

Overview:

The objective of this lab was to familiarize myself with the concepts and practical applications of hash functions. By completing the tasks, I gained hands-on experience with generating message digests, using keyed hashes like HMAC, and understanding the behavior of one-way hash functions, especially in terms of how sensitive they are to changes in input.

Lab Environment:

For this lab, I used the following tools and environment:

1. **OpenSSL:** I used OpenSSL commands and libraries to perform various hashing tasks. The OpenSSL binaries were already installed on my Kali Linux environment. I also made sure that the necessary header files and manuals were available in case I needed them for programming purposes.
2. **Hex Editor:** I utilized GHex, a hex editor, to view and modify binary files during the lab. GHex allowed me to load data from files and view/edit them in hexadecimal or ASCII format. While some suggested using Bless as an alternative, GHex worked well for my needs.

Analysis:

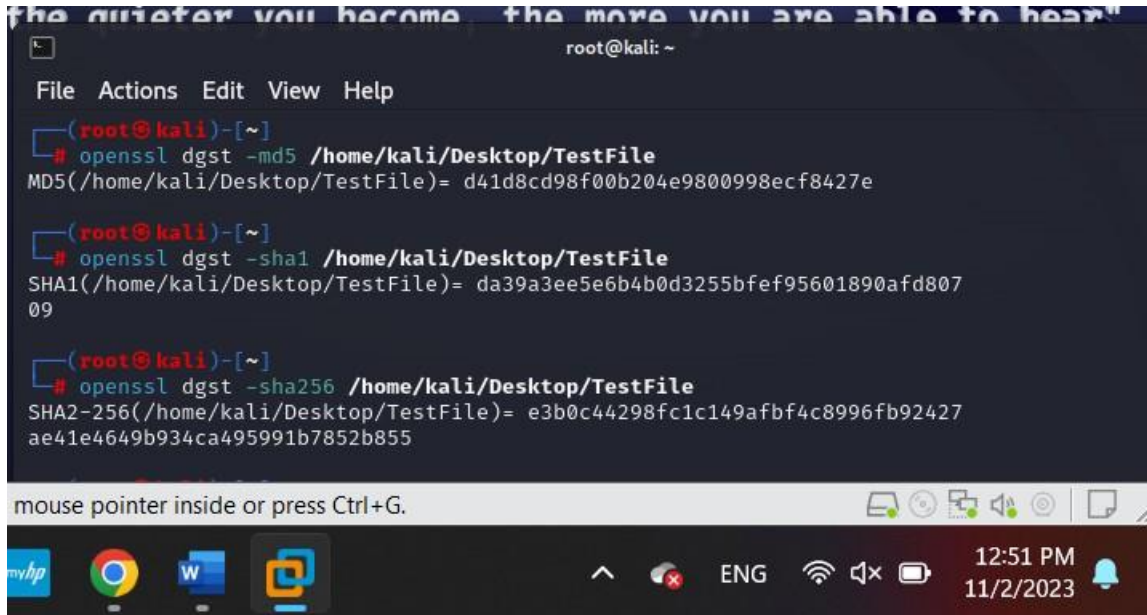
Generating Message Digest and MAC

For this task, I used OpenSSL's `dgst` command to generate hash values for various files using different one-way hash algorithms. The command format I used is:

```
% openssl dgst dgsttype filename
```

Here are the three algorithms I tested:

1. **MD5**: `openssl dgst -md5 testfile.txt`
2. **SHA1**: `openssl dgst -sha1 testfile.txt`
3. **SHA256**: `openssl dgst -sha256 testfile.txt`



```
the quieter you become, the more you are able to hear"
root@kali: ~
File Actions Edit View Help
(root@kali)~[~]
# openssl dgst -md5 /home/kali/Desktop/TestFile
MD5(/home/kali/Desktop/TestFile)= d41d8cd98f00b204e9800998ecf8427e

(root@kali)~[~]
# openssl dgst -sha1 /home/kali/Desktop/TestFile
SHA1(/home/kali/Desktop/TestFile)= da39a3ee5e6b4b0d3255bfef95601890afd807
09

(root@kali)~[~]
# openssl dgst -sha256 /home/kali/Desktop/TestFile
SHA2-256(/home/kali/Desktop/TestFile)= e3b0c44298fc1c149afbf4c8996fb92427
ae41e4649b934ca495991b7852b855

mouse pointer inside or press Ctrl+G.
```

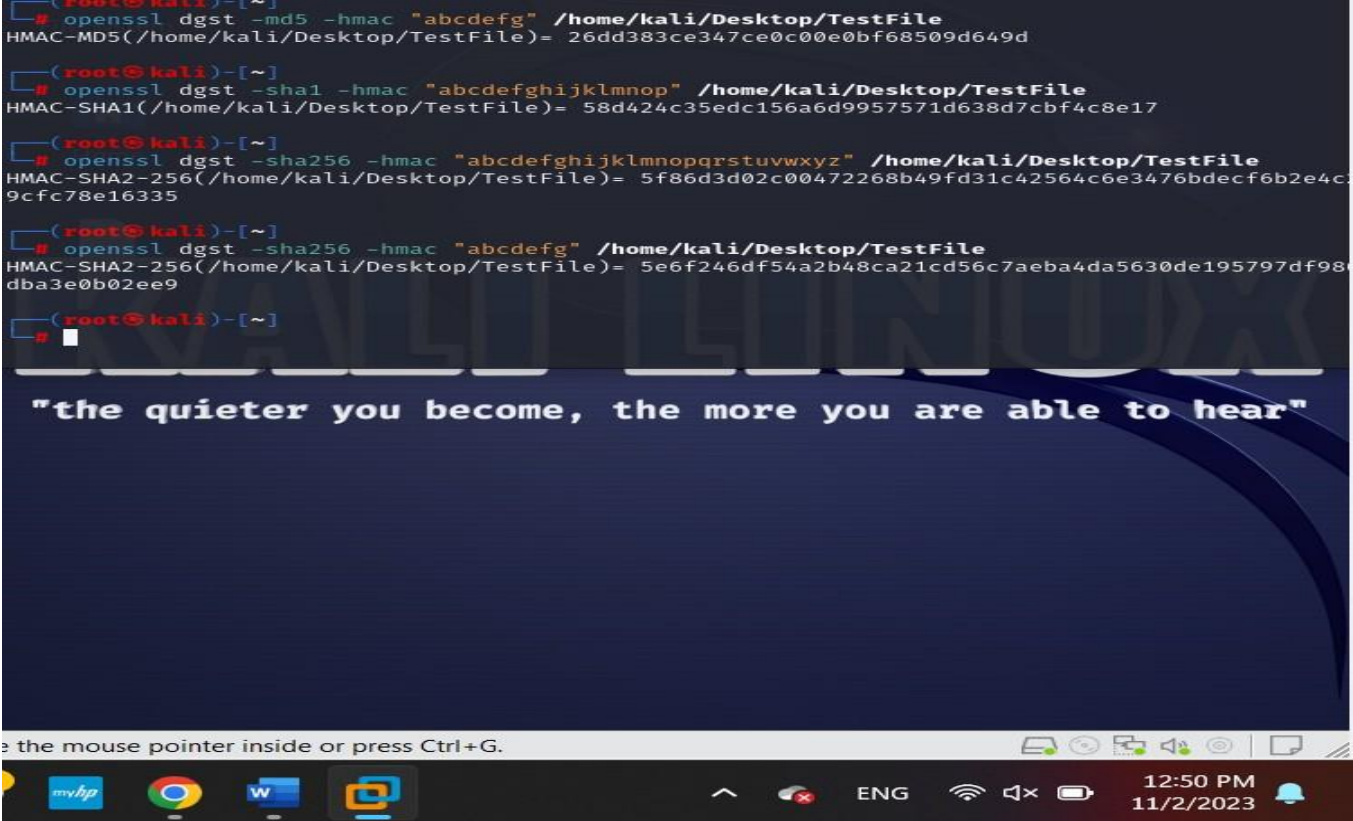
I observed that each algorithm produced hash values of different lengths. The MD5 hash was the shortest, while SHA256 produced the longest hash. Despite these differences, all hashes seemed to consistently represent the content of the file.

Keyed Hash and HMAC

For this task, I generated a keyed hash (MAC) for a file using the `-hmac` option in OpenSSL. I tested several algorithms by specifying different keys and hash types. Below is an example command to generate a keyed hash using HMAC-MD5:

```
% openssl dgst -md5 -hmac "abcdefg" filename
```

I also tried HMAC-SHA256 and HMAC-SHA1 with the same file, testing different key lengths. From my observations, I found that the length of the key didn't seem to affect the output hash, as the hash output was always of a fixed length for each algorithm. The key size didn't seem to be fixed, as varying the key length didn't change the hash length, but it did affect the actual hash value.



```
(root@kali)~# openssl dgst -md5 -hmac "abcdefg" /home/kali/Desktop/TestFile
HMAC-MD5(/home/kali/Desktop/TestFile)= 26dd383ce347ce0c00e0bf68509d649d

(root@kali)~# openssl dgst -sha1 -hmac "abcdefghijklmnop" /home/kali/Desktop/TestFile
HMAC-SHA1(/home/kali/Desktop/TestFile)= 58d424c35edc156a6d9957571d638d7cbf4c8e17

(root@kali)~# openssl dgst -sha256 -hmac "abcdefghijklmnopqrstuvwxyz" /home/kali/Desktop/TestFile
HMAC-SHA2-256(/home/kali/Desktop/TestFile)= 5f86d3d02c00472268b49fd31c42564c6e3476bdecf6b2e4c9cfc78e16335

(root@kali)~# openssl dgst -sha256 -hmac "abcdefg" /home/kali/Desktop/TestFile
HMAC-SHA2-256(/home/kali/Desktop/TestFile)= 5e6f246df54a2b48ca21cd56c7aeba4da5630de195797df98db3e0b02ee9

(root@kali)~#
```

"the quieter you become, the more you are able to hear"

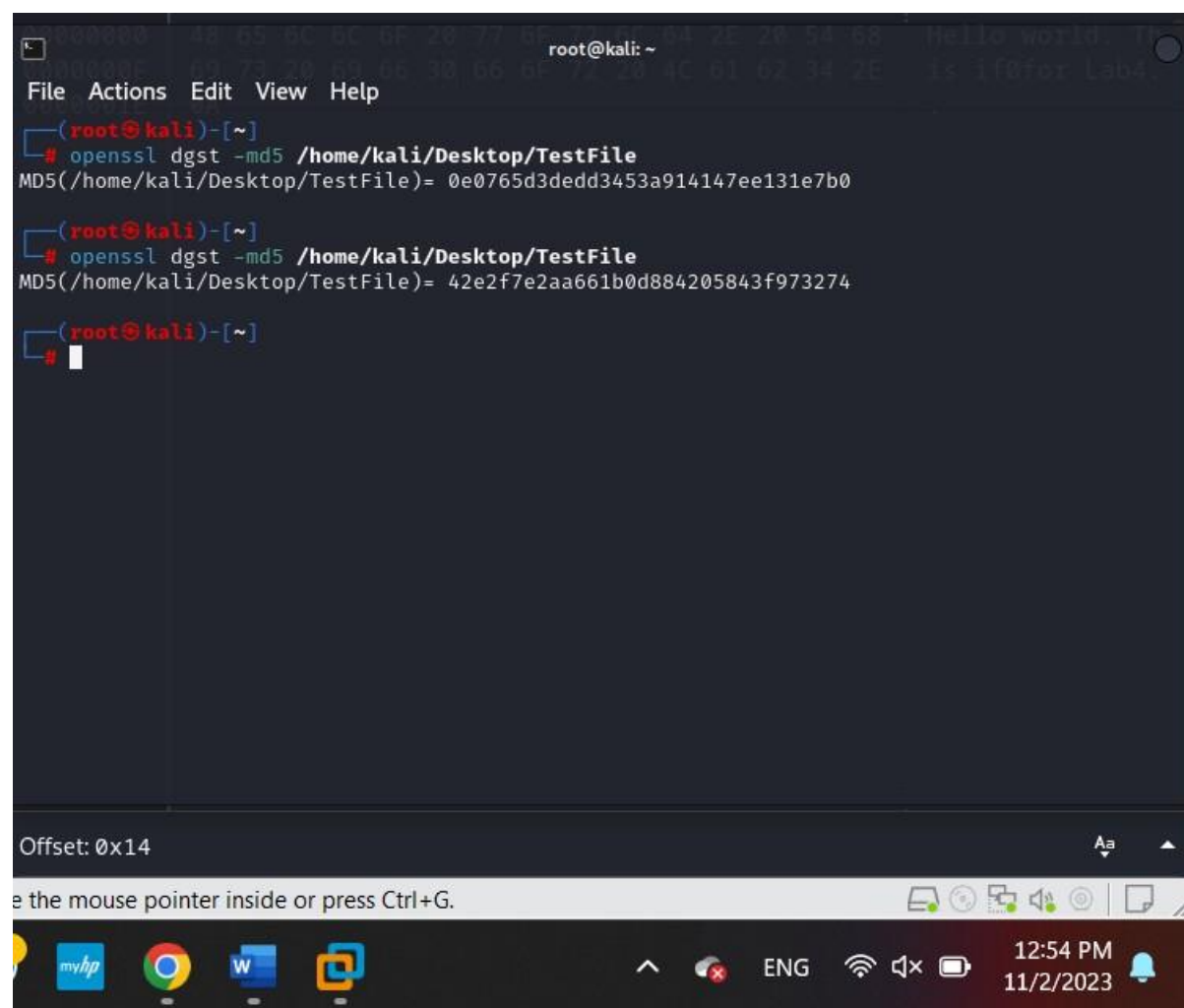
the mouse pointer inside or press Ctrl+G.

my hp | Chrome | W | | ^ | ENG | 12:50 PM 11/2/2023

The Randomness of One-way Hash

To understand the properties of one-way hash functions, I performed the following steps using MD5 and SHA256:

1. Created a text file.
2. Generated the hash value **H1** for the file using a specific hash algorithm (MD5 or SHA256).
3. Flipped one bit of the input file using GHex.
4. Generated the hash value **H2** for the modified file.

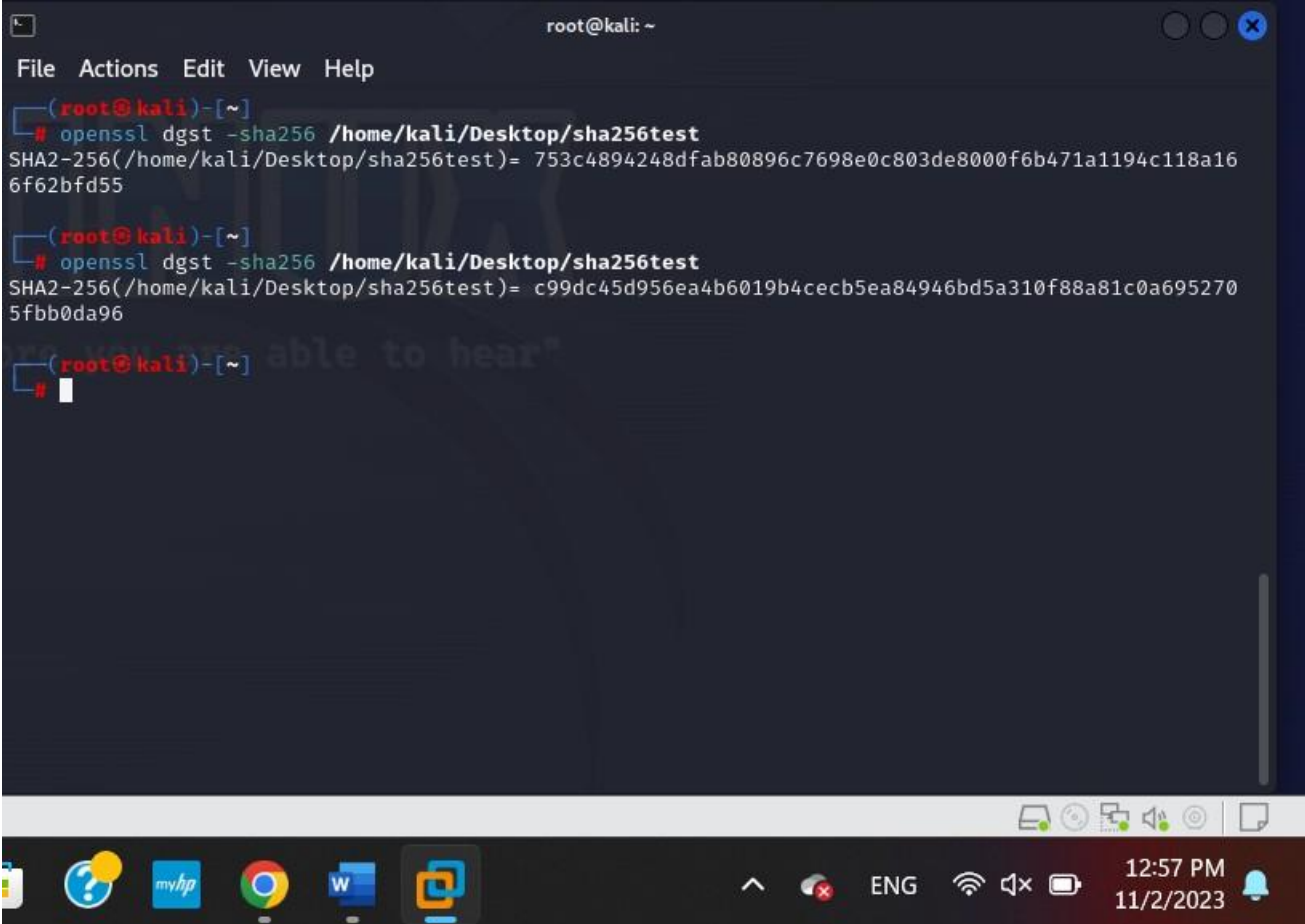


The screenshot shows a terminal window with a dark background. At the top, the prompt is 'root@kali: ~'. Below the prompt, there is a menu bar with 'File', 'Actions', 'Edit', 'View', and 'Help'. The terminal displays two commands and their outputs:

```
(root@kali)-[~]  
# openssl dgst -md5 /home/kali/Desktop/TestFile  
MD5(/home/kali/Desktop/TestFile)= 0e0765d3dedd3453a914147ee131e7b0  
  
(root@kali)-[~]  
# openssl dgst -md5 /home/kali/Desktop/TestFile  
MD5(/home/kali/Desktop/TestFile)= 42e2f7e2aa661b0d884205843f973274  
  
(root@kali)-[~]  
#
```

At the bottom of the terminal window, there is a status bar showing 'Offset: 0x14' and a search icon. Below the terminal window, there is a taskbar with icons for 'myhp', 'Chrome', 'Word', and 'Gnome'. The system tray at the bottom right shows the time '12:54 PM' and the date '11/2/2023'.

I observed that even with a single bit change in the input file, the resulting hash values (H1 and H2) were completely different. For both MD5 and SHA256, the outputs were vastly different, even though the input change was minimal. This demonstrated the sensitivity of hash functions and reinforced their value in ensuring data integrity.



```
root@kali: ~  
File Actions Edit View Help  
(root@kali)-[~]  
# openssl dgst -sha256 /home/kali/Desktop/sha256test  
SHA2-256(/home/kali/Desktop/sha256test)= 753c4894248dfab80896c7698e0c803de8000f6b471a1194c118a16  
6f62bfd55  
(root@kali)-[~]  
# openssl dgst -sha256 /home/kali/Desktop/sha256test  
SHA2-256(/home/kali/Desktop/sha256test)= c99dc45d956ea4b6019b4cecb5ea84946bd5a310f88a81c0a695270  
5fbb0da96  
(root@kali)-[~]  
#
```