| Ex.No.12<br>23/09/2024 | **IMPLEMENTATION OF SECURE HASH ALGORITHM** |
|---|---|

## AIM:

To implement one step in one round of Secure Hash Algorithm -1

## THEORY

### Properties of Hash Functions:

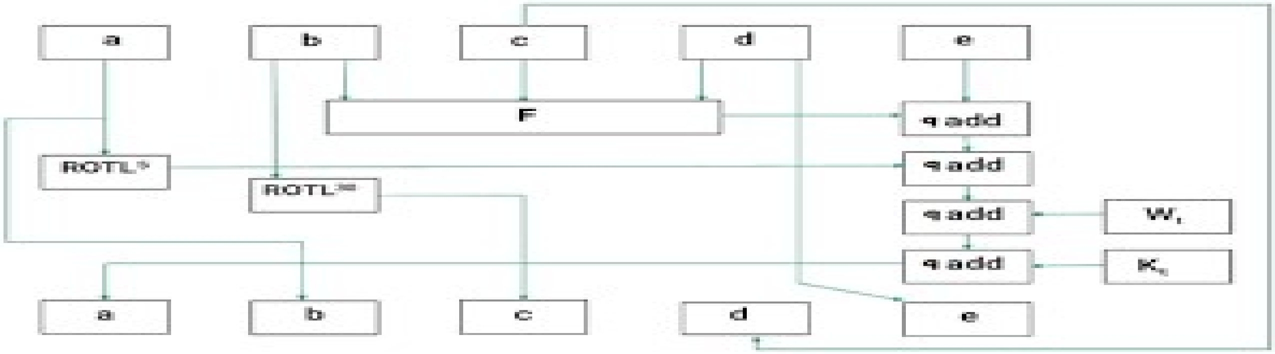
The properties of hash functions are,

Deterministic: The same input always produces the same hash output.

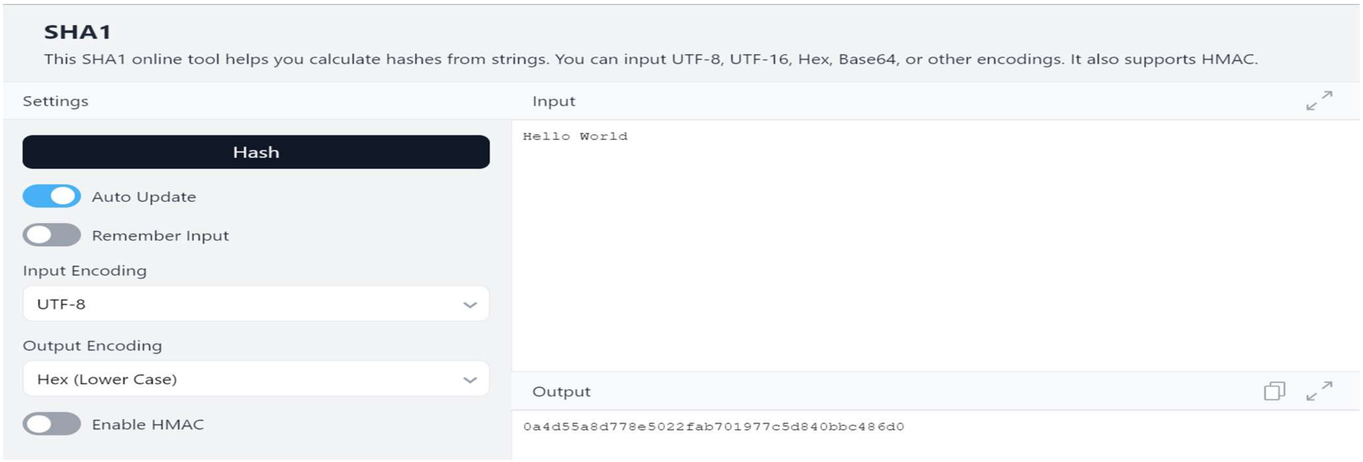
Fixed Output Length: Hash output is always the same length, regardless of input size.

Collision Resistant: Different inputs are unlikely to produce the same hash value.

### Schematic diagram for one step:



### Online Calculator – SHA1 – Screenshots


**SHA1**

This SHA1 online tool helps you calculate hashes from strings. You can input UTF-8, UTF-16, Hex, Base64, or other encodings. It also supports HMAC.

Settings

Hash

● Auto Update

○ Remember Input

Input Encoding: UTF-8

Output Encoding: Hex (Lower Case)

○ Enable HMAC

Input

```
Hello World
```

Output

```
0a4d55a8d778e5022fab701977c5d840bbc486d0
```

## ALGORITHM:

```
1. Preprocessing:
  Padding:
    Append a single '1' bit to the message.
    Append '0' bits until the length is 448 mod 512 (making the total length 64 bits shy of a multiple of 512).
    Append the original message length as a 64bit integer.

2. Initialize Hash Values:
  Set initial hash values (five 32bit words):
    h_0 = 0x67452301
    h_1 = 0xEFCDAB89
    h_2 = 0x98BADCFE
    h_3 = 0x10325476
    h_4 = 0xC3D2E1F0

3. Process the Message in 512bit Chunks:
  Divide the padded message into 512bit blocks.
  For each block:
    Break it into sixteen 32bit words W[0], W[1],..., W[15].
    Extend the sixteen words into eighty 32bit words:
      For t = 16 to 79:
        W[t] = (W[t3] +W[t8] + W[t14] + W[t16]) left rotated by 1

4. Initialize Working Variables:
  Set working variables:
    a = h_0
    b = h_1
    c = h_2
    d = h_3
    e = h_4
```

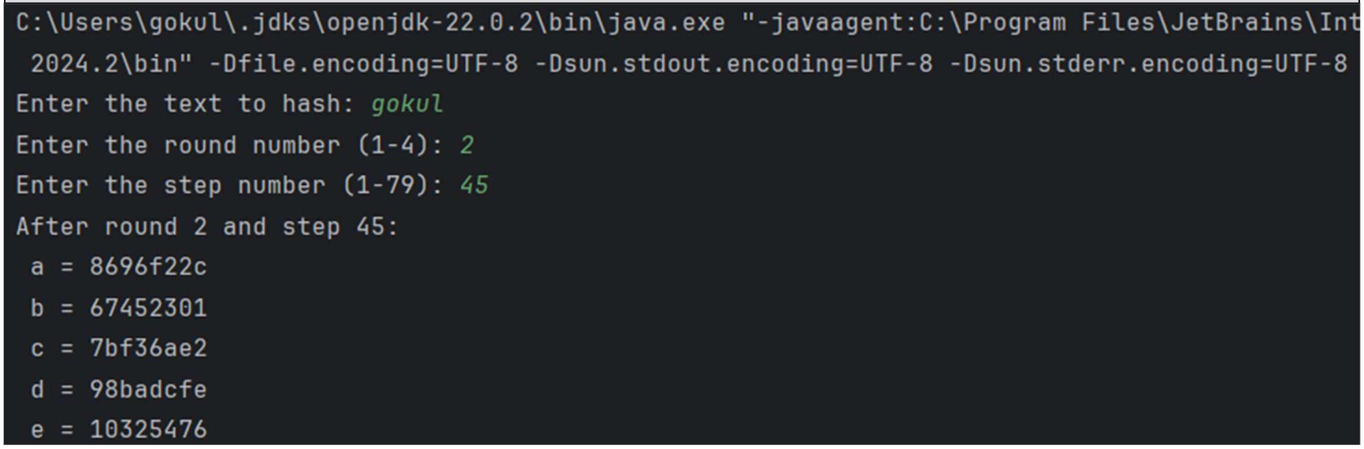
## CODING:

```
import java.util.Scanner;

public class SHA1 {
  private static final int H0 = 0x67452301;
  private static final int H1 = 0xEFCDAB89;
  private static final int H2 = 0x98BADCFE;
  private static final int H3 = 0x10325476;
  private static final int H4 = 0xC3D2E1F0;

  private static int leftRotate(int value, int shift) {
    return (value << shift) | (value >>> (32 - shift));
  }

  private static byte[] padMessage(byte[] message) {
    int originalLength = message.length;
    long originalLengthBits = (long) originalLength * 8;
    int paddingLength = (56 - (originalLength + 1) % 64 + 64) % 64;
    byte[] paddedMessage = new byte[originalLength + paddingLength + 9];

    System.arraycopy(message, 0, paddedMessage, 0, originalLength);
    paddedMessage[originalLength] = (byte) 0x80;

    for (int i = 0; i < 8; i++) {
      paddedMessage[paddedMessage.length - 1 - i] = (byte) (originalLengthBits >>> (i * 8));
    }
    return paddedMessage;
  }

  public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);

    System.out.print("Enter the text to hash: ");
    String inputText = scanner.nextLine();

    byte[] paddedMessage = padMessage(inputText.getBytes());

    System.out.print("Enter the round number (1-4): ");
    int round = scanner.nextInt();
    System.out.print("Enter the step number (1-79): ");
```

```
int step = scanner.nextInt();

if (round < 1 || round > 4 || step < 0 || step > 79) {
    System.out.println("Invalid round or step number. Please enter valid values.");
    return ;
}

int[] w = new int[80];

for (int i = 0; i < 16; i++) {
    w[i] = ((paddedMessage[i * 4] & 0xFF) << 24) |
            ((paddedMessage[i * 4 + 1] & 0xFF) << 16) |
            ((paddedMessage[i * 4 + 2] & 0xFF) << 8) |
            (paddedMessage[i * 4 + 3] & 0xFF);
}

// Extend the sixteen 32-bit words into eighty 32-bit words
for (int i = 16; i < 80; i++) {
    w[i] = leftRotate(w[i - 3] ^ w[i - 8] ^ w[i - 14] ^ w[i - 16], 1);
}

// Initial hash values
int a = H0, b = H1, c = H2, d = H3, e = H4;

// Determine the value of f and k based on the round number
int f, k;
if (round == 1) {
    f = (b & c) | (~b & d);  // First 20 rounds
    k = 0x5A827999;
} else if (round == 2) {
    f = b ^ c ^ d;          // 20 to 39 rounds
    k = 0x6ED9EBA1;
} else if (round == 3) {
    f = (b & c) | (b & d) | (c & d);  // 40 to 59 rounds
    k = 0x8F1BBCDC;
} else {
    f = b ^ c ^ d;          // 60 to 79 rounds
    k = 0xCA62C1D6;
}

// Perform the specific step (for the given step in the block)
```

```
        int temp = leftRotate(a, 5) + f + e + k + w[step];
        e = d;
        d = c;
        c = leftRotate(b, 30);
        b = a;
        a = temp;
 System.out.printf("After round %d and step %d:\n a = %08x\n b = %08x\n c = %08x\n d =
%08x\n e = %08x\n", round, step, a, b, c, d, e);
        scanner.close();
    }
}
```

## SCREEN SHOTS:



```
C:\Users\gokul\.jdks\openjdk-22.0.2\bin\java.exe "-javaagent:C:\Program Files\JetBrains\Int
 2024.2\bin" -Dfile.encoding=UTF-8 -Dsun.stdout.encoding=UTF-8 -Dsun.stderr.encoding=UTF-8
Enter the text to hash: gokul
Enter the round number (1-4): 2
Enter the step number (1-79): 45
After round 2 and step 45:
 a = 8696f22c
 b = 67452301
 c = 7bf36ae2
 d = 98badcfe
 e = 10325476
```

## RESULT:

Thus, we have implemented SHA1 algorithm Successfully.

## Evaluation

| Parameter | Max Marks | Marks Obtained |
|---|---|---|
| Uniqueness of the Code | 50 | |
| Completion of experiment on time | 10 | |
| Documentation | 15 | |
| Total | 75 | |
| Signature of the faculty with Date | | |