FR. CONCEICAO RODRIGUES COLLEGE OF ENGINEERING
Department of Computer Engineering

Course, Subject & Experiment Details

Academic Year	**2023-24**	**Estimated Time**	**02 - Hours**
Course & Semester	**T.E. (CMPN)- Sem VI**	**Subject Name & Code**	**CSS - (CSC602)**
Module No.	**03 – Mapped to CO- 3**	**Chapter Title**	**Cryptographic Hash Functions**

Practical No:	**5**
Title:	**Performance Analysis of Hash Algorithms**
Date of Performance:	**13/02/2024**
Date of Submission:	**28/02/2024**
Roll No:	**9603**
Name of the Student:	**Zane Vijay Falcao**

Evaluation:

Sr. No	Rubric	Grade
1	On time submission Or completion (2)	
2	Preparedness(2)	
3	Skill (4)	
4	Output (2)	

Signature of the Teacher: **Date:**

Title: For varying message sizes, test integrity of message using MD-5, SHA-1, and analyse the performance of the two protocols.

Lab Objective:

This lab provides insight into:

· The working of MD5 and SHA-1 and variations of SHA-1 and analyze the performance of both for varying message sizes.

Reference: "Cryptography and Network Security" B. A. Forouzan
"Cryptography and Network Security" Atul Kahate
www.md5summer.org/download.html

Prerequisite: Java or Python and Knowledge of hashing and Crypt API.

Theory:

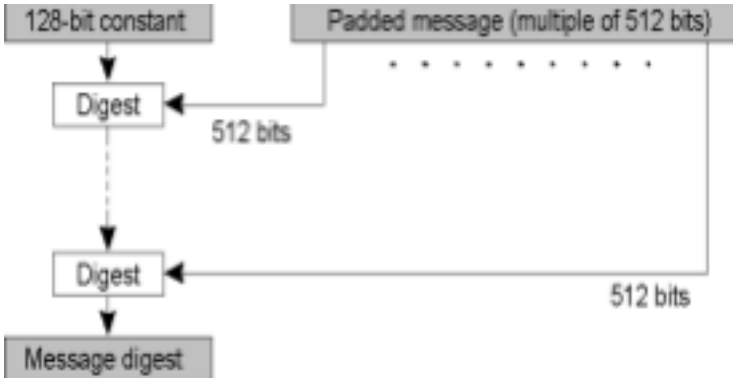
Cryptographic hash functions are a very useful tool in cryptography. They are applied in many areas like integrity of messages, storage of passwords securely and protect signatures. The three hash algorithms SHA-1, SHA-512 and MD5 are considered to analyze their performance.

MD5

· Takes as input a message of arbitrary length and produces as output a 128 bit "fingerprint" or "message digest" of the input.

· It is conjectured that it is computationally infeasible to produce two messages having the same message digest.

· Intended where a large file must be "compressed" in a secure manner before being encrypted with a private key under a public-key cryptosystem such as PGP



Input:
Suppose a b-bit message as input, and that we need to find its message digest.

Algorithm:

Step 1 – append padding bits:

– The message is padded so that its length is congruent to 448, modulo 512. - Means extended to just 64 bits of being of 512 bits long.

– A single "1" bit is appended to the message, and then "0" bits are appended so that the length in bits equals 448 modulo 512.

• Step 2 – append length

– A 64 bit binary representation of b is appended to the result of the previous step. – The resulting message has a length that is an exact multiple of 512 bits.

• Step 3 – Divide the input into 512-bit blocks

Now we divide the input message into into blocks , each of length 512 bits.

• Step 4 – 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 32 bit register.
- These registers are initialized to the following values in hexadecimal:

word A: 01 23 45 67
word B: 89 ab cd ef
word C: fe dc ba 98
word D: 76 54 32 10

Four auxiliary functions

In addition MD5 uses four auxiliary functions that each take as input three 32-bit words and produce as output one 32-bit word. They apply the logical operators and, or, not and xor to the input bits.

Round 1 = (b and c) or ((not(b) and d))
Round 2 = (b and d) or (c and not(d))
Round 3 = B xor c xor d
Round 4 = C xor (b or not(d))

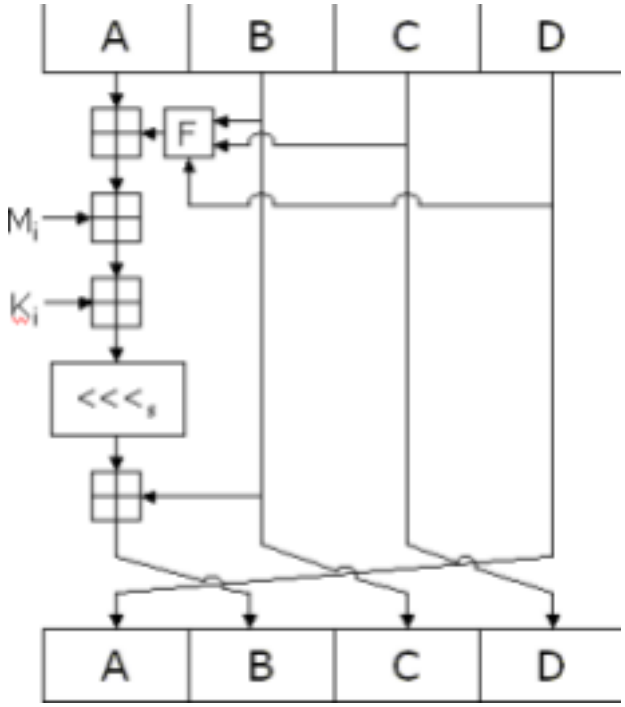
The Constant t[i] or k[i]

MD5 further uses a table K that has 64 elements. Element number i is indicated as K_i. The table is computed beforehand to speed up the computations. The elements are computed using the mathematical sin function:

$K_i = abs(sin(i + 1)) * 2^{32}$

• Step 5 – Process message in 16-word blocks.

1. – Process message in 16-word (512-bit) blocks:
   – Using 4 rounds of 16 bit operations on message block & buffer
   – Add output to buffer input to form new buffer value
2. Output hash value is the final buffer value
3. The contents of the four buffers (A, B, C and D) are now mixed with the words of the input, using the four auxiliary functions (F). There are four *rounds*, each involves 16 basic *operations*. One operation is illustrated in the figure below.



The figure shows how the auxiliary function F is applied to the four buffers (A, B, C and D), using message word M_i and constant K_i. The item "<<<s" denotes a binary left shift by *s* bits.

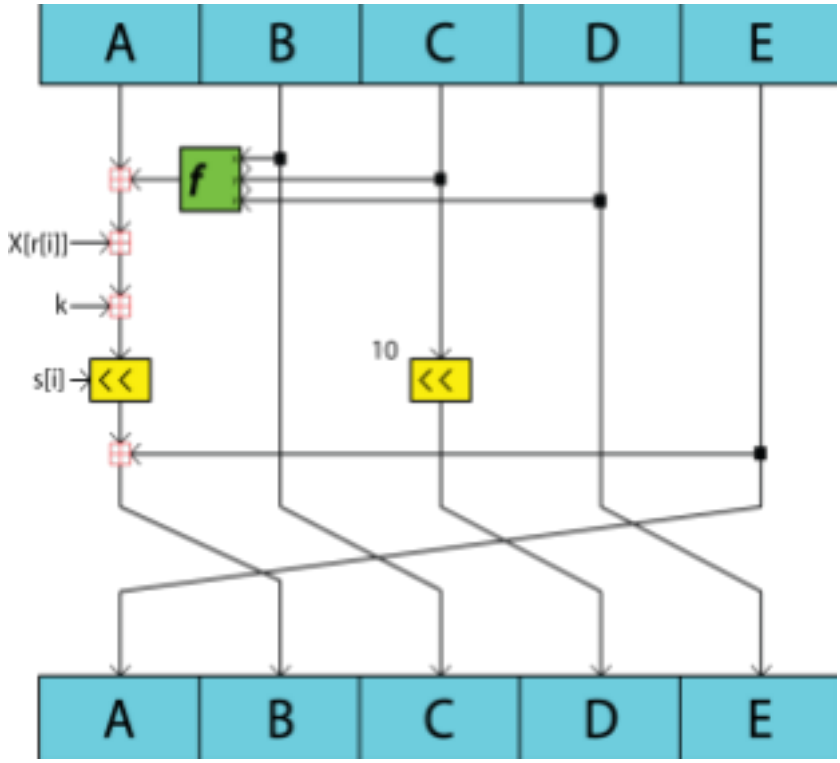
Round 1.
[abcd k s i] denote the operation a = b + ((a + F (b, c, d) + X [k] + T [i]) <<< s).

Do the following 16 operations.
[ABCD 0 7 1] [DABC 1 12 2] [CDAB 2 17 3] [BCDA 3 22 4] [ABCD 4 7 5] [DABC 5 12 6] [CDAB 6 17 7] [BCDA 7 22 8] [ABCD 8 7 9] [DABC 9 12 10] [CDAB 10 17 11] [BCDA 11 22 12] [ABCD 12 7 13] [DABC 13 12 14] [CDAB 14 17 15] [BCDA 15 22 16]

Output:

– The message digest produced as output is A, B, C, D.
– That is, output begins with the low-order byte of A, and end with the high-order byte of D.

SHA-1
Processing is similar to SHA-1 with small variations. In SHA-1, chaining variables are 5 and Boolean operations are different.



Analysis

Differences between MD5 and SHA Algorithms

Keys For Comparison	**MD5**	**SHA**
Security	Less Secure than SHA	High Secure than MD5
Message Digest Length	128 Bits	160 Bits
Attacks required to find out original Message	2^{128} bit operations required to break	2^{160} bit operations required to break
Attacks to try and find two messages producing the	2^{64} bit operations required to break	2^{80} bit operations required to break
same MD		
Speed	Faster, only 64 iterations	Slower than MD5, Required 80 Iterations
Successful attacks so far	Attacks reported to some extents	No such attach report yet

MD5 Execution

Test Strings	MD5	SHA-1
1234567890	f807f1fcf80d030febe008fa17 08e 1ef 31	
abcdefghijklm nopqrstuvwxyz	f3fcf3f711e2f4001dfb191cfa1 7f1 0b 15	
message digest	f91b191d1ce7e3ed121a0f01ea f11 1f0 15	

Timing comparison between MD5 and SHA-1

File Size	MD5	SHA-1
1 KB		
5 KB		
10 KB		

Practical and Real Time Applications

- In Windows OS, PowerShell function "Get-FileHash"
- Android ROMs
- File servers - file servers often provide a pre-computed MD5 (known as md5sum) checksum for the files, so that a user can compare the checksum of the downloaded file to it.
- Most unix-based operating systems include MD5 sum utilities in their distribution packages

Conclusion:
The program was tested for different sets of inputs. Program is working SATISFACTORY NOT SATISFACTORY (Tick appropriate outcome)

Post Lab Assignment:

1. Why is SHA-1 more secure than MD5?
2. Which of the following is not included in hash function?
   a. Authentication.
   b. Message integrity.
   c. Fingerprinting.
   d. Inefficiency.
3. Which of the following is used to detect transmission errors, and not to detect intentional tampering with data?
   a. CRC.
   b. Similar checksum.
   c. WEP.
   d. Hash function.
4. Which of the following is not provide by hash function?
   a. Efficiency.
   b. Two-way.
   c. Compression.
   d. Weak collision resistance.

MD5

In [1]:
```
import hashlib
from datetime import datetime
```

In [2]:
```
# MD5 Test String: 1234567890

str2hash = "1234567890"
t1 = datetime.now()

result = hashlib.md5(str2hash.encode())
t2 = datetime.now()

print("The hexadecimal equivalent hash is: ",end="")
print(result.hexdigest())
print("time taken by MD5: ",t2-t1)
```

```
The hexadecimal equivalent hash is: e807f1fcf82d132f9bb018ca6738a19f
time taken by MD5:  0:00:00
```

In [3]:
```
 # MD5 Test String: abcdefghijklmnopqrstuvwxyz


str2hash = "abcdefghijklmnopqrstuvwxyz"
t1 = datetime.now()

result = hashlib.md5(str2hash.encode())
t2 = datetime.now()

print("The hexadecimal equivalent hash is: ",end="")
print(result.hexdigest())
print("time taken by MD5: ",t2-t1)
```

```
The hexadecimal equivalent hash is: c3fcd3d76192e4007dfb496cca67e13b
time taken by MD5:  0:00:00
```

In [4]:
```
 # MD5 Test String: message digest


str2hash = "message digest"
t1 = datetime.now()

result = hashlib.md5(str2hash.encode())
t2 = datetime.now()

print("The hexadecimal equivalent hash is: ",end="")
print(result.hexdigest())
print("time taken by MD5: ",t2-t1)
```

```
The hexadecimal equivalent hash is: f96b697d7cb7938d525a2f31aaf161d0
time taken by MD5:  0:00:00
```

SHA-1

In [5]:
```
# MD5 Test String: 1234567890

str2hash = "1234567890"
t1 = datetime.now()

result = hashlib.sha1(str2hash.encode())
t2 = datetime.now()

print("The hexadecimal equivalent hash is: ",end="")
print(result.hexdigest())
print("time taken by SHA-1: ",t2-t1)
```

```
The hexadecimal equivalent hash is: 01b307acba4f54f55aafc33bb06bbbf6ca803e9a
time taken by SHA-1:  0:00:00.001001
```

In [6]:
```
# SHA-1 Test String: abcdefghijklmnopqrstuvwxyz

str2hash = "abcdefghijklmnopqrstuvwxyz"
t1 = datetime.now()

result = hashlib.sha1(str2hash.encode())
t2 = datetime.now()

print("The hexadecimal equivalent hash is: ",end="")
print(result.hexdigest())
print("time taken by SHA-1: ",t2-t1)
```

```
The hexadecimal equivalent hash is: 32d10c7b8cf96570ca04ce37f2a19d84240d3a89
time taken by SHA-1:  0:00:00
```

In [7]:
```
 # SHA-1 Test String: message digest


str2hash = "message digest"
t1 = datetime.now()

result = hashlib.sha1(str2hash.encode())
t2 = datetime.now()

print("The hexadecimal equivalent hash is: ",end="")
print(result.hexdigest())
print("time taken by SHA-1: ",t2-t1)
```

```
The hexadecimal equivalent hash is: c12252ceda8be8994d5fa0290a47231c1d16aae3
time taken by SHA-1:  0:00:00
```

In [8]:
```python
# time comparison between MD5 and SHA1

import hashlib
import time

def calculate_hash_md5(file_path):
    with open(file_path, 'rb') as f:
        md5_hash = hashlib.md5()
        while chunk := f.read(4096):
            md5_hash.update(chunk)
    return md5_hash.hexdigest()

def calculate_hash_sha1(file_path):
    with open(file_path, 'rb') as f:
        sha1_hash = hashlib.sha1()
        while chunk := f.read(4096):
            sha1_hash.update(chunk)
    return sha1_hash.hexdigest()

def measure_time(file_path, algorithm):
    start_time = time.time()
    if algorithm == 'md5':
        calculate_hash_md5(file_path)
    elif algorithm == 'sha1':
        calculate_hash_sha1(file_path)
    end_time = time.time()
    return end_time - start_time

file_sizes = {'1KB': '1kb.txt','5KB': '5kb.txt','10KB': '10kb.txt'}

for size, file_name in file_sizes.items():
    print(f"Calculating hash for {size} file...")
    md5_time = measure_time(file_name,'md5')
    sha1_time = measure_time(file_name, 'sha1')

    print(f"MD5 hash time: {md5_time: .6f} seconds")
    print(f"SHA1 hash time: {sha1_time: .6f} seconds")
    print()
```

```
Calculating hash for 1KB file...
MD5 Hash time: 0.000580 seconds
SHA1 Hash time: 0.000017 seconds

Calculating hash for 5KB file...
MD5 Hash time: 0.000104 seconds
SHA1 Hash time: 0.000009 seconds

Calculating hash for 10KB file...
MD5 Hash time: 0.000097 seconds
SHA1 Hash time: 0.000011 seconds
```

Postlab - Exp 3 9603 Zane Fulcao Batch C

1)

Ans SHA1 is more secure than MD5 in the following ways

1. Longer digest length: SHA1 employs a stranger a longer digest as compared to MD5 making it less prone to collision attacks due to increased number of possible hash values.
2. Stronger algorithm design: SHA-1 employs a ~~stronger algorithm~~ algorithmic design with more complex operations including bitwise operations and logical funtions compared to MD5 making it less suspectible to cryptographic vulnerabilities.
3. Resistance to preimage attack: SHA-1 is more resistent to preimage attacks compared to MD5 meaning, its harder to reverse-engineer the original input from its hash value, ~~each~~ enhancing its security for applications requiring data integerity verifications.
4. Cryptographic strength: SHA-1 provides storage cryptographic strength than MD5 due to to its larger digest size & algorithmic complexity making it a more robust choice for security sensitive applications.
5. Industry Recommendations: SHA-1 is recommend over MD5 by cryptographic experts & industry standards bodies like NIST due to its improved security properties making it a more trusted and widely adopted hashing algorithm in practice.

2) d) Inefficiency

Hash functions are primarily used for authentication message integerity and finger-printing. while inefficiency can be a consideration in choosing a hash functions, its not a fundamental property of hash functions themselves

3) a) CRC

Cyclic Redundancy check is primarily used to detect transmission errors such as those that can occur during data transmission, over a network or storage medium while CRC can detect accidental errors, it is not designed for intentionally tampering with data

1) b) Two way

Hash functions are designed to become way functions meaning they are easy to complete in one direction but computationally infeasible to reverse.