DESIGN AND ANALYSIS OF ALGORITHMS PROJECT

TOPIC: CRYPTOGRAPHIC ALGORITHMS

AMRUT S (PES1201701505)

CHALLA DHEERAJ REDDY (PES1201700830)

ABSTRACT:

Cryptography is the practice and study of techniques for securing communication and data in the presence of adversaries. Encryption is essentially important because it secures data and information from unauthorized access and thus maintains the confidentiality. Cryptography is broadly classified into two categories: ***Symmetric key Cryptography*** and ***Asymmetric key Cryptography*** (popularly known as public key cryptography). Now Symmetric key Cryptography is further categorized as Classical Cryptography and Modern Cryptography.

->Symmetric Key Cryptography

*An **encryption** system in which the sender and receiver of a message share a single, common **key** that is used to encrypt and decrypt the message. The most popular **symmetric–key** system is the Data **Encryption** Standard (DES)*

->Asymmetric Key Encryption (or Public Key Cryptography)

The encryption process where different keys are used for encrypting and decrypting the information. Keys are different but are mathematically related, such that retrieving the plain text by decrypting ciphertext is feasible.

ALGORITHMS:

1)ASCII ALGORITHM

Summary:

While encrypting the given string, 3 is added to the ASCII value of the characters. Similarly, for decrypting the string, 3 is subtracted from the ASCII value of the characters to print an original string.

CODE:

```
#include <stdio.h>

int main()
{
   int i, x;
   char str[100];

   printf("\nPlease enter a string:\t");
   gets(str);
```

```
   printf("\nPlease choose following options:\n");
   printf("1 = Encrypt the string.\n");
   printf("2 = Decrypt the string.\n");
   scanf("%d", &x);

   //using switch case statements
   switch(x)
   {
   case 1:
      for(i = 0; (i < 100 && str[i] != '\0'); i++)
        str[i] = str[i] + 3; //the key for encryption is 3 that is added to ASCII
value

      printf("\nEncrypted string: %s\n", str);
      break;

   case 2:
      for(i = 0; (i < 100 && str[i] != '\0'); i++)
        str[i] = str[i] - 3; //the key for encryption is 3 that is subtracted to ASCII
value

      printf("\nDecrypted string: %s\n", str);
      break;

   default:
      printf("\nError\n");
   }
   return 0;
}
```

2)MORSE ALGORITHM

Summary:

Morse code is a method of transmitting text information as a series of on-off tones, lights, or clicks that can be directly understood by a skilled listener or observer without special equipment. It is named for Samuel F. B. Morse, an inventor of the telegraph.
The algorithm is very simple. Every character in the English language is substituted by a series of ‘dots’ and ‘dashes’ or sometimes just singular ‘dot’ or ‘dash’ and vice versa.

Every text string is converted into the series of dots and dashes. For this every character is converted into its Morse code and appended in encoded message. Here we have copied space as it is. And we have not considered numbers, only alphabets.

CODE:

//Encryption Code

```
#include<stdio.h>
#include<string.h>
#include<ctype.h>
main()
{
 char *alphamorse[]={".-","-...","-.-.","-..",".","..-.","--.","....","..",".---",
                     "-.-",".-..","--","-.","---",".--.","--.-",".-.","...","-","..-",
     "...-",".--","-..-","-.--","--.."};
 char *nummorse[]={"-----",".----","..---","...--","....-",".....","-....","--...","--
-..","----."};

 int i;
 char str[1000],str1[1000];
 printf("Enter a sentence\n");
 gets(str);
 i=0;
  while(str[i]!='\0')
  {
 if(str[i]!=' '&&(!isdigit(str[i])))
 {
     printf("%s ",alphamorse[toupper(str[i])-65]);
 }
    if(str[i]==' ')
    {
     printf(" ");
    }
    if(isdigit(str[i])&&str[i]!=' ')
    {
      printf("%s ",nummorse[str[i]-48]);
    }

   i++;
  }
  printf("\n");

}
```

//Decryption Code

```
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
main()
{
    int i=0,j=0,k=0,flag=0;
    char *alphamorse[]={".-","-...","-.-.","-..",".","..-.","--.","....","..",".---",
                     "-.-",".-..","--","-.","---",".--.","--.-",".-.","...","-","..-",
     "...-",".--","-..-","-.--","--.."};
 char *nummorse[]={"-----",".----","..---","...--","....-",".....","-....","--...","--
-..","----."};

    char morseCode[1000]={0},substr[1000][100]={0},c;
    printf("Enter a Morse Code with spaces\n");
    gets(morseCode);
    while(morseCode[k]!='\0')//for splitting sentence
    {
        j=0;
        while(morseCode[k]!=' '&&morseCode[k]!='\0')
        {
            substr[i][j]=morseCode[k];
            k++;
```

```
            j++;
        }
        substr[i][j]='\0';
        i++;
        if(morseCode[k]!='\0')
        {
            k++;
        }
    }
    int len=i;
 for(i=0;i<len;i++)
 {
  for(j=0;j<25;j++)
  {
   if(!strcmp(alphamorse[j],substr[i]))//strcmp returns 0 if they are same
   {
    printf("%c",(j+65));
    break;
   }
  }
  for(j=0;j<9;j++)
  {
   if(!strcmp(nummorse[j],substr[i]))
   {
    printf("%d",j);
    break;
   }
  }
 }
 printf("\n");
}
```

3)RSA ALGORITHM

Summary:

RSA algorithm is asymmetric cryptography algorithm. Asymmetric actually means that it works on two different keys i.e. **Public Key** and **Private Key.** As the name describes that the Public Key is given to everyone and Private key is kept private. An example of asymmetric cryptography :

1. A client (for example browser) sends its public key to the server and requests for some data.
2. The server encrypts the data using client's public key and sends the encrypted data.
3. Client receives this data and decrypts it.

Since this is asymmetric, nobody else except browser can decrypt the data even if a third party has public key of browser.

The idea of RSA is based on the fact that it is difficult to factorize a large integer. The public key consists of two numbers where one number is multiplication of two large prime numbers. And private key is also derived from the same two prime numbers. So if somebody can factorize the large number, the private key is compromised. Therefore encryption strength totally lies on the key size and if we double or triple the key size, the strength of encryption increases exponentially. RSA keys can be

typically 1024 or 2048 bits long, but experts believe that 1024 bit keys could be broken in the near future. But till now it seems to be an infeasible task.

CODE:

```
#include<stdio.h>
#include<stdlib.h>
#include<math.h>
#include<string.h>

int x, y, n, t, i, flag;
long int e[50], d[50], temp[50], j, m[50], en[50];
char msg[100];
int prime(long int);
void encryption_key();
long int cd(long int);
void encrypt();
void decrypt();

int main()
{
  printf("\nENTER FIRST PRIME NUMBER\n");
  scanf("%d", &x);
  flag = prime(x);
  if(flag == 0)
  {
    printf("\nINVALID INPUT\n");
    exit(0);
  }
  printf("\nENTER SECOND PRIME NUMBER\n");
  scanf("%d", &y);
  flag = prime(y);
  if(flag == 0 || x == y)
  {
    printf("\nINVALID INPUT\n");
    exit(0);
  }
  printf("\nENTER MESSAGE OR STRING TO ENCRYPT\n");

  scanf("%s",msg);
  for(i = 0; msg[i] != NULL; i++)
    m[i] = msg[i];
  n = x * y;
  t = (x-1) * (y-1);
  encryption_key();
  printf("\nPOSSIBLE VALUES OF e AND d ARE\n");
  for(i = 0; i < j-1; i++)
    printf("\n%ld\t%ld", e[i], d[i]);
  encrypt();
  decrypt();
  return 0;
}
int prime(long int pr)
{
  int i;
  j = sqrt(pr);
  for(i = 2; i <= j; i++)
```

```
  {
   if(pr % i == 0)
     return 0;
  }
  return 1;
 }

//function to generate encryption key
void encryption_key()
{
  int k;
  k = 0;
  for(i = 2; i < t; i++)
  {
    if(t % i == 0)
     continue;
    flag = prime(i);
    if(flag == 1 && i != x && i != y)
    {
     e[k] = i;
     flag = cd(e[k]);
    if(flag > 0)
    {
     d[k] = flag;
     k++;
    }
   if(k == 99)
    break;
   }
 }
}
long int cd(long int a)
{
  long int k = 1;
  while(1)
  {
    k = k + t;
    if(k % a == 0)
     return(k / a);
  }
}

//function to encrypt the message
void encrypt()
{
  long int pt, ct, key = e[0], k, len;
  i = 0;
  len = strlen(msg);
  while(i != len)
  {
    pt = m[i];
    pt = pt - 96;
    k = 1;
    for(j = 0; j < key; j++)
    {
     k = k * pt;
     k = k % n;
    }
   temp[i] = k;
   ct = k + 96;
   en[i] = ct;
   i++;
```

```
  }
  en[i] = -1;
  printf("\n\nTHE ENCRYPTED MESSAGE IS\n");
  for(i = 0; en[i] != -1; i++)
    printf("%c", en[i]);
}

//function to decrypt the message
void decrypt()
{
  long int pt, ct, key = d[0], k;
  i = 0;
  while(en[i] != -1)
  {
    ct = temp[i];
    k = 1;
    for(j = 0; j < key; j++)
    {
      k = k * ct;
      k = k % n;
    }
   pt = k + 96;
   m[i] = pt;
   i++;
  }
  m[i] = -1;
  printf("\n\nTHE DECRYPTED MESSAGE IS\n");
  for(i = 0; m[i] != -1; i++)
   printf("%c", m[i]);
  printf("\n");
}
```

4)MD5 ALGORITHM

Summary:

The **MD5 message-digest algorithm** is a widely used hash function producing a 128-bit hash value. Although MD5 was initially designed to be used as a cryptographic hash function, it has been found to suffer from extensive vulnerabilities. It can still be used as a checksum to verify data integrity, but only against unintentional corruption. It remains suitable for other non-cryptographic purposes, for example for determining the partition for a particular key in a partitioned database.[3]

One basic requirement of any cryptographic hash function is that it should be computationally infeasible to find two distinct messages that hash to the same value. MD5 fails this requirement catastrophically; such collisions can be found in seconds on an ordinary home computer.

MD5 processes a variable-length message into a fixed-length output of 128 bits. The input message is broken up into chunks of 512-bit blocks (sixteen 32-bit words); 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 fewer than a multiple of 512. The remaining bits are filled up with 64 bits representing the length of the original message, modulo 2^{64}.

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. The processing of a message block consists of four similar stages, termed *rounds*; each round is composed of 16 similar operations based on a non-linear function F, modular addition, and left rotation. Figure 1 illustrates one operation within a round. There are four possible functions; a different one is used in each round:

F(B,C,D) = (B^C) || (!B ^ D)
G(B,C,D) = (B^D) || (C ^ !D)
H(B,C,D) = B ~ C ~ D
I(B,C,D) = C ~ (B || !D)

~ , ^ , || , ! denote the XOR, AND, OR and NOT operations respectively.

CODE:

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <math.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)
{
    /*Initialize Digest Array as A , B, C, D */
    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     b[64];
    }mm;
    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;
}

int main( int argc, char *argv[] )
{
    int j,k;
    const char *msg = "Hello World";
    printf("Input String to be Encrypted using MD5 : \n\t%s",msg);
    unsigned *d = md5(msg, strlen(msg));
    MD5union u;
    printf("\n\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");
    return 0;
}
```