Experiment Number: D1

TITLE:

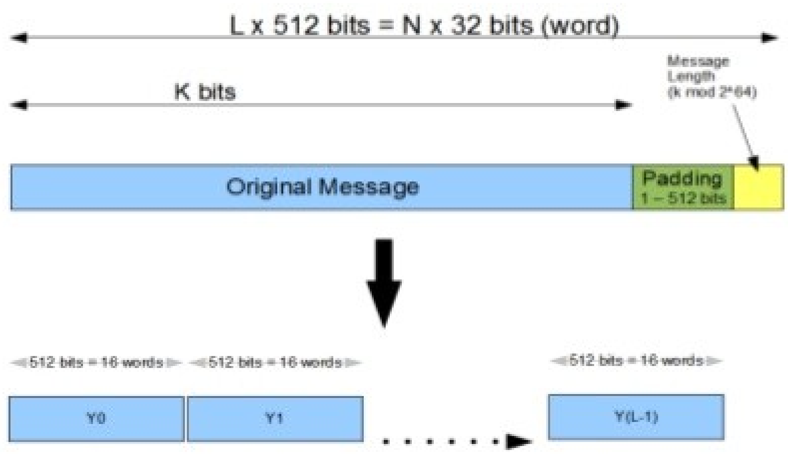
A message is to be transmitted using network resources from one machine to another, calculate and demonstrate the use of a Hash value equivalent to SHA-1. Develop program in C++/Python/Scala/Java using Eclipse.

OBJECTIVES:

To understand SHA-1 algorithm and implement a program to transmit a message from one machine to another by using a hash value equivalent to SHA-1.

THEORY:

SHA-1 (Secure Hash Algorithm) is a most commonly used from SHA series of cryptographic hash functions, designed by the National Security Agency of USA and published as their government standard. SHA-1 produces the 160-bit hash value. Original SHA (or SHA-0) also produces 160-bit hash value, but SHA-0 has been withdrawn by the NSA shortly after publication and was superseded by the revised version commonly referred to as SHA-1. The other functions of SHA series produce 224-, 256-, 384- and 512-bit hash values.



MATHEMATICAL MODEL:

Let U = {s, e, f, S, F, I, O, DD, NDD } be a universal set where,

s = start

e = end

f = set of functions

I = Input set

O = Output set

DD = Deterministic Data

ND = Non-Deterministic Data

S = Cases of Success

F = Cases of Failure

In our program, f=sha1() // which calculates the hash value of the text given as a parameter

Sha1() also consists of functions for chunking i.e. chunk(), Rot() for rotating

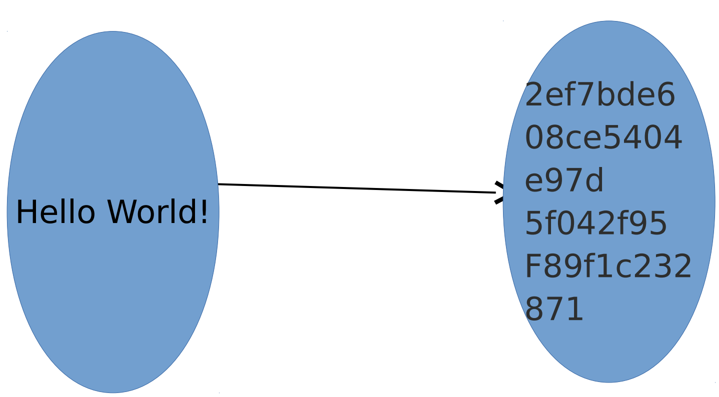
I= input text whose hash value has to be calculated

O= Result that Message has been tampered or not along with the hash value

DD= h0,h1, h2,h3,h4

Venn Diagram :

Every file will correspond to a unique hash value. Thus there is 1:1 relationship between the input file and hash values calculated. A small change in the input file makes a significant difference in the calculated hash value.



IMPLEMENTATION DETAILS / DESIGN LOGIC:

SHA1 Algorithm:-

Consider an input text to be hashed “Hello World!”

Step 0: Initialize variables

There are five variables that need to be initialized.

h0 = 01100111010001010010001100000001

h1 = 11101111110011011010101110001001

h2 = 10011000101110101101110011111110

h3 = 00010000001100100101010001110110

h4 = 11000011110100101110000111110000

Step 1: Select the input text

In this example I am going to use the string: 'Hello World!'.

Step 2: Break it into characters

H e l l o W o r l d !
Note that spaces count as characters.

Step 3: Convert characters to ASCII codes

Each character should now be converted from text into ASCII. ASCII or the 'American Standard Code for Information Interchange' is a standard that allows computers to communicate different symbols by assigning each one a number. No matter what font or language you use, the same character will always have the same ASCII code.

72 101 108 108 111 32 87 111 114 108 100 33

Step 4: Convert numbers into binary

All base ten numbers are now converted into 8-bit binary numbers. The eight-bit part means that if they don't actually take up a full eight place values, simply append zeros to the beginning so that they do.

Step 5: Add '1' to the end

Put the numbers together and add the number '1' to the end :

Step 6: Append '0's' to the end

In this step you add zeros to the end until the length of the message is congruent to 448 mod 512. That means that after dividing the message length by 512, the remainder will be 448.

Step 6.1: Append original message length

This is the last of the 'message padding' steps. You will now add the 64-bit representation of the original message length, in binary, to the end of the current message.
The message length should now be an exact multiple of 512.

Step 7: 'Chunk' the message

We will now break the message up into 512 bit chunks. In this case the message is only 512 bit's long, so there will be only one chunk that will look exactly the same as the last step.

Step 8: Break the 'Chunk' into 'Words'

Break each chunk up into sixteen 32-bit words

Step 9: 'Extend' into 80 words

This is the first sub-step. Each chunk will be put through a little function that will create 80 words from the 16 current ones.
This step is a loop. What that means is that every step after this will be repeated until a certain condition is true.
In this case we will start by setting the variable 'i' equal to 16. After each run through of the loop we will add 1 to 'i' until 'i' is equal to 79.

We begin by selecting four of the current words. The ones we want are: [i-3], [i-8], [i-14] and [i-16]. That means for the first time through the loop we want the words numbered: 13, 8, 2 and 0.

0: 01000001001000000101010001100101

2: 00000000000000000000000000000000

8: 00000000000000000000000000000000

13: 00000000000000000000000000000000

Now that we have our words selected we will start by performing what's known as an 'XOR' or 'Exclusive OR' on them. In the end all four words will be XOR'ed together.

00000000000000000000000000000000 :word 13

00000000000000000000000000000000 :word 8

00000000000000000000000000000000 :word 13 XOR word 8

00000000000000000000000000000000 :word 13 XOR word 8
01110010011011000110010000100001 :word 2

01110010011011000110010000100001 :(word 13 XOR word 8) XOR word 2

01110010011011000110010000100001 (word 13 XOR word 8) XOR word 2
01001000011001010110110001101100 :word 0

00111010000010010000100001001101 ((word 13 XOR word 8) XOR word 2) XOR word 0

01110100000100100001000010011010 :Left Rotated 1

After step nine is complete we will now have 80 words. The last 5 have been listed below:

75: 01100011110010111001110001110101
76: 010011000000101110100111011110000
77: 01001000000100111100011010100001
78: 01011100010101101011111100111010
79: 10101011100010110110010010110100

Step 10: Initialize variables

Set the letters A-->E equal to the variables h0-->h4.

A = h0

B = h1

C = h2

D = h3

E = h4

Step 11: The main loop

This loop will be run once for each word in succession.

Step 11.1: Four choices

Depending on what number word is being input, one of four functions will be run on it.

Words 0-19 go to function 1.
Words 20-39 go to function 2
Words 40-59 go to function 3
Words 60-79 go to function 4

Function1 : (B AND C) or (!B AND D)
Function2 : B XOR C XOR D
Function3 : (B AND C) OR (B AND D) OR (C AND D)
Function4 : B XOR C XOR D

After completing one of the four functions above, each variable will move on to this sub step before restarting the loop with the next word. For this step we are going to create a new variable called 'temp' and set it equal to: (A left rotate 5) + F + E + K + (the current word).

Step 12:

h0 = h0 + A

h1 = h1 + B

h2 = h2 + C

h3 = h3 + D

h4 = h4 + E

If these variables are longer than 32 bits they should be truncated.

Finally the variables are converted into base 16 (hex) and joined together.

2ef7bde6 :h0 in hex
8ce5404 :h1 in hex
e97d5f04 :h2 in hex
2f95f89f :h3 in hex
1c232871 :h4 in hex

2ef7bde608ce5404e97d5f042f95f89f1c232871 :digest

Input :- Hello World!

Expected Output :- 2ef7bde608ce5404e97d5f042f95f89f1c232871
If hash values match at both machines then message is not tampered.

TEST CASES:

hello World!	788245B4dad73c1e5a630c126c484c7a2464f280
Hello World!	2ef7bde608ce5404e97d5f042f95f89f1c232871

For every file , hash value is calculated at the sending machine and appended with the file. This appended file is sent over the network to the destination machine and hash value is calculated again at the destination. If the hash values match then message is securely transferred.

CONCLUSION:

We have successfully implemented this program to transmit a message using network resources from one machine to another and demonstrated the use of a Hash value equivalent to SHA-1.

COURSE OUTCOMES ACHIEVED:

Understood Working of SHA-1 algorithm.

FAQ's

- What characteristics are needed in secure hash function?
- What is the role of a compression function in a hash function?
- What are the applications of SHA-1?