

## Lab-4

### Task 1:

Write a program to implement MD4 and MD5 algorithm using library functions.

### Theory:

MD4 and MD5 are cryptographic hash functions designed by Rivest. Several hash functions have been influenced by their design. Practical attacks exist for MD4 and MD5, with high impact on commonly used applications.

MD4 and MD5 are the initial members of the MD4 type hash functions. Both were designed by Rivest. They take variable length input messages and hash them to fixed-length outputs. Both operate on 512-bit message blocks divided into 32-bit words and produce a message digest of 128 bits. First, the message is padded according to the so-called Merkle-Damgård strengthening technique. Next, the message is processed block by block by the underlying compression function. This function initializes four 32-bit chaining variables to a fixed value prior to hashing the first message block, and to the current hash value for the following message blocks. Each step of the compression function updates in turn one of the chaining variables according to one message word. Both compression functions are organized into rounds of 16 steps each. MD4 has three such rounds, while MD5 consists of 4 rounds

### Source Code:

```
## MD4 implementation using python

from Crypto.Hash import MD4

text = 'Hello There'
hashObject = MD4.new(text.encode('utf-8'))
digest = hashObject.hexdigest()

print("The hexadecimal equivalent of hash using MD4 is : ", end = "")
print(digest)
```

```
# Python 3 code to demonstrate the working of MD5

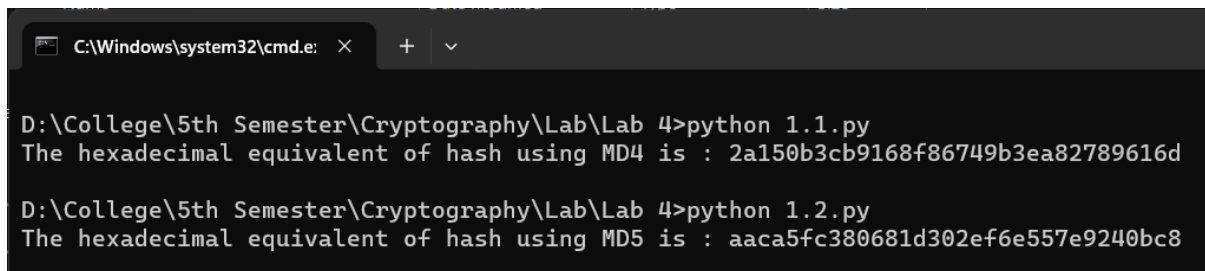
import hashlib

# initializing string
str2hash = "Master Kenobi"

result = hashlib.md5(str2hash.encode())

print("The hexadecimal equivalent of hash using MD5 is : ", end = "")
print(result.hexdigest())
```

Output:



```
C:\Windows\system32\cmd.exe
D:\College\5th Semester\Cryptography\Lab\Lab 4>python 1.1.py
The hexadecimal equivalent of hash using MD4 is : 2a150b3cb9168f86749b3ea82789616d

D:\College\5th Semester\Cryptography\Lab\Lab 4>python 1.2.py
The hexadecimal equivalent of hash using MD5 is : aaca5fc380681d302ef6e557e9240bc8
```

Conclusion:

Hence, in this way we can implement use MD4 and MD5 hash algorithm in the laboratory.

## Task 2:

Write a program to implement SHA-1 and SHA-2 algorithm using library functions.

### Theory:

SHA1 is a cryptographic hash function which is designed by United States National Security Agency. It takes an input and produces a 160 bits hash value. Further the output produced by this function is converted into a 40 digits long hexadecimal number. It is a U.S. Federal Information Processing Standard. It was first published in 1995. It is successor to SH0 published in 1993.

SHA1 is also a cryptographic hash function which is designed by United States National Security Agency. It is constructed using the Merkle-Damgard structure from a one-way compression function. The compression function used is constructed using the Davies-Meyer structure from a classified block cipher. It was first published in 2001. It is successor to SH1.

### Source Code:

```
## encryption using SHA-1 in python

import hashlib

text = 'Hello There'
result = hashlib.sha1(text.encode())

print("The hexadecimal equivalent of hash using SHA-1 is : ", end = "")
print(result.hexdigest())

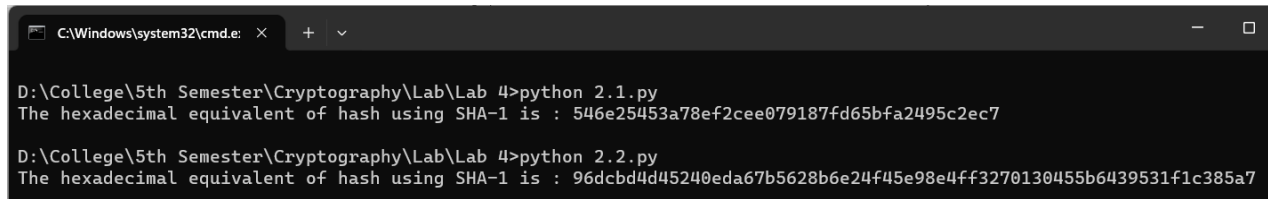
## encryption using SHA-2 (256) in python

import hashlib

text = 'Master Kenobi'
result = hashlib.sha256(text.encode())

print("The hexadecimal equivalent of hash using SHA-1 is : ", end = "")
print(result.hexdigest())
```

### Output:



```
C:\Windows\system32\cmd.e: X + v

D:\College\5th Semester\Cryptography\Lab\Lab 4>python 2.1.py
The hexadecimal equivalent of hash using SHA-1 is : 546e25453a78ef2cee079187fd65bfa2495c2ec7

D:\College\5th Semester\Cryptography\Lab\Lab 4>python 2.2.py
The hexadecimal equivalent of hash using SHA-1 is : 96dcbd4d45240eda67b5628b6e24f45e98e4ff3270130455b6439531f1c385a7
```

### Conclusion:

Hence, in this way we can implement use SHA1 and SHA2 hash algorithm in the laboratory.

**Task 3:**

Download SSL (Digital Certificate) of a website and analyze its content.

**Theory:**

An SSL certificate is a Digital certificate that can be used for authentication of a website, and it creates a secure connection between client and web server. When a certificate is installed, it makes the website from HTTP to HTTPS.

Working process of SSL certificate :

The SSL certificate enables the encryption of data which is then sent to the server-side. It has two keys one is public and the other one is a private key. Data encrypted with the public key can be decrypted with a private key only. The web server with a private key can understand the data. If data packets are stolen from in between those are useless because they are encrypted

**Conclusion:**

Hence, in this way we can observe and analyze the SSL of a website.

#### Task 4:

Write a malicious logic code program that performs some malicious code.

#### Theory:

Malicious Logic is hardware, firmware, or software that is intentionally included or inserted in a system to perform an unauthorized function or process that will have adverse impact on the confidentiality, integrity, or availability of an information system.

Malware is a software that gets into the system without user consent with an intention to steal private and confidential data of the user that includes bank details and password. They also generates annoying pop up ads and makes changes in system settings

They get into the system through various means:

1. Along with free downloads.
2. Clicking on suspicious link.
3. Opening mails from malicious source.
4. Visiting malicious websites.
5. Not installing an updated version of antivirus in the system.

#### Types:

1. Virus
2. Worm
3. Logic Bomb
4. Trojan/Backdoor
5. Rootkit
6. Advanced Persistent Threat
7. Spyware and Adware

#### Source Code:

```
#include <iostream>
#include <fstream>
#include <string>

using namespace std;

int main()
{
    ofstream fout;

    string line = "aaa";
```

```

string txt = ".txt";
// by default ios::out mode, automatically deletes
// the content of file. To append the content, open in ios::app
// fout.open("sample.txt", ios::app)
for (int i = 0; i <= 10; i++) {
    string name = to_string(i);
    string fname = name + txt;

    // Open the file outside the loop
    fout.open(fname);














    // Execute a loop if the file is successfully opened
    while (fout) {
        for (int j = 0; j <= 10000; j++) {
            // Generate large text content, you can modify this as
needed
            fout << "This is line " << j << " in file " << i << endl;
        }

        // Write line in the file
        //fout << line << endl;
        fout.close();
    }
}
}

```

### Output:

On Running the above C code, it generates 10 files with each having 10000 lines of text amounting to each file of around 300 kb. On modifying the number of iterations of the file generation in the outer loop of 'i', the number of files being generated can be modified.

Name	Date modified	Type	Size
 0	1/11/2024 10:43 AM	Text Document	283 KB
 1	1/11/2024 10:43 AM	Text Document	283 KB
 2	1/11/2024 10:43 AM	Text Document	283 KB
 3	1/11/2024 10:43 AM	Text Document	283 KB
 4	1/9/2024 8:19 AM	C++ Source File	1 KB
 4	1/11/2024 10:43 AM	Application	1,885 KB
 4	1/11/2024 10:43 AM	Text Document	283 KB
 5	1/11/2024 10:43 AM	Text Document	283 KB
 6	1/11/2024 10:43 AM	Text Document	283 KB
 7	1/11/2024 10:43 AM	Text Document	283 KB
 8	1/11/2024 10:43 AM	Text Document	283 KB
 9	1/11/2024 10:43 AM	Text Document	283 KB
 10	1/11/2024 10:43 AM	Text Document	292 KB

### Conclusion:

Hence, in this way we can create and see the effect of a simple malicious logic in the laboratory.



