

Assignment No:

Title of the Assignment: MD5 Algorithm

Problem statement: Calculate a message digest of a text using MD5 algorithm in JAVA

Objective:

- To understand the fingerprinting of a message.
- To implement MD5 using JAVA.

Theory:

MD5 or "message digest 5" algorithm was designed by professor Ronald Rivest. MD5 is a one way hashing function. So by definition it should fulfill two properties. One, it is one way which means one can create a hash value from a message but cannot recreate the message from the hash value. Two, it should be collision free that is two distinct messages cannot have the same hash value. MD5 creates a 128bit message digest from data input.



Figure 1: Hexadecimal representation of input by md5

The output must be unique from other message digests. Imagine a b-bits message to digest. To digest this message we need to follow 5 steps.

1. Append Padding Bits

The first step is to extend (padded) the b-bits message(input) so that the length of the message is equal to 448, modulo 512. In simpler word the message should be just 64-bits shy of being a multiple of 512 that is after padding the message+64 bit should be divisible by 512. $(\text{Message}+64)/512$ will have remainder 0. No matter the size of the message padding is always done. First a '1' bit is appended to the message and then a series of '0' bits.

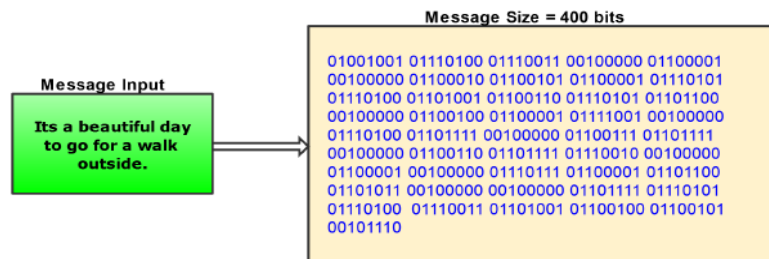


Figure 2: 400 bits original message

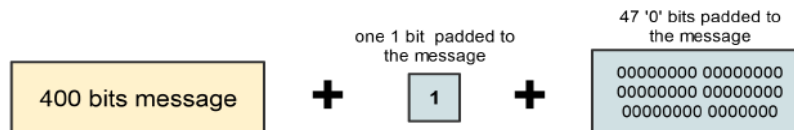


Figure 3: First one '1' bit is added and then '0' bits

For example if our message is 400 bits then we will add one '1' bit and 47 '0' bits which gives us 448 bits which is 64 bits shy of being divisible by 512. If our message is 1200 bits in size then we will add one '1' bit and 271 '0' bits which gives us 1472 bits. $1472 + 64$ is divisible by 512. At least 1 bit and at most 512 bits are padded or extended to the original message.

2. Append Length

Now we take the original message and make a 64-bit representation of the original b-bit message. We append this to the result of the previous step. Now the message has a length that is exactly divisible by 512 or the message is multiple of 512. Which itself is divisible by 16. For example if our message is 400 bits then we will add one '1' bit and 47 '0' bits which gives us 448 bits which is 64 bits shy of being divisible by 512. If our message is 1200 bits in size then we will add one '1' bit and 271 '0' bits which gives us 1472 bits. $1472 + 64$ is divisible by 512. At least 1 bit and at most 512 bits are padded or extended to the original message.

At this point the message is divided into blocks of 512 bits each. Each 512 bits block is divided into 16 words of 32-bits each. We denote the words as $M[0.....N-1]$ where N is a multiple of 16.

3. Initialize MD Buffer

MD5 uses a four word buffer each 32-bits long. We denote them by A,B,C,D.

These are pre-initialized as:

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

4. Process Message in 16-word Blocks

We use two other helper functions. One, we use four auxiliary functions for MD5. Two, we use a table consisting 64-elements

Auxiliary Functions

Auxiliary functions take three inputs of 32-bits word and gives an output of 32-bit word. The auxiliary function apply logical and, or and xor to the inputs. Later on we will show how we use each of these 4 functions in 4 rounds (each round has 16 operations). The functions are:

$F(B, C, D) = (B \wedge C) \vee (\neg B \wedge D)$
$G(B, C, D) = (B \wedge D) \vee (C \wedge \neg D)$
$H(B, C, D) = B \oplus C \oplus D$
$I(B, C, D) = C \oplus (B \vee \neg D)$

The Table

The table consists of 64-elements where each element is computed using mathematical sin function:

$$\text{abs}(\sin(i + 1)) _ 232$$

We denote the table as K[1.....64]. K[i] is the i-th element of the table. Elements of the table is pre calculated.

K[0.. 3]	0xd76aa478, 0xe8c7b756, 0x242070db, 0xc1bdceee
K[4.. 7]	0xf57c0faf, 0x4787c62a, 0xa8304613, 0xfd469501
K[8..11]	0x698098d8, 0x8b44f7af, 0xffff5bb1, 0x895cd7be
K[12..15]	0x6b901122, 0xfd987193, 0xa679438e, 0x49b40821
K[16..19]	0xf61e2562, 0xc040b340, 0x265e5a51, 0xe9b6c7aa
K[20..23]	0xd62f105d, 0x02441453, 0xd8a1e681, 0xe7d3fbc8
K[24..27]	0x21e1cde6, 0xc33707d6, 0xf4d50d87, 0x455a14ed
K[28..31]	0xa9e3e905, 0xfcefa3f8, 0x676f02d9, 0x8d2a4c8a
K[32..35]	0xfffa3942, 0x8771f681, 0x6d9d6122, 0xfde5380c
K[36..39]	0xa4beea44, 0x4bdecfa9, 0xf6bb4b60, 0xbebfbc70
K[40..43]	0x289b7ec6, 0xeaa127fa, 0xd4ef3085, 0x04881d05
K[44..47]	0xd9d4d039, 0xe6db99e5, 0x1fa27cf8, 0xc4ac5665
K[48..51]	0xf4292244, 0x432aff97, 0xab9423a7, 0xfc93a039
K[52..55]	0x655b59c3, 0x8f0ccc92, 0xffeff47d, 0x85845dd1
K[56..59]	0x6fa87e4f, 0xfe2ce6e0, 0xa3014314, 0x4e0811a1
K[60..63]	0xf7537e82, 0xbd3af235, 0x2ad7d2bb, 0xeb86d391

Recall from step 2, where we divided the message into blocks of 512-bits and then each 512-bits block to 16 words of 32-bits. Those 16 word (each 32-bits) blocks are denoted as $M[0....N-1]$.

Now for each word block we perform 4 rounds where each round has 16 operations. Each round follows basic pattern. The operations of each round is denoted as $[abcd \ g \ s \ i]$. Here i is used to get the i -th value from the table $k[i]$. i also denotes the number of operation from 0 to 63 total 64 operations. s denotes the number of bits to shift. g is used to get the word from our 32-bit word block $M[j]$, $0 \leq j \leq 15$. We can pre calculate g . For each of the four rounds we can get the value of g based on i .

For round 1 $g = i$.

For round 2 $g = (5 - i + 1) \bmod 16$.

For round 3 $g = (3 - i + 1) \bmod 16$.

For round 4 $g = (7 - i) \bmod 16$.

Each of the rounds use one auxiliary function for each of its operation. After all operations the buffer AA, BB, CC, DD contains MD5 digest of our input message.

5. Output

After all the operations and all 4 previous steps are done. The buffer AA,BB, CC, DD contains MD5 digest of our input message.

Program:

```
//package com.sanfoundry.setandstring;

public class MD5

{
    private static final int    INIT_A      = 0x67452301;
    private static final int    INIT_B      = (int) 0xEFCDAB89L;
    private static final int    INIT_C      = (int) 0x98BADCFEL;
    private static final int    INIT_D      = 0x10325476;
    private static final int[]  SHIFT_AMTS  = { 7, 12, 17, 22, 5, 9, 14, 20,
4, 11, 16, 23, 6, 10, 15, 21    };

    private static final int[]  TABLE_T    = new int[64];

    static
    {
        for (int i = 0; i < 64; i++)
            TABLE_T[i] = (int) (long) ((1L << 32) * Math.abs(Math.sin(i +
1))) );
    }

    public static byte[] computeMD5(byte[] message)
    {
        int messageLenBytes = message.length;
        int numBlocks = ((messageLenBytes + 8) >>> 6) + 1;
        int totalLen = numBlocks << 6;
        byte[] paddingBytes = new byte[totalLen - messageLenBytes];
        paddingBytes[0] = (byte) 0x80;
        long messageLenBits = (long) messageLenBytes << 3;
        for (int i = 0; i < 8; i++)
        {
            paddingBytes[paddingBytes.length - 8 + i] = (byte)
messageLenBits;
            messageLenBits >>>= 8;
        }
        int a = INIT_A;
        int b = INIT_B;
        int c = INIT_C;
        int d = INIT_D;
        int[] buffer = new int[16];
        for (int i = 0; i < numBlocks; i++)
        {
```

```

int index = i << 6;
for (int j = 0; j < 64; j++, index++)
    buffer[j >>> 2] = ((int) ((index < messageLenBytes) ?
message[index]
        : paddingBytes[index - messageLenBytes]) << 24)
        | (buffer[j >>> 2] >>> 8);
int originalA = a;
int originalB = b;
int originalC = c;
int originalD = d;
for (int j = 0; j < 64; j++)
{
    int div16 = j >>> 4;
    int f = 0;
    int bufferIndex = j;
    switch (div16)
    {
        case 0:
            f = (b & c) | (~b & d);
            break;
        case 1:
            f = (b & d) | (c & ~d);
            bufferIndex = (bufferIndex * 5 + 1) & 0x0F;
            break;
        case 2:
            f = b ^ c ^ d;
            bufferIndex = (bufferIndex * 3 + 5) & 0x0F;
            break;
        case 3:
            f = c ^ (b | ~d);
            bufferIndex = (bufferIndex * 7) & 0x0F;
            break;
    }
    int temp = b+ Integer.rotateLeft(a + f + buffer[bufferIndex]
+ TABLE_T[j],
        SHIFT_AMTS[(div16 << 2) | (j & 3)]);
    a = d;
    d = c;
    c = b;
    b = temp;
}
a += originalA;
b += originalB;
c += originalC;
d += originalD;
}
byte[] md5 = new byte[16];
int count = 0;
for (int i = 0; i < 4; i++)
{

```

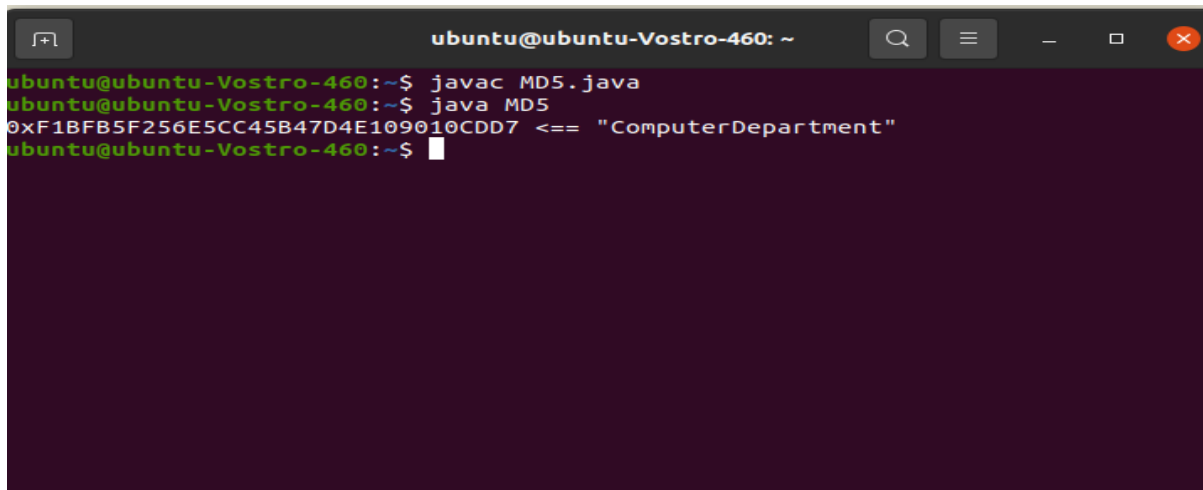
```

        int n = (i == 0) ? a : ((i == 1) ? b : ((i == 2) ? c : d));
        for (int j = 0; j < 4; j++)
        {
            md5[count++] = (byte) n;
            n >>>= 8;
        }
    }
    return md5;
}

public static String toHexString(byte[] b)
{
    StringBuilder sb = new StringBuilder();
    for (int i = 0; i < b.length; i++)
    {
        sb.append(String.format("%02X", b[i] & 0xFF));
    }
    return sb.toString();
}

public static void main(String[] args)
{
    String[] testStrings = { "ComputerDepartment" };
    for (String s : testStrings)
        System.out.println("0x" + toHexString(computeMD5(s.getBytes()))
            + " <= \"" + s + "\"");
    return;
}
}

```

Output:A terminal window titled 'ubuntu@ubuntu-Vostro-460: ~' with standard window controls. The terminal shows the following commands and output:

```
ubuntu@ubuntu-Vostro-460:~$ javac MD5.java
ubuntu@ubuntu-Vostro-460:~$ java MD5
0xF1BFB5F256E5CC45B47D4E109010CDD7 <== "ComputerDepartment"
ubuntu@ubuntu-Vostro-460:~$
```

Conclusion:

We have implemented MD5 algorithm using Java.

Oral questions:

1. Define a cryptographic hash function.
2. What is hash value? What is fingerprinting?
3. What is the difference between hashing and hash tables?
4. What is hash collisions?
5. What basic arithmetical and logical functions are used in MD5?
6. Is it possible to decrypt MD5 hashes?
7. Compared to other message digest algorithms, to what extent you think MD5 is stronger and Why?