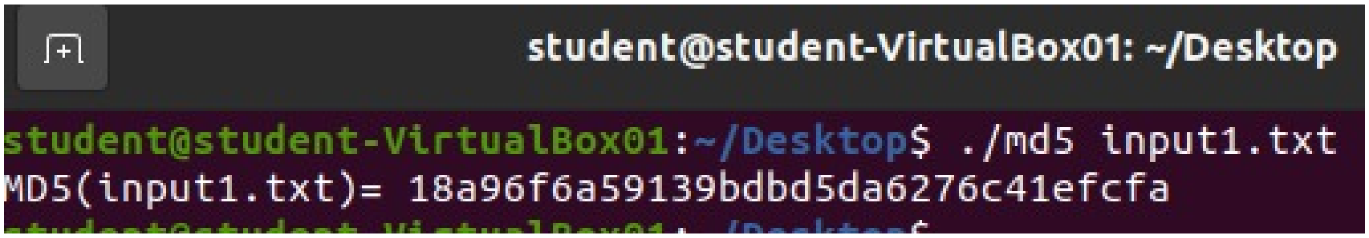
2. After that, we compile that code, as shown in figure 25.



```
student@student-VirtualBox01:~/Desktop$
student@student-VirtualBox01:~/Desktop$ gcc md5.c -o md5 -lcrypto
student@student-VirtualBox01:~/Desktop$
```

Figure 25: Compile C code.

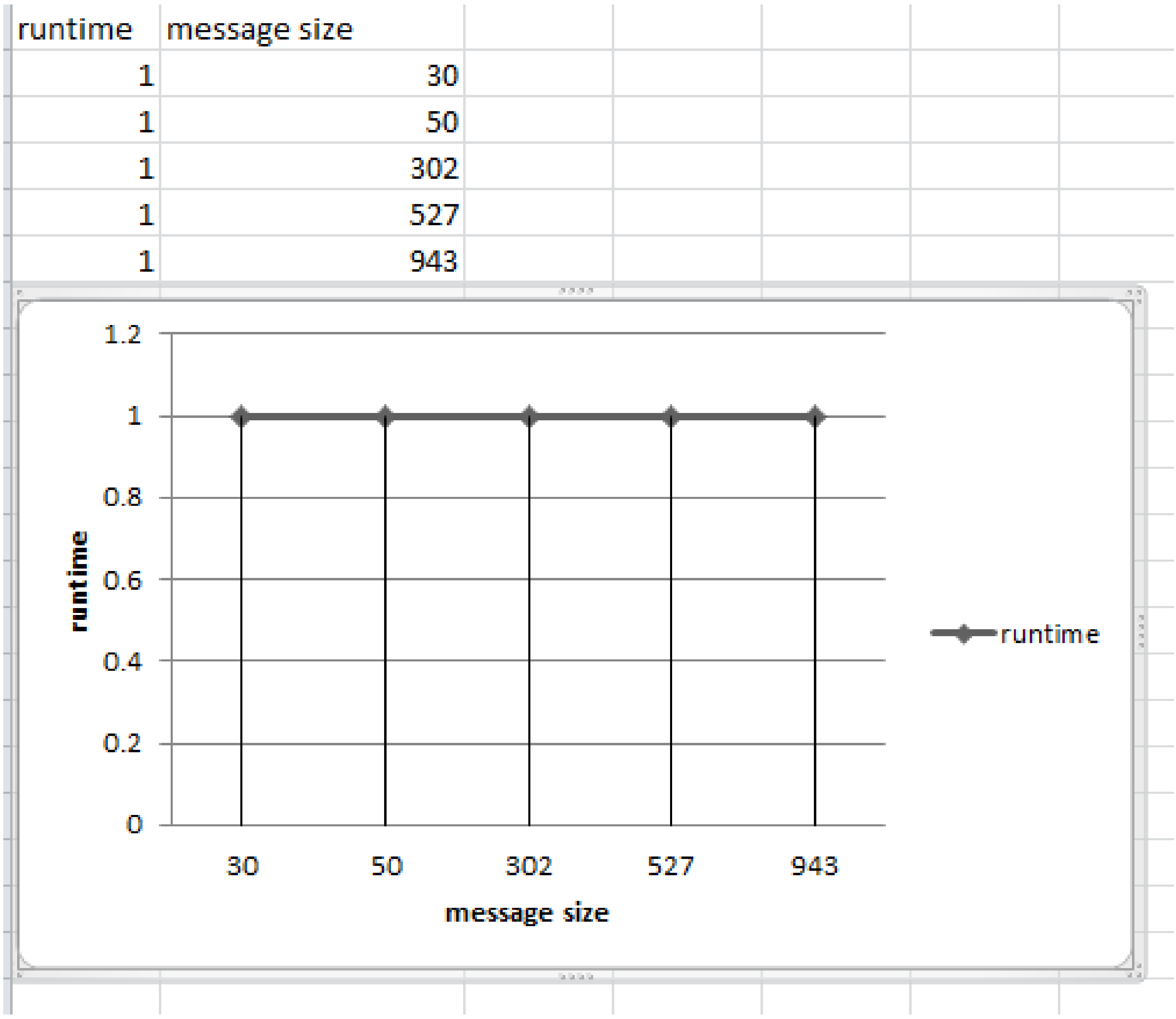
3. We understood that this code will receive an input file as an argument then it will generate hash code for that input using the md5 algorithm.
4. We run that code, and as we see in figure 26, that this code generates a hash code using the md5 algorithm.



```
student@student-VirtualBox01: ~/Desktop
student@student-VirtualBox01:~/Desktop$ ./md5 input1.txt
MD5(input1.txt)= 18a96f6a59139bdbd5da6276c41efcfa
```

Figure 26: Running C code

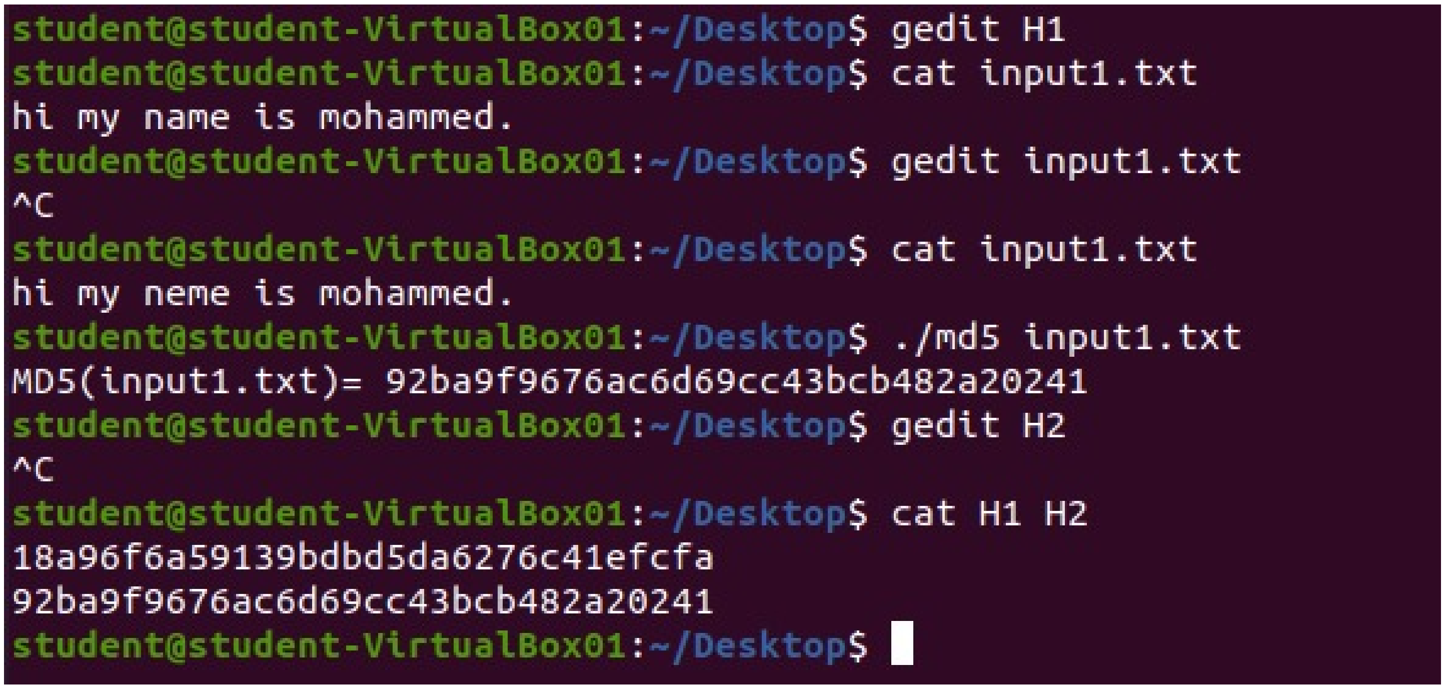
5. We try to run the code many times using different message sizes, then we calculate the run time**(ms)** for each time using **time ./md5 input1.txt** command. After that, we draw a plot using excel, as figure 27 shows, so we conclude that when the message time increase, the run-time doesn't change.



runtime	message size
1	30
1	50
1	302
1	527
1	943

Figure 27: Runtime versus message size.

6. **Question :** Does the MD5 hashing satisfy the Avalanche effect requirement?Does the MD5 hashing satisfy the Avalanche effect requirement? Yes, because as figure28 shows that when changing one input bit, at lest50 percent of the hash code will change.



```
student@student-VirtualBox01:~/Desktop$ gedit H1
student@student-VirtualBox01:~/Desktop$ cat input1.txt
hi my name is mohammed.
student@student-VirtualBox01:~/Desktop$ gedit input1.txt
^C
student@student-VirtualBox01:~/Desktop$ cat input1.txt
hi my neme is mohammed.
student@student-VirtualBox01:~/Desktop$ ./md5 input1.txt
MD5(input1.txt)= 92ba9f9676ac6d69cc43bcb482a20241
student@student-VirtualBox01:~/Desktop$ gedit H2
^C
student@student-VirtualBox01:~/Desktop$ cat H1 H2
18a96f6a59139bdbd5da6276c41efcfa
92ba9f9676ac6d69cc43bcb482a20241
student@student-VirtualBox01:~/Desktop$ 
```

Figure 28: Avalanche effect.

4 Conclusion

In brief , hashing and hash functions are essential tools in computer security. We have learned the objectives of hashing which include data integrity and authentication. We have learned what hashing is, how it works, and went over hash functions in cryptography.

5 References

https://en.wikipedia.org/wiki/Message$_{a}uthentication_{c}ode$
https://crypto.stackexchange.com/questions/10757/difference-between-salted-hash-and-keyed-hashing: :text=Keyed
https://en.wikipedia.org/wiki/Cryptographic$_{h}ash_{f}unction$
https://en.wikipedia.org/wiki/Message$_{a}uthentication_{c}ode : \ : text =$ https://community.cisco.com/t5/networking-documents/how-to-configure-vlans-on-the-catalyst-switches/ta-p/3131780: :text=VLAN