**Garvin Chanderia**

**2203132**

**LY-ISA1**

**Experiment No:5A**

**Title: MD5 Algorithm**

After completion of this experiment students will be able to:

* verify and maintain the integrity of information.

**Aim:** Implement the MD5 hashing technique

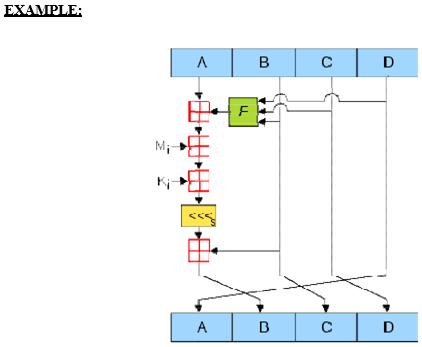
**Theory:**

MD5 processes a varia le-length message into a fixed-length output of 128 bits. The input message is broken up into chunks of 512-bit blocks. 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 64 bits representing the length of the original message, modulo 264.The main MD5 algorithm operates on a 128-bit state, divided into four 32-bit words, denoted A, B, C, and D. These are initialized to certain fixed constants. The main algorithm then uses each 512-bit message block in turn to modify the state.

b

d

M



**Algorithm/Pseudocode:**

**STEP-1:** Read the 128-bit plain text.

**STEP-2:** Divide into four blocks of 32-bits named as A, B, C and D.

**STEP-3:** Compute the functions f, g, h and i with operations such as, rotations, permutations, etc,.

**STEP-4:** The output of these functions are combined together as F and performed circular shifting and then given to key round.

**STEP-5:** Finally, right shift of ‘s’ times are performed and the results are combined together to produce the final output.

**Program:**

**#include<stdlib.h> #include <stdio.h> #include <string.h> #include <math.h> #include<conio.h> typedef union uwb**

**{**

**unsigned w; unsigned char b[4];**

**} MD5union;**

**typedef unsigned DigestArray[4]; unsigned func0( unsigned abcd[] ){**

**return ( abcd[1] & abcd[2]) | (~abcd[1] & abcd[3]);} unsigned func1( unsigned abcd[] ){**

**return ( abcd[3] & abcd[1]) | (~abcd[3] & abcd[2]);} unsigned func2( unsigned abcd[] ){**

**return abcd[1] ^ abcd[2] ^ abcd[3];} unsigned func3( unsigned abcd[] ){ return abcd[2] ^ (abcd[1] |~ abcd[3]);}**

**typedef unsigned (\*DgstFctn)(unsigned a[]); unsigned \*calctable( unsigned \*k)**

**{**

**double s, pwr; int i;**

**pwr = pow( 2, 32); for (i=0; i<64; i++)**

**{**

**s = fabs(sin(1+i));**

**k[i] = (unsigned)( s \* pwr );**

**}**

**return k;**

**}**

**unsigned rol( unsigned r, short N )**

**{**

**unsigned mask1 = (1<<N) -1;**

**return ((r>>(32-N)) & mask1) | ((r<<N) & ~mask1);**

**}**

**unsigned \*md5( const char \*msg, int mlen)**

**{**

**static DigestArray h0 = { 0x67452301, 0xEFCDAB89, 0x98BADCFE, 0x10325476 };**

**static DgstFctn ff[] = { &func0, &func1, &func2, &func3}; static short M[] = { 1, 5, 3, 7 };**

**static short O[] = { 0, 1, 5, 0 }; static short rot0[] = { 7,12,17,22}; static short rot1[] = { 5, 9,14,20}; static short rot2[] = { 4,11,16,23}; static short rot3[] = { 6,10,15,21};**

**static short \*rots[] = {rot0, rot1, rot2, rot3 }; static unsigned kspace[64];**

**static unsigned \*k; static DigestArray h; DigestArray abcd; DgstFctn fctn;**

**short m, o, g; unsigned f; short \*rotn; union**

**{**

**unsigned w[16]; char [64];**

**int os = 0;**

**int grp, grps, q, p;**

**unsigned char \*msg2;**

**if (k==NULL) k= calctable(kspace);**

**for (q=0; q<4; q++) h[q] = h0[q]; // initialize**

**{**

**grps = 1 + (mlen+8)/64; msg2 = malloc( 64\*grps); memcpy( msg2, msg, mlen);**

**msg2[mlen] = (unsigned char)0x80; q = mlen + 1;**

**while (q < 64\*grps){ msg2[q] = 0; q++ ; }**

**{**

**MD5union u;**

**u.w = 8\*mlen; q -= 8;**

**memcpy(msg2+q, &u.w, 4 );**

**}**

**}**

**for (grp=0; grp<grps; grp++)**

**{**

**memcpy( mm.b, msg2+os, 64);**

**for(q=0;q<4;q++) abcd[q] = h[q]; for (p = 0; p<4; p++)**

**{**

**fctn = ff[p]; rotn = rots[p];**

**m = M[p]; o= O[p];**

**for (q=0; q<16; q++)**

**{**

**g = (m\*q + o) % 16;**

**f = abcd[1] + rol( abcd[0]+ fctn(abcd)+k[q+16\*p]**

**+ mm.w[g], rotn[q%4]); abcd[0] = abcd[3];**

**abcd[3] = abcd[2];**

**abcd[2] = abcd[1]; abcd[1] = f;**

**}}**

**for (p=0; p<4; p++) h[p] += abcd[p];**

**os += 64;**

**}**

**return h;} void main()**

**{**

**int j,k;**

**const char \*msg = "The quick brown fox jumps over the lazy dog";**

**unsigned \*d = md5(msg, strlen(msg)); MD5union u;**

**clrscr();**

**printf("\t MD5 ENCRYPTION ALGORITHM IN C \n\n");**

**printf("Input String to be Encrypted using MD5 :**

**\n\t%s",msg);**

**printf("\n\nThe MD5 code for input string is: \n"); printf("\t= 0x");**

**for (j=0;j<4; j++){**

**u.w = d[j];**

**for (k=0;k<4;k++) printf("%02x",u.b[k]);**

**}**

**printf("\n");**

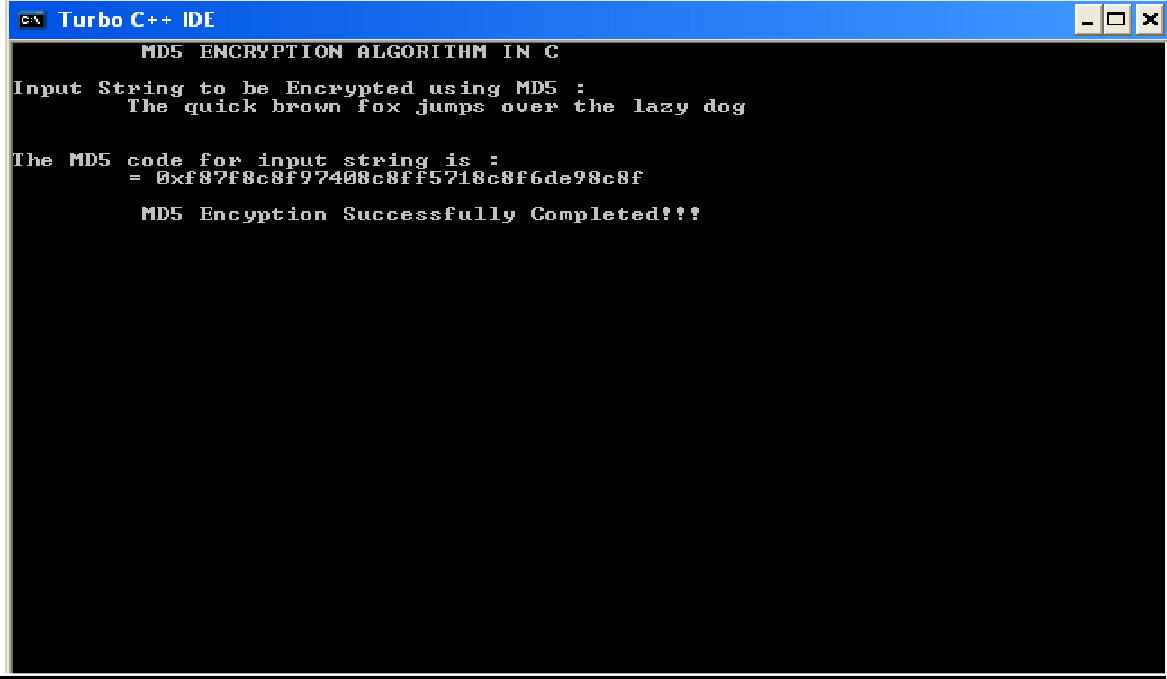
**printf("\n\t MD5 Encyption Successfully Completed!!!\n\n");**

**getch(); system("pause");**

**getch();**

**}**

**OUTPUT:**



**Conclusion: We have studied and implemented MD5 hashing technique.**

**Garvin Chanderia**

**2203132**

**LY-ISA1**

**Experiment No:5B**

**Title: SHA- Algorithm**

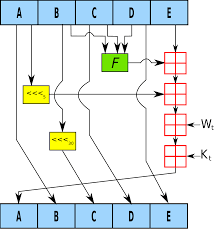
After completion of this experiment students will be able to:

* verify and maintain the integrity of information.

**Aim:** Implement the SHA-1 hashing technique

**Theory:**

In cryptography, SHA-1 (Secure Hash Algorithm 1) is a cryptographic hash function. SHA-1 produces a 160-bit hash value known as a message digest. The way this algorithm works is that for a message of size < 264 bits it computes a 160-bit condensed output called a message digest. The SHA-1 algorithm is designed so that it is practically infeasible to find two input messages that hash to the same output message. A hash function such as SHA-1 is used to calculate an alphanumeric string that serves as the cryptographic representation of a file or a piece of data. This is called a digest and can serve as a digital signature. It is supposed to be unique and non-reversible.

**EXAMPLE:**

**ALGORITHM:**

**STEP-1:** Read the 256-bit key values.

**STEP-2:** Divide into five equal-sized blocks named A, B, C, D and E.

**STEP-3:** The blocks B, C and D are passed to the function F.

**STEP-4:** The resultant value is permuted with block E.

**STEP-5:** The block A is shifted right by ‘s’ times and permuted with the result of step-4.

**STEP-6:** Then it is permuted with a weight value and then with some other key pair and taken as the first block.

**STEP-7:** Block A is taken as the second block and the block B is shifted by ‘s’ times and taken as the third block.

**STEP-8:** The blocks C and D are taken as the block D and E for the final output.

**PROGRAM: (Secure Hash Algorithm)**

**import java.security.\*; public class SHA1 {**

**public static void main(String[] a) { try {**

**MessageDigest md = MessageDigest.getInstance("SHA1"); System.out.println("Message digest object info: "); System.out.println(" Algorithm = " +md.getAlgorithm()); System.out.println(" Provider = " +md.getProvider()); System.out.println(" ToString = " +md.toString()); String input = "";**

**md.update(input.getBytes()); byte[] output = md.digest(); System.out.println();**

**System.out.println("SHA1(\""+input+"\") =**

**+bytesToHex(output)); input = "abc";**

**md.update(input.getBytes()); output = md.digest(); System.out.println();**

**System.out.println("SHA1(\""+input+"\") = "**

**+bytesToHex(output));**

**input = "abcdefghijklmnopqrstuvwxyz"; md.update(input.getBytes());**

**output = md.digest(); System.out.println();**

**System.out.println("SHA1(\"" +input+"\") = "**

**+bytesToHex(output)); System.out.println(""); } catch (Exception e) {**

**System.out.println("Exception: " +e);**

**}**

**}**

**public static String bytesToHex(byte[] b)**

**{**

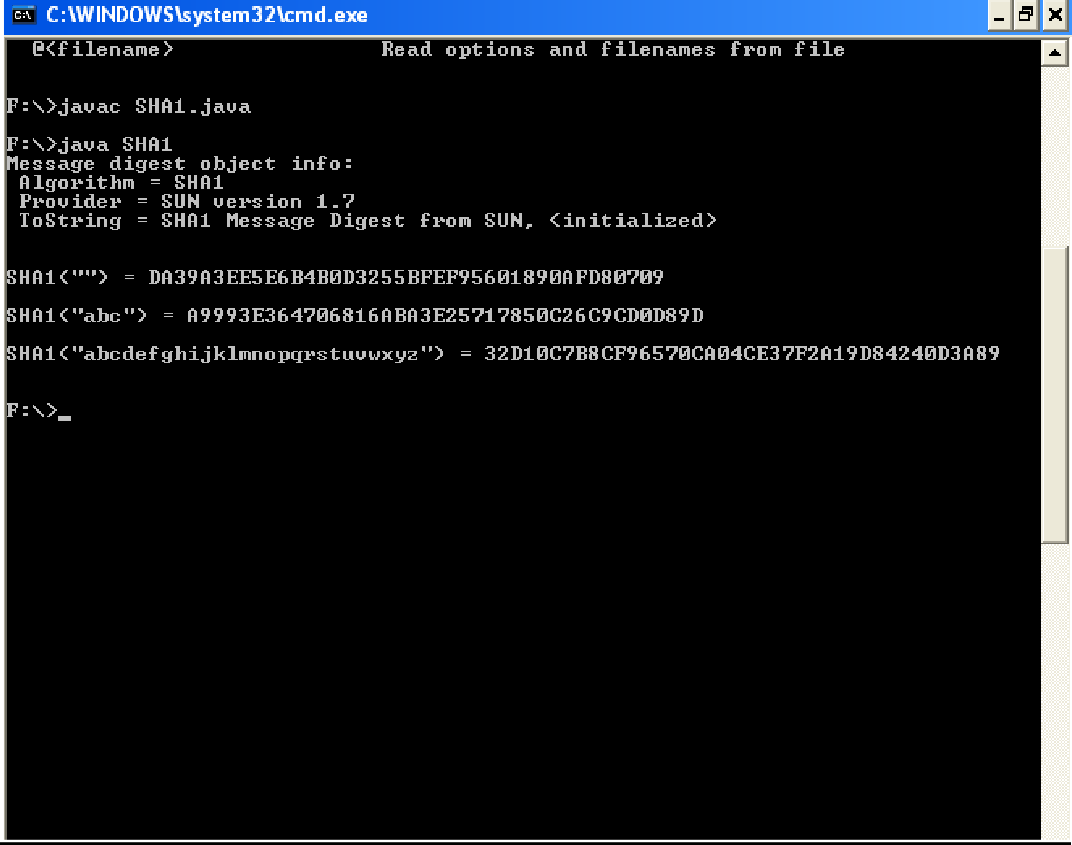
**char hexDigit[] = {'0', '1', '2', '3', '4', '5', '6',**

**'7', '8', '9', 'A', 'B', 'C', 'D', 'E', 'F'};**

**StringBuffer buf = new StringBuffer(); for (int j=0; j<b.length; j++) {**

**buf.append(hexDigit[(b[j] >> 4) & 0x0f]); buf.append(hexDigit[b[j] & 0x0f]); } return buf.toString(); }**

**}**

**OUTPUT:**