Terna Engineering College

Computer Engineering Department

Class: TE **Sem.: VI**

Course: System Security Lab

PART A

(PART A: TO BE REFERRED BY STUDENTS)

Experiment No.03

A.1 Aim:

For varying message sizes, test the integrity of the message using MD-5, SHA-1, and analyze the performance of the two protocols. Use crypt APIs.

A.2 Prerequisite:

Basic Knowledge of MD5 and SHA 1.

A.3 Outcome:

After the successful completion of this experiment, students will be able to analyze and evaluate the performance of hashing algorithms.

A.4 Theory:

MD5 (Message-Digest algorithm 5) is a widely used cryptographic hash function with a 128-bit hash value. An MD5 hash is typically expressed as a 32 digit hexadecimal number. MD5 processes a variable-length message into a fixed-length output of 128 bits. The input message is broken up into chunks of 512-bit blocks (sixteen 32bit little-endian integers) ; The message is padded so that its length is divisible by 512. The padding works as follows: first, a single bit, 1, is appended to the end of the message. This is followed by as many zeros as are required to bring the length of the message up to 64 bits less than a multiple of 512. The remaining bits are filled up with a 64bit integer representing the length of the original message, in bits.

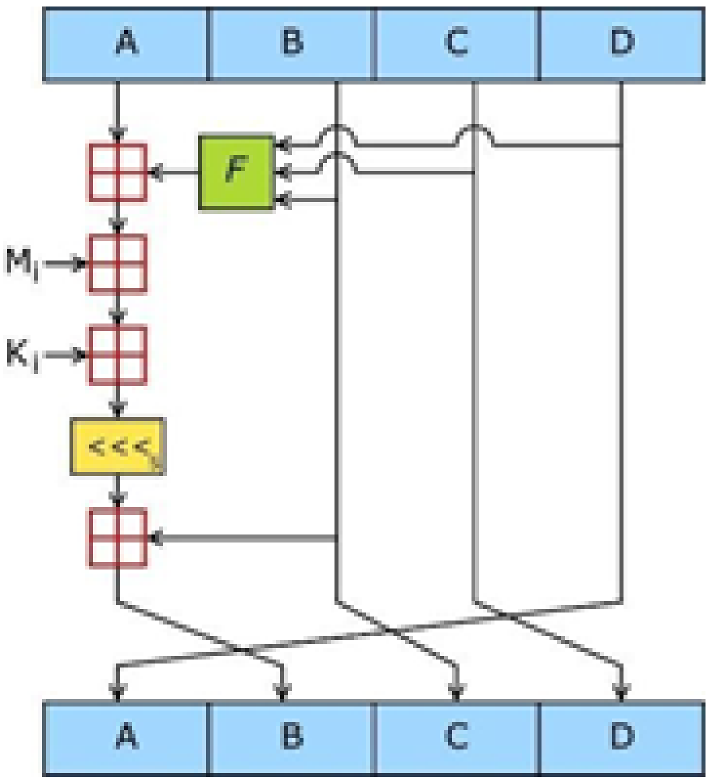


Figure 1: One MD5 operation.

MD5 consists of 64 of these operations, grouped in four rounds of 16 operations. F is a nonlinear function; one function is used in each round. Mi denotes a 32bit block of the message input and Denotes a 32bit constant, different for each operation.

The main MD5 algorithm operates on a 128bit state, divided into four 32bit words, denoted A, B, C and D. These are initialized to certain fixed constants. The main algorithm then operates on each 512bit message block, in turn, each block modifying the state. The processing of a message block consists of our similar stages, termed rounds; each round is composed of 16 similar operations based on a nonlinear function F, modular addition, and left rotation.

Figure 1 illustrates one operation within one round. There are four possible functions F; a different one is used in each round:

$$F(X,Y,Z) = (X \wedge Y) \vee (\neg X \wedge Z)$$
$$G(X,Y,Z) = (X \wedge Z) \vee (Y \wedge \neg Z)$$
$$H(X,Y,Z) = X \oplus Y \oplus Z$$
$$I(X,Y,Z) = Y \oplus (X \vee \neg Z)$$

$\oplus$, $\wedge$, $\vee$, $\neg$ denote the XOR, AND, OR and NOT operations respectively.

Algorithm:

1. **Append Padding Bits**

The message is "padded" (extended) so that its length (in bits) is congruent to 448, modulo512. That is, the message is extended so that it is just 64 bits shy of being a multiple of 512 bits long. Padding is always performed, even if the length of the message is already congruent to 448, modulo512. Padding is performed as follows: a single "1" bit is appended to the message, and then "0" bits are appended so that the length in bits of the padded message becomes congruent to 448, modulo 512. In all, at least one bit and at most 512 bits are appended.

2. **Append Length**

A 64-bit representation of b (the length of the message before the padding bits were added) is appended to the result of the previous step. In the unlikely event that b is greater than 2^64, then only the low order 64 bits of b are used. (These bits are appended as two 32bit words and appended low- order word first following the previous conventions.) At this point, the resulting message (after padding with bits and with b) has a length that is an exact multiple of 512 bits. Equivalently, this message has a length that is an exact multiple of 16 (32 bit) words. Let M[0 ... N1] denote the words of the resulting message, where N is a multiple of16.

3. **Initialize MD Buffer**

A four-word buffer (A, B, C, D) is used to compute the message digest. Here each of A, B, C, D is a 32bit register. These registers are initialized to the following values in hexadecimal, low-order bytes first):

4. **Process Message in 16 Word Blocks**

We first define four auxiliary functions that each take an input of three 32 bit word sand produce as output one 32 bit word.

Note: Don't write code for MD5 or SHA1. Analyze the performance using crypt APIs.

PART B

(PART B: TO BE COMPLETED BY STUDENTS)

(Students must submit the soft copy as per the following segments within two hours of the practical. The soft copy must be uploaded on the Blackboard or emailed to the concerned lab in charge faculties at the end of the practical in case there is no Blackboard access available)

Roll No. 50	**Name:** AMEY THAKUR
Class: Comps TE B	**Batch:** B3
Date of Experiment: 18/02/2021	**Date of Submission:** 18/02/2021
Grade:	

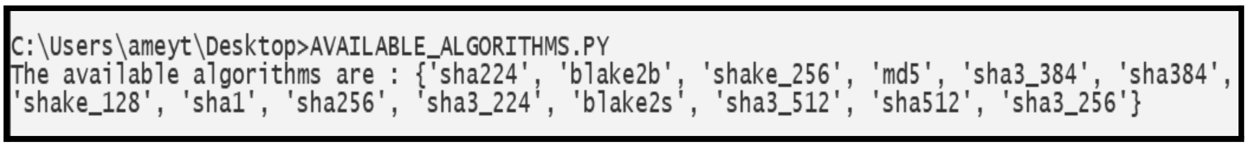
B.1 Output / Observations:

- **Available Algorithms in hashlib**

INPUT:

```
import hashlib

# prints all available algorithms
print ("The available algorithms are : ", end ="")
print (hashlib.algorithms_guaranteed)
```

OUTPUT:



```
C:\Users\ameyt\Desktop>AVAILABLE_ALGORITHMS.PY
The available algorithms are : {'sha224', 'blake2b', 'shake_256', 'md5', 'sha3_384', 'sha384',
'shake_128', 'sha1', 'sha256', 'sha3_224', 'blake2s', 'sha3_512', 'sha512', 'sha3_256'}
```

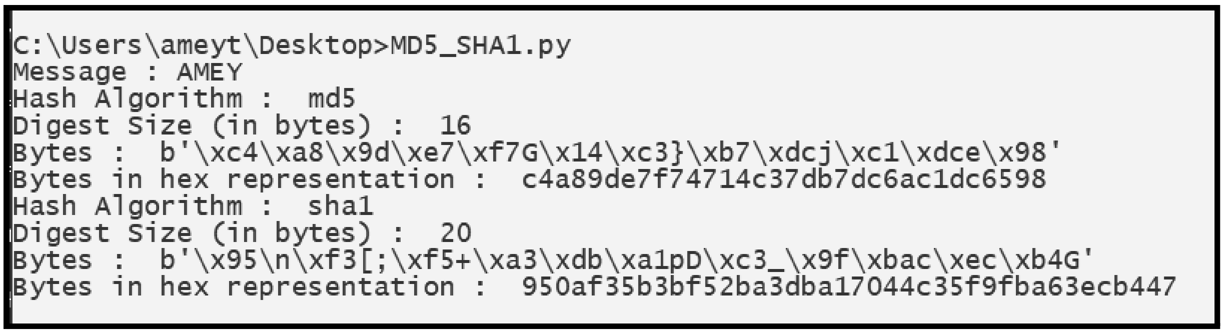
- **MD5 and SHA1 Hashing**

INPUT:

```
import hashlib

print("Message : AMEY")

result = hashlib.md5(b"AMEY").hexdigest()

result = hashlib.md5("AMEY".encode("utf-8")).hexdigest()

m = hashlib.md5(b"AMEY")
```

```
print("Hash Algorithm : ",m.name)
print("Digest Size (in bytes) : ",m.digest_size)
print("Bytes : ",m.digest())    # bytes
print("Bytes in hex representation : ",m.hexdigest()) # bytes in hex representation

result = hashlib.sha1(b"AMEY").hexdigest()

result = hashlib.sha1("AMEY".encode("utf-8")).hexdigest()

m = hashlib.sha1(b"AMEY")

print("Hash Algorithm : ",m.name)
print("Digest Size (in bytes) : ",m.digest_size)
print("Bytes : ",m.digest())    # bytes
print("Bytes in hex representation : ",m.hexdigest()) # bytes in hex representation
```

OUTPUT:



```
C:\Users\ameyt\Desktop>MD5_SHA1.py
Message : AMEY
Hash Algorithm :  md5
Digest Size (in bytes) :  16
Bytes :  b'\xc4\xa8\x9d\xe7\xf7G\x14\xc3}\xb7\xdcj\xc1\xdce\x98'
Bytes in hex representation :  c4a89de7f74714c37db7dc6ac1dc6598
Hash Algorithm :  sha1
Digest Size (in bytes) :  20
Bytes :  b'\x95\n\xf3[;\xf5+\xa3\xdb\xa1pD\xc3_\x9f\xbac\xec\xb4G'
Bytes in hex representation :  950af35b3bf52ba3dba17044c35f9fba63ecb447
```

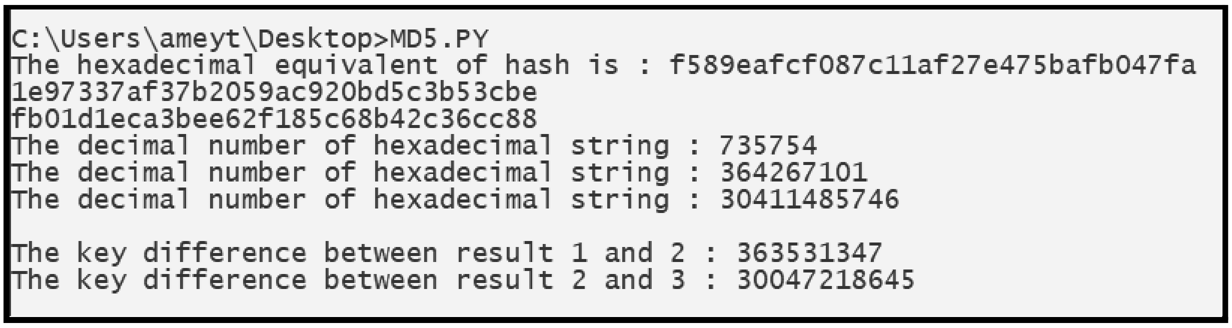
- **MD5**

INPUT:

```
import hashlib

# initializing string
str1 = "MEGA"
str2 = "ARCHIT"
str3 = "SAAKSHI"

# then sending to md5()
result1 = hashlib.md5(str1.encode())
result2 = hashlib.md5(str2.encode())
result3 = hashlib.md5(str3.encode())

# printing the equivalent hexadecimal value.
print("The hexadecimal equivalent of hash is : ", end ="")
print(result1.hexdigest())
print(result2.hexdigest())
print(result3.hexdigest())
```

```
result1 = int(str1, 32)
result2 = int(str2, 32)
result3 = int(str3, 32)

print("The decimal number of hexadecimal string : " + str(result1))
print("The decimal number of hexadecimal string : " + str(result2))
print("The decimal number of hexadecimal string : " + str(result3))

diff1=result2-result1
diff2=result3-result2

print(" ")
print("The key difference between result 1 and 2 : " + str(diff1))
print("The key difference between result 2 and 3 : " + str(diff2))
```

OUTPUT:



```
C:\Users\ameyt\Desktop>MD5.PY
The hexadecimal equivalent of hash is : f589eafcf087c11af27e475bafb047fa
1e97337af37b2059ac920bd5c3b53cbe
fb01d1eca3bee62f185c68b42c36cc88
The decimal number of hexadecimal string : 735754
The decimal number of hexadecimal string : 364267101
The decimal number of hexadecimal string : 30411485746

The key difference between result 1 and 2 : 363531347
The key difference between result 2 and 3 : 30047218645
```

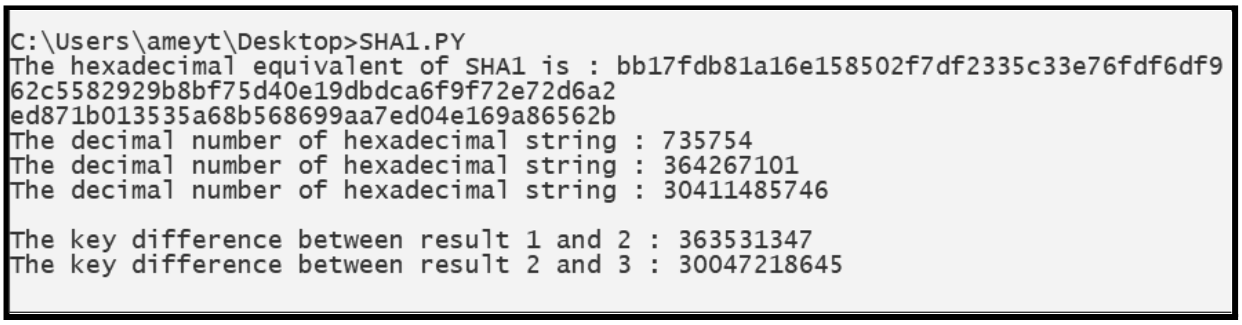
- **SHA1**

INPUT:

```
import hashlib

# initializing string
str1 = "MEGA"
str2 = "ARCHIT"
str3 = "SAAKSHI"

# then sending to SHA1()
result1 = hashlib.sha1(str1.encode())
result2 = hashlib.sha1(str2.encode())
result3 = hashlib.sha1(str3.encode())

# printing the equivalent hexadecimal value.
print("The hexadecimal equivalent of SHA1 is : ", end ="")
print(result1.hexdigest())
print(result2.hexdigest())
print(result3.hexdigest())

result1 = int(str1, 32)
```

```
result2 = int(str2, 32)
result3 = int(str3, 32)

print("The decimal number of hexadecimal string : " + str(result1))
print("The decimal number of hexadecimal string : " + str(result2))
print("The decimal number of hexadecimal string : " + str(result3))

diff1=result2-result1
diff2=result3-result2

print(" ")
print("The key difference between result 1 and 2 : " + str(diff1))
print("The key difference between result 2 and 3 : " + str(diff2))
```

OUTPUT:



```
C:\Users\ameyt\Desktop>SHA1.PY
The hexadecimal equivalent of SHA1 is : bb17fdb81a16e158502f7df2335c33e76fdf6df9
62c5582929b8bf75d40e19dbdca6f9f72e72d6a2
ed871b013535a68b568699aa7ed04e169a86562b
The decimal number of hexadecimal string : 735754
The decimal number of hexadecimal string : 364267101
The decimal number of hexadecimal string : 30411485746

The key difference between result 1 and 2 : 363531347
The key difference between result 2 and 3 : 30047218645
```

OBSERVATION:

MD5 codes any stream of bytes into a 128-bit value while SHA1 codes any stream of bytes into a 160-bit value. Therefore the SHA1 will provide more security compared to MD5. The MD5 algorithm is cheaper to compute, however, MD5 is found to be more vulnerable to collision attacks. Comparison of both algorithms in terms of output length, memory usage, processing time, variance and time to brute force can be seen in more detail can be seen in Table.

Analyzing Subject	**MD5**	**SHA-1**	**Better Algorithm**
Output length	128 bit	160 bit	SHA-1
Memory usage	30060 byte	30267,6 byte	MD5
Processing time	17,233 ms	17,633 ms	MD5
Variance	0,9013 x 10^{-3}	0,4857 x 10^{-3}	SHA-1
Time for brute force	6,8841x10^{29} years	2,8825x10^{29} years	SHA-1

For passwords of length 6, 7, 8, and 9 characters, indicating that the algorithm SHA-1 algorithm is stronger than MD5. The brute force attack time for 6 characters in SHA-1 was 9.71 minutes on average while in MD5 was 8.65 minutes. The estimated time of brute force attack for SHA-1 for 7, 8, and 9 characters passwords was 18.92 days, 974.06 hours, and 8397.8 hours each, while for MD5 was 16.34 hours, 860.51 hours, and 8229.95 hours.

SHA-1 has a processing time that is not much different than MD5, even relatively the same. The MD5 algorithm has a processing time of 0.029 seconds for each variation of the password length (8, 9, 10 characters), MD5 + salt for 0.028 seconds, SHA-1 for 0.029 seconds, while SHA-1 + salt has 0.03 seconds. While for the use of CPU resources, SHA-1 uses more resources than MD5, but the differences were not so big. The MD5 algorithm used 9.56%, 9.96%, and 10.57% CPU resources for a password length of 8, 9, and 10 characters respectively. While MD5 + salt used 9.85%, 10.29%, and 10.96%, SHA-1 used 10.36%, 10.96%, and 12.4%, and SHA1 + salt used 10.39%, 11.27%, and 12.65 % respectively. From the three testings conducted, we can conclude that SHA-1 + salt gives better security for password protection in Simple-O compared to MD5 + salt while not overload the performance of existing systems, and the use of SHA-1 for securing authentication system hopefully can be used widely in the future to replace the use of MD5.

B.2 Question of Curiosity:

1. How many bits of output MD5 generate?

Ans:

➔ MD5 processes a variable-length message into a fixed-length output of 128-bits.

2. How many bits of output SHA1 generates?

Ans:

➔ In cryptography, SHA-1 (Secure Hash Algorithm 1) is a cryptographic hash function that takes an input and produces a 160-bit (20-byte) hash value known as a message digest – typically rendered as a hexadecimal number, 40 digits long.

3. Discuss the strength and limitations of MD5 and SHA1.

Ans:

MD5	SHA1
MD5 stands for Message Digest.	While SHA1 stands for Secure Hash Algorithm.
MD5 can have 128 bits length of the message digest.	Whereas SHA1 can have 160 bits length of the message digest.
The speed of MD5 is fast in comparison to SHA1's speed.	While the speed of SHA1 is slow in comparison to MD5's speed.
To make out the initial message the aggressor would want 2^128 operations whereas exploitation the MD5 algorithmic program.	On the opposite hand, in SHA1 it'll be 2^160 that makes it quite troublesome to seek out.
MD5 is simpler than SHA1.	While SHA1 is more complex than MD5.
MD5 provides indigent or poor security.	While it provides balanced or tolerable security.
In MD5, if the assailant needs to seek out the 2 messages having identical message digest then the assailant would need to perform 2^64 operations.	Whereas in SHA1, the assailant would need to perform 2^80 operations which is greater than MD5.
MD5 was presented in 1992.	While SHA1 was presented in the year 1995.

4. How SHA512 differs from other variants of SHA.

Ans:

- ➔ SHA-512 Cryptographic Hash Algorithm. A cryptographic hash (sometimes called 'digest') is a kind of 'signature' for a text or a data file. SHA-512 generates an almost-unique 512-bit (32-byte) signature for a text.
- ➔ Note that SHA512 is a lot slower to compute than SHA256. In the context of secure hashing, this is an asset. Slower to compute hashes mean it takes more compute time to crack, so if you can afford the compute cost SHA512 will be more secure for this reason.
- ➔ SHA-512 is a hashing algorithm that performs a hashing function on some data given to it. It's part of a group of hashing algorithms called SHA-2 which includes SHA-256 as well which is used in the bitcoin blockchain for hashing.
- ➔ SHA-512 does its work in a few stages. These stages go as follows:
    1. Input formatting
    2. Hash buffer initialization
    3. Message Processing
    4. Output

B.3 Conclusion:

(Write an appropriate conclusion.)

In conclusion, hashing is a useful tool to verify files are copied correctly between two resources. It can also be used to check if files are identical without opening and comparing them. In this experiment, we find differences between the two hash values. using SHA1 And MD5. After the successful completion of this experiment, we can analyze and evaluate the performance of hashing algorithms.