# COMPUTER NETWORK SECURITY

**Project 2**

**MD5 Algorithm**

**Name: Karthikeyan Kumar** **A20376688**

## Contents

**Objective**

Implement the MD5 algorithm and evaluate the randomness of the output.

**Introduction**

MD5 is a message digest algorithm (hash function) that is responsible for producing the hash or the digest of a given message. The algorithm takes in an input message of any arbitrary length, and produces an output of specific length, 128 bits to be precise. This 128 bit output forms the digest/hash of the input message. It works on the base assumption that it is infeasible to produce two messages having the same digest value. It finds application in places where it is necessary to transmit data of large size, for instance say a large file, so the large file is first compressed in size by applying the MD5 algorithm and later on that compressed version is used for transmission. The difficulty of coming up with two messages with the same message digest is on the order of 2^64 operations.

**Algorithm**

1. Append the message with additional bits such that the length of the message is congruent to 448 mod 512. First append a single bit "1" to the message, and then append the zeros such that the total length is equal to 448 mod 512.
2. A 64 bit representation of the input message's length before the padding procedure is then appended to the padded message such that the resulting message is exactly multiple of 512 bits.
3. There are in total 64 rounds, and the number of shifts to be done in each round is specified by a variable. The keys that are employed for each round of the processing is also represented by a variable. The keys are generated as a sine function of the integers.
4. The hash value is calculated separately using four separate words each of 32 bit length, and each of those words are initialized to certain values represented in hexadecimal.a0=0x67452301, b0=0xefcdab89, c0=0x98badcfe, d0=0x10325476
5. The message is separated into blocks each of 512 bits long, and each block is processed in such a way that the 512 bits are broken into sixteen 32-bit words.
6. For each round, the computation procedure is carried out by four auxiliary functions that take three 32-bit word as input and produce a single 32-bit output, where each function is a bit wise operation.

   F(X, Y, Z) = (X&Y) | (~X & Z)

   G(X, Y, Z) = (X&Z) | (Y & ~Z)

   H(X, Y, Z) = X | Y | Z

   I(X, Y, Z) = Y | (X | ~Z)
7. Output of each function is summed with the hash value initialized at the beginning, the final hash value is a0 to d0 represented by the least significant bit of a0 till the most significant bit of d0.

**Observation**

From repeated trials for various inputs, it is found that if the input bits X, Y, Z to the function is independent of each other, then the function output is also independent of each other. Let us see how to hash values and percentage of ones change for the variations in the phrase “Hello World”.

**Hello World**

Number of ones in a0 is 10110001000010101000110110110001

Number of ones in b0 is 01100100111000000111010101000001

Number of ones in c0 is 00000101101101111010100110011011

Number of ones in d0 is 11100111001011100011111111100101

Total Bits set 65

Percentage of ones is 50.7812

The hash value is b10a8db164e075415b7a99be72e3fe5

Now let’s observe how the output hash values change for a single change in the bit ie from Hello World to Eello World.

**Iello World**

Number of ones in a0 is 00111001001001101010001010011110

Number of ones in b0 is 11000101010001010101000110011000

Number of ones in c0 is 11000100101110110011111010011000

Number of ones in d0 is 11110100101111011111011010100100

Total Bits set 65

Percentage of ones is 50.7812

The hash value is 3926a29ec5455198c4bb3e98f4bdf6a

If we observe closely there is a lot of bit flips in the hash value of a0, but the bit flips keep on reducing as we proceed to b0, c0, and least flips in d0. This shows that a single bit change in the input drastically affects the output only in the hash value where the single bit was flipped.

In order to better understand how the changes in bit affects the percentage of ones, let’s test it out with a long sentence and find the hash.

**The quick brown fox jumps over the lazy dog**

Number of ones in a0 is 10011110000100000111110110011101

Number of ones in b0 is 00110111001010111011011010000010

Number of ones in c0 is 01101011110110000001110100110101

Number of ones in d0 is 01000010101001000001100111010110

Total Bits set 63

Percentage of ones is 49.2188

The hash value is 9e107d9d372bb6826bd81d3542a419d6

**Uhe quick brown fox jumps over the lazy dog**

Number of ones in a0 is 00001010101111001110100010010111

Number of ones in b0 is 01101011010111101010111101111101

Number of ones in c0 is 11101110010111001100010001110000

Number of ones in d0 is 11110100001101011010111111110001

Total Bits set 74

Percentage of ones is 57.8125

The hash value is abce8976b5eaf7dee5cc470f435aff1

**When gravity falls and earth becomes sky beware of the beast with just one eye**

Number of ones in a0 is 10101010010010000101001101011010

Number of ones in b0 is 01011110001010011010101111101010

Number of ones in c0 is 10010100111110111101111001111010

Number of ones in d0 is 01111100110110001000111001001010

Total Bits set 69

Percentage of ones is 53.9062

The hash value is aa48535a5e29abea94fbde7a7cd88e4a

**Try over and over and over again then try again**

Number of ones in a0 is 11010001101111010001001011000010

Number of ones in b0 is 00000101001110000110100000010100

Number of ones in c0 is 01100111000010001110111111000010

Number of ones in d0 is 00010010100001011000001001001101

Total Bits set 52

Percentage of ones is 40.625

The hash value is d1bd12c253868146708efc21285824d

**Hey how do you do? isn't it a really nice day today! We should go out for a walk**

Number of ones in a0 is 10001000110000000111100111010111

Number of ones in b0 is 11111001000000101010111100111101

Number of ones in c0 is 10011000101010100110010001101101

Number of ones in d0 is 11010110010001001100010010000011

Total Bits set 61

Percentage of ones is 47.6562

The hash value is 88c079d7f902af3d98aa646dd644c483

**lolol bowbow barkbark chripchrip honkhonk**

Number of ones in a0 is 11000000100110110011101110001011

Number of ones in b0 is 11111000110011110101000110010010

Number of ones in c0 is 10100111001101001011001101111001

Number of ones in d0 is 10100011101011000011100101101000

Total Bits set 66

Percentage of ones is 51.5625

The hash value is c09b3b8bf8cf5192a734b379a3ac3968

**Lets have a great night on the dance floor**

Number of ones in a0 is 01110100000101001110111010001010

Number of ones in b0 is 11100101110011000010001001001110

Number of ones in c0 is 00100000010100001110011100001001

Number of ones in d0 is 00001001101000111000001011110011

Total Bits set 55

Percentage of ones is 42.9688

The hash value is 7414ee8ae5cc224e2050e7099a382f3

**Sun rises in the east,sets in the west. The Earth spins from east to west, the mars is the closest planet to our Earth,while we live in milky way**

Number of ones in a0 is 10011010011110010010001010011011

Number of ones in b0 is 00110101000011010100010101100100

Number of ones in c0 is 00001110100111100110110101101001

Number of ones in d0 is 00011001101100001101000110011010

Total Bits set 60

Percentage of ones is 46.875

The hash value is 9a79229b350d4564e9e6d6919b0d19a

**skadkjasbdkja kjasbdkjasbdkjabdkjassdkjas asjbjkadbakjsbdkjas asjkbdkajsbdjkaskj asbdkasbdja**

Number of ones in a0 is 01001110000000011000101110101000

Number of ones in b0 is 11011101011001111010101001011101

Number of ones in c0 is 10100001001011101001110011100000

Number of ones in d0 is 01011111010001010111000000010000

Total Bits set 59

Percentage of ones is 46.0938

The hash value is 4e018ba8dd67aa5da12e9ce05f457010

It is observed from all the test cases that the number of set bits are bounded between the percentages of 40 to 60. Even in the case of long sentences, it is within the range. It is due to the processing of the data after padding of zeros and hence maintains the value of set bits within that range. The average percentage of set bits is approximately 49%. Which is pushing fifty. It is observed that over a large collection of data, the percentage of set bits in the MD5 hash value tends to be 50%. This shows that the bit flips take place even if there is a small change in the input. But the frequent bit flips tend to nullify each other out.

**Simplification 1**

By reducing the number of hash functions we try to simplify the algorithm. The output for the same are

**The quick brown fox jumps over the lazy dog**

Number of ones in a0 is 10011111101000101010111001100000

Number of ones in b0 is 10001111110000000111100100000001

Number of ones in c0 is 11110000010011001110111110111010

Number of ones in d0 is 00001111101000101111110000001010

Total Bits set 63

Percentage of ones is 49.2188

The hash value is 9fa2ae608fc07901f04cefbafa2fc0a

**Uhe quick brown fox jumps over the lazy dog**

Number of ones in a0 is 01000110011010101100101101100010

Number of ones in b0 is 10100110100000100001000000001111

Number of ones in c0 is 10010011100001100011011110000010

Number of ones in d0 is 10000000100110001100001011001111

Total Bits set 53

Percentage of ones is 41.4062

The hash value is 466acb62a682100f938637828098c2cf

**When gravity falls and earth becomes sky beware of the beast with just one eye**

Number of ones in a0 is 00101001101000000001110100101110

Number of ones in b0 is 11100100111110100110000010100100

Number of ones in c0 is 01001110010111001011011010111011

Number of ones in d0 is 10100001001110010100011101001000

Total Bits set 60

Percentage of ones is 46.875

The hash value is 29a01d2ee4fa60a44e5cb6bba1394748 The number of hash function is reduced, so there isn't enough randomness in the computation of the hash values, therefore the flips in the bits are not as often as seen in the original algorithm. This may give way to vulnerabilities.

**Simplification 2**

Instead of varying the hash function for different intervals of the round, we can have it same for all the rounds.

**The quick brown fox jumps over the lazy dog**

Number of ones in a0 is 11001110101010110010111101100010

Number of ones in b0 is 01011001101110101101100011011111

Number of ones in c0 is 10001111010110111010101001110101

Number of ones in d0 is 10101001110110010011010001000110

Total Bits set 72

Percentage of ones is 56.25

The hash value is ceab2f6259bad8df8f5baa75a9d93446

**Uhe quick brown fox jumps over the lazy dog**

Number of ones in a0 is 11010000110000110110010110001001

Number of ones in b0 is 10110010000110100100111011001011

Number of ones in c0 is 00100101100010111000001101011101

```
Number of ones in d0 is 11010010110100100001110011101001

Total Bits set 61

Percentage of ones is 47.6562

The hash value is d0c36589b21a4ecb258b835dd2d21ce9
```

**When gravity falls and earth becomes sky beware of the beast with just one eye**

```
Number of ones in a0 is 11110000000000000100100100101111

Number of ones in b0 is 11110100111110110011111100000110

Number of ones in c0 is 11111101100110000111110000100100

Number of ones in d0 is 10000100101110110000001001010011

Total Bits set 62

Percentage of ones is 48.4375

The hash value is f000492ff4fb3f06fd987c2484bb0253.
```

The number of ones vary drastically, unlike the original algorithm, this is due to the lack of variety in the hash function defined. This allows the attacker to use different values of input and predict the output.

**PROGRAMS**

MD5 Original

```
#include<iostream>
#include<stdio.h>
#include<stdint.h>
#include<string>
#include<stdlib.h>
#include<cmath>
#include<vector>
#include<stdint.h>
#include<bitset>
using namespace std;
uint32_t reverse(uint32_t q);          //Function Declaration
unsigned int bitCount (unsigned int value);
int main()
{

//Number of per round shifts amount
const uint32_t shifts[64] = {7, 12, 17, 22, 7, 12, 17, 22, 7, 12, 17, 22,
7, 12, 17, 22,
                          5,  9, 14, 20, 5,  9, 14, 20, 5,  9, 14, 20, 5,
9, 14, 20,
                          4, 11, 16, 23, 4, 11, 16, 23, 4, 11, 16, 23, 4,
11, 16, 23,
                          6, 10, 15, 21, 6, 10, 15, 21, 6, 10, 15, 21, 6,
10, 15, 21};
//Sine value of Integers
const uint32_t keys[64] = {
0xd76aa478, 0xe8c7b756, 0x242070db, 0xc1bdceee ,
0xf57c0faf, 0x4787c62a, 0xa8304613, 0xfd469501 ,
0x698098d8, 0x8b44f7af, 0xffff5bb1, 0x895cd7be ,
0x6b901122, 0xfd987193, 0xa679438e, 0x49b40821 ,
0xf61e2562, 0xc040b340, 0x265e5a51, 0xe9b6c7aa ,
0xd62f105d, 0x02441453, 0xd8a1e681, 0xe7d3fbc8 ,
0x21e1cde6, 0xc33707d6, 0xf4d50d87, 0x455a14ed ,
0xa9e3e905, 0xfcefa3f8, 0x676f02d9, 0x8d2a4c8a ,
0xfffa3942, 0x8771f681, 0x6d9d6122, 0xfde5380c ,
0xa4beea44, 0x4bdecfa9, 0xf6bb4b60, 0xbebfbc70 ,
0x289b7ec6, 0xeaa127fa, 0xd4ef3085, 0x04881d05 ,
0xd9d4d039, 0xe6db99e5, 0x1fa27cf8, 0xc4ac5665 ,
0xf4292244, 0x432aff97, 0xab9423a7, 0xfc93a039 ,
0x655b59c3, 0x8f0ccc92, 0xffeff47d, 0x85845dd1 ,
0x6fa87e4f, 0xfe2ce6e0, 0xa3014314, 0x4e0811a1 ,
0xf7537e82, 0xbd3af235, 0x2ad7d2bb, 0xeb86d391 };

//Variables initialisation
uint32_t a0=0x67452301;     //A
uint32_t b0=0xefcdab89;        //B
uint32_t c0=0x98badcfe;        //C
uint32_t d0=0x10325476;        //D
uint32_t a, b, c, d, i, f, g, temp;
size_t len= 0;
float totalones=0;
vector <uint32_t> w;
#define left(x, c) (((x) << (c)) | ((x) >> (32 - (c)))); //Left shift
function

//Processing the input data,padding.
string ss;
getline(cin,ss);                  //Get input from user
vector <uint32_t> by(ss.begin(),ss.end());   //Initialise to a vector
len=by.size();
```

```
by.push_back(0x80);                     //Appending the bit '1'
for(int i=0; (by.size()*8)%512!=448;i++)      // Padding with zeros
{
    by.push_back(0x00);
}
by.push_back(len*8);                    //appending the length of the
original message

for(int i=0;(by.size()*8)%512!=0;i++)        //Appending zeros
{
    by.push_back(0x00);
}

//Breaking into blocks of 512-bit and each block is divided into 16 32-
bit word
len=by.size()/64;
int iterator=0,j=0,limit=0;
while(iterator<len){
j=limit;
limit=64*(iterator+1);
for(j=j; j<limit; j+=4)
{
        w.push_back(by[j+3]<<24|(by[j+2]<<16)|(by[j+1]<<8)|(by[j]));
//32-bit word, formed by shift operation
}

//Initial hash values
a=a0;
b=b0;
c=c0;
d=d0;

//Computing the hash values
for(int i=0;i<64;i++){
    if(i<16){
        f = (b & c) | ((~b) & d);
                g = i;
            } else if (i < 32) {
                f = (d & b) | ((~d) & c);
                g = (5*i + 1) % 16;
            } else if (i < 48) {
                f = b ^ c ^ d;
                g = (3*i + 5) % 16;
            } else {
                f = c ^ (b | (~d));
                g = (7*i) % 16;
            }

            temp = d;
            d = c;
            c = b;
            b = b + left((a+f+keys[i]+w[g]), shifts[i]);
            a = temp;
          //cout<<w[g]<<endl;
}
iterator++;
w.clear();
        // Adding the hash values:
        a0 += a;
        b0 += b;
        c0 += c;
        d0 += d;
}
totalones=bitCount(a0)+bitCount(b0)+bitCount(c0)+bitCount(d0);
a0=reverse(a0);
b0=reverse(b0);
c0=reverse(c0);
d0=reverse(d0);
cout<<"number of ones in a0 is "<<bitset<32>(a0)<<endl;
cout<<"number of ones in b0 is "<<bitset<32>(b0)<<endl;
cout<<"number of ones in c0 is "<<bitset<32>(c0)<<endl;
cout<<"number of ones in d0 is "<<bitset<32>(d0)<<endl;
cout<<"Total Bits set "<<totalones<<endl;
cout<<"Percentage of ones is "<<float((totalones*100)/128)<<endl;
cout<<"a0 is "<<hex<<a0<<endl;
cout<<"b0 is "<<hex<<b0<<endl;
cout<<"c0 is "<<hex<<c0<<endl;
cout<<"d0 is "<<hex<<d0<<endl;
cout<<"The hash value is " <<hex<<a0<<hex<<b0<<hex<<c0<<hex<<d0<<endl;
      //Final hash values
}
uint32_t reverse(uint32_t q)
{uint32_t z=q;
return
(((z>>24)&0xff)|(((z>>16)&0xff)<<8)|(((z>>8)&0xff)<<16)|(((z)&0xff)<<24))
;
}
uint32_t bitCount(uint32_t n)
{
    size_t count = 0;
    while (n)
    {
      n &= (n-1) ;
      count++;
    }
    return count;
}
```

Simplification 1

```
#include<iostream>
#include<stdio.h>
#include<stdint.h>
#include<string>
#include<stdlib.h>
#include<cmath>
#include<vector>
#include<stdint.h>
#include<bitset>
using namespace std;
uint32_t reverse(uint32_t q);           //Function Declaration
unsigned int bitCount (unsigned int value);
int main()
{

//Number of per round shifts amount
const uint32_t shifts[64] = {7, 12, 17, 22, 7, 12, 17, 22, 7, 12, 17, 22,
7, 12, 17, 22,
                           5,  9, 14, 20, 5,  9, 14, 20, 5,  9, 14, 20, 5,
9, 14, 20,
                           4, 11, 16, 23, 4, 11, 16, 23, 4, 11, 16, 23, 4,
11, 16, 23,
                           6, 10, 15, 21, 6, 10, 15, 21, 6, 10, 15, 21, 6,
10, 15, 21};
//Sine value of Integers
const uint32_t keys[64] = {
0xd76aa478, 0xe8c7b756, 0x242070db, 0xc1bdceee ,
0xf57c0faf, 0x4787c62a, 0xa8304613, 0xfd469501 ,
0x698098d8, 0x8b44f7af, 0xffff5bb1, 0x895cd7be ,
0x6b901122, 0xfd987193, 0xa679438e, 0x49b40821 ,
0xf61e2562, 0xc040b340, 0x265e5a51, 0xe9b6c7aa ,
0xd62f105d, 0x02441453, 0xd8a1e681, 0xe7d3fbc8 ,
0x21e1cde6, 0xc33707d6, 0xf4d50d87, 0x455a14ed ,
0xa9e3e905, 0xfcefa3f8, 0x676f02d9, 0x8d2a4c8a ,
0xfffa3942, 0x8771f681, 0x6d9d6122, 0xfde5380c ,
0xa4beea44, 0x4bdecfa9, 0xf6bb4b60, 0xbebfbc70 ,
0x289b7ec6, 0xeaa127fa, 0xd4ef3085, 0x04881d05 ,
0xd9d4d039, 0xe6db99e5, 0x1fa27cf8, 0xc4ac5665 ,
0xf4292244, 0x432aff97, 0xab9423a7, 0xfc93a039 ,
0x655b59c3, 0x8f0ccc92, 0xffeff47d, 0x85845dd1 ,
0x6fa87e4f, 0xfe2ce6e0, 0xa3014314, 0x4e0811a1 ,
0xf7537e82, 0xbd3af235, 0x2ad7d2bb, 0xeb86d391 };

//Variables initialisation
uint32_t a0=0x67452301;     //A
uint32_t b0=0xefcdab89;        //B
uint32_t c0=0x98badcfe;        //C
uint32_t d0=0x10325476;        //D
uint32_t a, b, c, d, i, f, g, temp;
size_t len= 0;
float totalones=0;
vector <uint32_t> w;
#define left(x, c) (((x) << (c)) | ((x) >> (32 - (c)))); //Left shift
function

//Processing the input data,padding.
string ss;
getline(cin,ss);                    //Get input from user
vector <uint32_t> by(ss.begin(),ss.end());   //Initialise to a vector
len=by.size();
```

```
by.push_back(0x80);                     //Appending the bit '1'
for(int i=0; (by.size()*8)%512!=448;i++)      // Padding with zeros
{
    by.push_back(0x00);
}
by.push_back(len*8);                    //appending the length of the
original message

for(int i=0;(by.size()*8)%512!=0;i++)        //Appending zeros
{
    by.push_back(0x00);
}

//Breaking into blocks of 512-bit and each block is divided into 16 32-
bit word
len=by.size()/64;
int iterator=0,j=0,limit=0;
while(iterator<len){
j=limit;
limit=64*(iterator+1);
for(j=j; j<limit; j+=4)
{
        w.push_back(by[j+3]<<24|(by[j+2]<<16)|(by[j+1]<<8)|(by[j]));
//32-bit word, formed by shift operation
}

//Initial hash values
a=a0;
b=b0;
c=c0;
d=d0;

//Computing the hash values
for(int i=0;i<64;i++){
    if(i<32){
        f = (b & c) | ((~b) & d);
                g = i;
            } //else if (i < 32) {
                //f = (d & b) | ((~d) & c);
                //g = (5*i + 1) % 16;
             //else if (i < 48) {
                //f = b ^ c ^ d;
                //g = (3*i + 5) % 16;
             else {
                f = c ^ (b | (~d));
                g = (7*i) % 16;
            }

            temp = d;
            d = c;
            c = b;
            b = b + left((a+f+keys[i]+w[g]), shifts[i]);
            a = temp;
          //cout<<w[g]<<endl;
}
iterator++;
w.clear();
        // Adding the hash values:
        a0 += a;
        b0 += b;
        c0 += c;
        d0 += d;
}

totalones=bitCount(a0)+bitCount(b0)+bitCount(c0)+bitCount(d0);
a0=reverse(a0);
b0=reverse(b0);
c0=reverse(c0);
d0=reverse(d0);
cout<<"Number of ones in a0 is "<<bitset<32>(a0)<<endl;
cout<<"Number of ones in b0 is "<<bitset<32>(b0)<<endl;
cout<<"Number of ones in c0 is "<<bitset<32>(c0)<<endl;
cout<<"Number of ones in d0 is "<<bitset<32>(d0)<<endl;
cout<<"Total Bits set "<<totalones<<endl;
cout<<"Percentage of ones is "<<float((totalones*100)/128)<<endl;
cout<<"a0 is "<<hex<<a0<<endl;
cout<<"b0 is "<<hex<<b0<<endl;
cout<<"c0 is "<<hex<<c0<<endl;
cout<<"d0 is "<<hex<<d0<<endl;
cout<<"The hash value is " <<hex<<a0<<hex<<b0<<hex<<c0<<hex<<d0<<endl;
      //Final hash values
}
uint32_t reverse(uint32_t q)
{
return
(((q>>24)&0xff)|(((q>>16)&0xff)<<8)|(((q>>8)&0xff)<<16)|(((q)&0xff)<<24))
;
}
uint32_t bitCount(uint32_t n)
{
    size_t count = 0;
    while (n)
    {
      n &= (n-1) ;
      count++;
    }
    return count;
}
```

```
#include<iostream>
#include<stdio.h>
#include<stdint.h>
#include<string>
#include<stdlib.h>
#include<cmath>
#include<vector>
#include<stdint.h>
#include<bitset>
using namespace std;
uint32_t reverse(uint32_t q);          //Function Declaration
unsigned int bitCount (unsigned int value);
int main()
{

//Number of per round shifts amount
const uint32_t shifts[64] = {7, 12, 17, 22, 7, 12, 17, 22, 7, 12, 17, 22,
7, 12, 17, 22,
                          5,  9, 14, 20, 5,  9, 14, 20, 5,  9, 14, 20, 5,
9, 14, 20,
                          4, 11, 16, 23, 4, 11, 16, 23, 4, 11, 16, 23, 4,
11, 16, 23,
                          6, 10, 15, 21, 6, 10, 15, 21, 6, 10, 15, 21, 6,
10, 15, 21};
//Sine value of Integers
const uint32_t keys[64] = {
0xd76aa478, 0xe8c7b756, 0x242070db, 0xc1bdceee ,
0xf57c0faf, 0x4787c62a, 0xa8304613, 0xfd469501 ,
0x698098d8, 0x8b44f7af, 0xffff5bb1, 0x895cd7be ,
0x6b901122, 0xfd987193, 0xa679438e, 0x49b40821 ,
0xf61e2562, 0xc040b340, 0x265e5a51, 0xe9b6c7aa ,
0xd62f105d, 0x02441453, 0xd8a1e681, 0xe7d3fbc8 ,
0x21e1cde6, 0xc33707d6, 0xf4d50d87, 0x455a14ed ,
0xa9e3e905, 0xfcefa3f8, 0x676f02d9, 0x8d2a4c8a ,
0xfffa3942, 0x8771f681, 0x6d9d6122, 0xfde5380c ,
0xa4beea44, 0x4bdecfa9, 0xf6bb4b60, 0xbebfbc70 ,
0x289b7ec6, 0xeaa127fa, 0xd4ef3085, 0x04881d05 ,
0xd9d4d039, 0xe6db99e5, 0x1fa27cf8, 0xc4ac5665 ,
0xf4292244, 0x432aff97, 0xab9423a7, 0xfc93a039 ,
0x655b59c3, 0x8f0ccc92, 0xffeff47d, 0x85845dd1 ,
0x6fa87e4f, 0xfe2ce6e0, 0xa3014314, 0x4e0811a1 ,
0xf7537e82, 0xbd3af235, 0x2ad7d2bb, 0xeb86d391 };

//Variables initialisation
uint32_t a0=0x67452301;     //A
uint32_t b0=0xefcdab89;        //B
uint32_t c0=0x98badcfe;        //C
uint32_t d0=0x10325476;        //D
uint32_t a, b, c, d, i, f, g, temp;
size_t len= 0;
float totalones=0;
vector <uint32_t> w;
#define left(x, c) (((x) << (c)) | ((x) >> (32 - (c)))); //Left shift
function

//Processing the input data,padding.
string ss;
getline(cin,ss);                     //Get input from user
vector <uint32_t> by(ss.begin(),ss.end());   //Initialise to a vector
len=by.size();
```

```
by.push_back(0x80);                     //Appending the bit '1'
for(int i=0; (by.size()*8)%512!=448;i++)      // Padding with zeros
{
    by.push_back(0x00);
}
by.push_back(len*8);                    //appending the length of the
original message

for(int i=0;(by.size()*8)%512!=0;i++)         //Appending zeros
{
    by.push_back(0x00);
}

//Breaking into blocks of 512-bit and each block is divided into 16 32-
bit word
len=by.size()/64;
int iterator=0,j=0,limit=0;
while(iterator<len){
j=limit;
limit=64*(iterator+1);
for(j=j; j<limit; j+=4)
{
        w.push_back(by[j+3]<<24|(by[j+2]<<16)|(by[j+1]<<8)|(by[j]));
//32-bit word, formed by shift operation
}

//Initial hash values
a=a0;
b=b0;
c=c0;
d=d0;

//Computing the hash values
for(int i=0;i<64;i++){
    if(i<16){
        f = (b & c) | ((~b) & d);
                g = i;
            } else if (i < 32) {
                f = (d & b) | ((~d) & c);
                g = i;
            } else if (i < 48) {
                f = b ^ c ^ d;
                g = i;
            } else {
                f = c ^ (b | (~d));
                g = i;
            }

            temp = d;
            d = c;
            c = b;
            b = b + left((a+f+keys[i]+w[g]), shifts[i]);
            a = temp;
          //cout<<w[g]<<endl;
}
iterator++;
w.clear();
        // Adding the hash values:
        a0 += a;
        b0 += b;
        c0 += c;
        d0 += d;
}
totalones=bitCount(a0)+bitCount(b0)+bitCount(c0)+bitCount(d0);
a0=reverse(a0);
b0=reverse(b0);
c0=reverse(c0);
d0=reverse(d0);
cout<<"number of ones in a0 is "<<bitset<32>(a0)<<endl;
cout<<"number of ones in b0 is "<<bitset<32>(b0)<<endl;
cout<<"number of ones in c0 is "<<bitset<32>(c0)<<endl;
cout<<"number of ones in d0 is "<<bitset<32>(d0)<<endl;
cout<<"Total Bits set "<<totalones<<endl;
cout<<"Percentage of ones is "<<((totalones*100)/128)<<endl;
cout<<"a0 is "<<hex<<a0<<endl;
cout<<"b0 is "<<hex<<b0<<endl;
cout<<"c0 is "<<hex<<c0<<endl;
cout<<"d0 is "<<hex<<d0<<endl;
cout<<"The hash value is " <<hex<<a0<<hex<<b0<<hex<<c0<<hex<<d0<<endl;
      //Final hash values
}
uint32_t reverse(uint32_t q)
{
return
(((q>>24)&0xff)|(((q>>16)&0xff)<<8)|(((q>>8)&0xff)<<16)|(((q)&0xff)<<24))
;
}
uint32_t bitCount(uint32_t n)
{
    size_t count = 0;
    while (n)
    {
      n &= (n-1) ;
      count++;
    }
    return count;
}
```