Experiment No. 4
For varying message sizes, test integrity of message using MD-5, SHA-1, and analyse the performance of the two protocols. Use crypt APIs
Date of Performance:
Date of Submission:

Experiment No. 4

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

Objectives:

• To understand the applications of cryptographic hash functions.

• To distinguish between MD5 & SHA-1.

• To differentiate between hashing and encryption.

Outcomes: The learner will be able to Apply security techniques and technologies to solve real-life security problems in practical systems.

Hardware / Software Required: C/C++/JAVA.

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.

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 Ki 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 four similar stages, termed rounds; each round is composed of 16 similar operations based on a nonlinear function F, modular addition, and left rotation.

There are four possible functions F; a different one is used in each round: 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, modulo 512. 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, modulo 512. 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 in accordance with 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 of 16.

3. Initialize MD Buffer:

A fourword 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, loworder bytes first):

4. Process Message in 16Word Blocks:

We first define four auxiliary functions that each take as input three 32bit words and produce as output one 32bit word.

5. Output

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

SHA Secure Hash Algorithm was developed by National Institute of Standards and Technology (NIST). Based on different digest lengths, SHA includes algorithms such as SHA-1, SHA-256,

SHA-384 and SHA-512. Unlike encryption, given a variable length message x, a secure hash algorithm computes a function H(x) which has a fixed bit. When a message of any length , the SHA-1 produces a 160-bit output called message digest. SHA-1 called secure because it is computationally infeasible to find a message which corresponds to a given message digest, or to find two different messages which produce the same message digest. The most commonly used hash function from the SHA family is SHA-1. SHA-1 is used in SSL/TLS, PGP, SSH, MIME and IPsec for security and authentication purpose.

Step – 1: Padding - The first step of SHA-1 is added padding to the end of original message to prepare message in multiple of 512 bits.

Step – 2: Append Length – The length of message excluding the length of the padding is now calculated and appended to the end of the padding as 64-bit block. (message length is 64 bits short of multiple of 512).

Step – 3: Divide the input into 512-bit blocks: The input message is now divided into blocks, each of length 512 bits.

Step – 4: Initialize chaining variables: Now, five chaining variables A to E are initialized. Each of 32 bits variable produces 160 bits length of message digest.

Step – 5: Process Block & Output – Combination of A-E chaining variable is called ABCDE, will be considered as a single register. Now divided the current 512-bit block into 16 sub blocks, each consisting of 32 bits. (32x16=512) SHA-1 has perform four rounds. Each round takes the current 512-bit block, the register ABCDE and constant K(t) (where t=0 to 79) as input. SHA consists of four rounds, each round containing 20 iterations. So total iteration is 80.

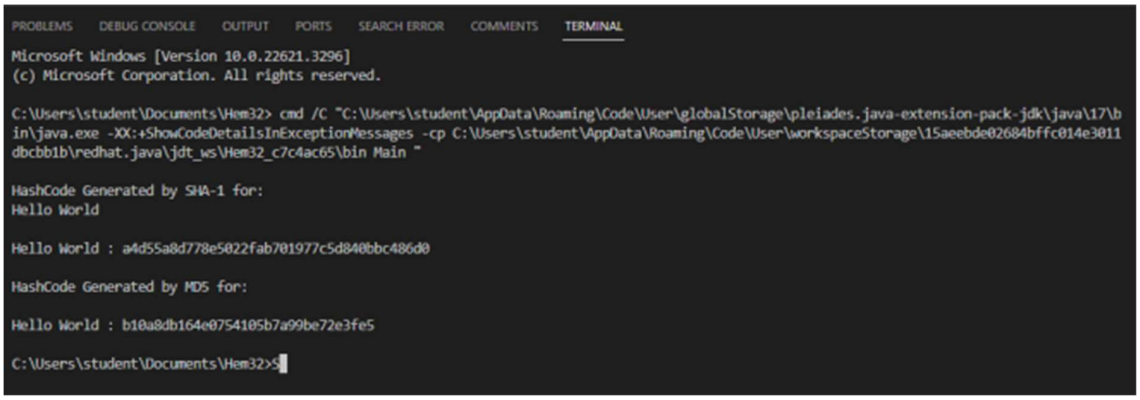
Implementation:

```
import java.math.BigInteger;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import java.util.Scanner;
```

```
public class Main {
public static String getMd5(String input) {
try {
MessageDigest md = MessageDigest.getInstance("MD5");
byte[] messageDigest = md.digest(input.getBytes());
BigInteger no = new BigInteger(1, messageDigest);
String hashtext = no.toString(16);
while (hashtext.length() < 32) {
hashtext = "0" + hashtext;
}
return hashtext;
} catch (NoSuchAlgorithmException e) {
throw new RuntimeException(e);
}
}
public static String encryptThisString(String input) {
try {
MessageDigest md = MessageDigest.getInstance("SHA-1");
byte[] messageDigest = md.digest(input.getBytes());
BigInteger no = new BigInteger(1, messageDigest);
String hashtext = no.toString(16);
while (hashtext.length() < 32) {
hashtext = "0" + hashtext;
}
return hashtext;
} catch (NoSuchAlgorithmException e) {
throw new RuntimeException(e);
}
```

```
}
public static void main(String args[]) throws NoSuchAlgorithmException {
System.out.println("\nHashCode Generated by SHA-1 for: ");
Scanner sc = new Scanner(System.in);
String str = sc.nextLine();
System.out.println("\n" + str + " : " + encryptThisString(str));
System.out.println("\nHashCode Generated by MD5 for: ");
System.out.println("\n" + str + " : " + getMd5(str));
sc.close();
}
}
```

Output:



Conclusion: To test the integrity of messages with varying sizes using MD5 and SHA-1 cryptographic hash functions, the experiment involved understanding their applications, distinguishing between MD5 and SHA-1, and differentiating hashing from encryption. MD5, with its 128-bit hash value, processes variable-length messages into fixed-length outputs, while SHA-1 produces a 160-bit output. Performance analysis was conducted to evaluate the efficiency of both protocols.It utilizes five chaining variables and performs four rounds with These cryptographic techniques are essential for ensuring message integrity and security in practical systems, with SHA-1 being widely utilized in various security applications like SSL/TLS, PGP, and IPsec.