JSS Mahavidyapeetha, Mysuru

JSS Academy of Technical Education

JSS Campus, Uttarahalli – Kengeri Main road, Bengaluru – 560060

Department of Computer Science and Engineering

CERTIFICATE

This is to certify that the project work entitled "**Assignment 2**" is a bona fide work carried out by **Mr. Prithviraj Patil (1JS19CS125)** in partial fulfillment of the requirements for the course Cryptography of 7th semester, Bachelor of engineering in Computer Science and Engineering of the Visvesvaraya Technological University, Belagavi, during the academic year 2022 – 2023. It is certified that all corrections and suggestions indicated for internal assessment have been incorporated in the report. The project report has been approved as it satisfies the academic requirements in respect of the project work prescribed for the said degree.

Mrs. Naidila Sadashiv B.E, MTech, Ph.D
Associate professor, Dept. of CSE,
JSSATE, Bengaluru

Dr. Naveen N.C. B.E, M.Tech ,Phd
Professor and Head, Dept. of CSE
JSSATE, Bengaluru

ACKNOWLEDGEMENT

I express my humble greetings to his holiness **Jagadguru Sri Shivarathri Deshikendra Mahaswamijigalavaru** who has showered their blessings on us for framing our career successfully.

The completion of any project involves the efforts of many people. I've been lucky to have received a lot of help and support from friends, family and all other quarters, during the making of this project. I take this opportunity to acknowledge all those, whose guidance and encouragement helped me emerge successful.

I am thankful to the resourceful guidance, timely assistance and graceful gesture of my guide **Mrs. Naidila Sadashiv,** Assistant professor, Department of Computer Science and engineering, who helped me in every aspect of my project work.

I am also indebted to **Dr. Naveen N.C.**, Professor and head of department of Computer Science and engineering for the facilities and support extended towards us.

I express our sincere thanks to our beloved principal, **Dr. Bhimasen Soragaon** for having supported us in our academic endeavors.

Last but not the least, I am pleased to express our heart full thanks to all the teaching and non-teaching staff of department of Computer and Science engineering and my friends who have rendered their help, motivation and support.

PRITHVIRAJ PATIL (1JS19CS125)

TABLE OF CONTENTS

RSA Algorithm

RSA algorithm is an 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 the Private key is kept private.

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 a 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 Implementation in C++.

Encrypting and decrypting small numeral values

```
#include <bits/stdc++.h>
using namespace std;
int gcd(int a, int h)
{
   int temp;
   while (1) {
      temp = a % h;
      if (temp == 0)
         return h;
      a = h;
      h = temp;
   }
}
int main()
{
   double p = 3;
```

Diffie-Hellman algorithm

The Diffie-Hellman algorithm is being used to establish a shared secret that can be used for secret communications while exchanging data over a public network using the elliptic curve to generate points and get the secret key using the parameters.

For the sake of simplicity and practical implementation of the algorithm, we will consider only 4 variables, one prime P and G (a primitive root of P) and two private values a and b.
P and G are both publicly available numbers. Users (say Alice and Bob) pick private values a and b and they generate a key and exchange it publicly. The opposite person receives the key and that generates a secret key, after which they have the same secret key to encrypt.

Code Implementation in C++.

```
#include <cmath>
#include <iostream>
using namespace std;

// Power function to return value of a ^ b mod P
long long int power(long long int a, long long int b,
                                    long long int P)
{
        if (b == 1)
                return a;

        else
                return (((long long int)pow(a, b)) % P);
}

// Driver program
int main()
{
        long long int P, G, x, a, y, b, ka, kb;

        // Both the persons will be agreed upon the
        // public keys G and P
        P = 23; // A prime number P is taken
        cout << "The value of P : " << P << endl;

        G = 9; // A primitive root for P, G is taken
        cout << "The value of G : " << G << endl;
```

Caesar Cipher Implementation

The Caesar Cipher technique is one of the earliest and simplest methods of encryption technique. It's simply a type of substitution cipher, i.e., each letter of a given text is replaced by a letter with a fixed number of positions down the alphabet. For example with a shift of 1, A would be replaced by B, B would become C, and so on. The method is apparently named after Julius Caesar, who apparently used it to communicate with his officials.

Thus to cipher a given text we need an integer value, known as a shift which indicates the number of positions each letter of the text has been moved down. The encryption can be represented using modular arithmetic by first transforming the letters into numbers, according to the scheme, A = 0, B = 1,…, Z = 25. Encryption of a letter by a shift n can be described mathematically as.

Code implementation in C++.

```
#include <iostream>

using namespace std;

string encrypt(string text, int s)

{

    string result = "";

    // traverse text

    for (int i = 0; i < text.length(); i++) {

        // apply transformation to each character

        // Encrypt Uppercase letters

        if (isupper(text[i]))

            result += char(int(text[i] + s - 65) % 26 + 65);

        // Encrypt Lowercase letters

        else

            result += char(int(text[i] + s - 97) % 26 + 97);

    }
```

Hill Cipher

Hill cipher is a polygraphic substitution cipher based on linear algebra.Each letter is represented by a number modulo 26. Often the simple scheme A = 0, B = 1, …, Z = 25 is used, but this is not an essential feature of the cipher. To encrypt a message, each block of n letters (considered as an n-component vector) is multiplied by an invertible n × n matrix, against modulus 26. To decrypt the message, each block is multiplied by the inverse of the matrix used for encryption.

The matrix used for encryption is the cipher key, and it should be chosen randomly from the set of invertible n × n matrices (modulo 26).

Code implementation in C++.

```
#include <iostream>
using namespace std;
// Following function generates the
//  key matrix for the key string
void getKeyMatrix(string key, int keyMatrix[][3])
{
   int k = 0;
   for (int i = 0; i < 3; i++)
   {
      for (int j = 0; j < 3; j++)
      {
         keyMatrix[i][j] = (key[k]) % 65;
         k++;
      }
   }
}
// Following function encrypts the message
void encrypt(int cipherMatrix[][1],
            int keyMatrix[][3],
            int messageVector[][1])
```

```
{
    int x, i, j;
    for (i = 0; i < 3; i++)
    {
        for (j = 0; j < 1; j++)
        {
            cipherMatrix[i][j] = 0;

             for (x = 0; x < 3; x++)
            {
                cipherMatrix[i][j] +=
                    keyMatrix[i][x] * messageVector[x][j];
            }
            cipherMatrix[i][j] = cipherMatrix[i][j] % 26;
        }
    }
}
// Function to implement Hill Cipher
void HillCipher(string message, string key)
{
    // Get key matrix from the key string
    int keyMatrix[3][3];
    getKeyMatrix(key, keyMatrix);
    int messageVector[3][1];
    // Generate vector for the message
    for (int i = 0; i < 3; i++)
        messageVector[i][0] = (message[i]) % 65;
    int cipherMatrix[3][1];
```

Data encryption standard

Data encryption standard (DES) has been found vulnerable to very powerful attacks and therefore, the popularity of DES has been found slightly on the decline. DES is a block cipher and encrypts data in blocks of size of **64 bits** each, which means 64 bits of plain text go as the input to DES, which produces 64 bits of ciphertext. The same algorithm and key are used for encryption and decryption, with minor differences.

Code Implementation in C++.

```
#include <bits/stdc++.h>

using namespace std;

string hex2bin(string s)

{

   // hexadecimal to binary conversion

   unordered_map<char, string> mp;

   mp['0'] = "0000";

   mp['1'] = "0001";

   mp['2'] = "0010";

   mp['3'] = "0011";

   mp['4'] = "0100";

   mp['5'] = "0101";

   mp['6'] = "0110";

   mp['7'] = "0111";

   mp['8'] = "1000";

   mp['9'] = "1001";

   mp['A'] = "1010";

   mp['B'] = "1011";

   mp['C'] = "1100";

   mp['D'] = "1101";

   mp['E'] = "1110";

   mp['F'] = "1111";

   string bin = "";
```

```
    for (int i = 0; i < s.size(); i++) {
        bin += mp[s[i]];
    }
    return bin;
}
string bin2hex(string s)
{
    // binary to hexadecimal conversion
    unordered_map<string, string> mp;
    mp["0000"] = "0";
    mp["0001"] = "1";
    mp["0010"] = "2";
    mp["0011"] = "3";
    mp["0100"] = "4";
    mp["0101"] = "5";
    mp["0110"] = "6";
    mp["0111"] = "7";
    mp["1000"] = "8";
    mp["1001"] = "9";
    mp["1010"] = "A";
    mp["1011"] = "B";
    mp["1100"] = "C";
    mp["1101"] = "D";
    mp["1110"] = "E";
    mp["1111"] = "F";
    string hex = "";
    for (int i = 0; i < s.length(); i += 4) {
        string ch = "";
        ch += s[i];
```

```
        ch += s[i + 1];
        ch += s[i + 2];
        ch += s[i + 3];
        hex += mp[ch];
    }
    return hex;
}

string permute(string k, int* arr, int n)
{
    string per = "";
    for (int i = 0; i < n; i++) {
        per += k[arr[i] - 1];
    }
    return per;
}
string shift_left(string k, int shifts)
{
    string s = "";
    for (int i = 0; i < shifts; i++) {
        for (int j = 1; j < 28; j++) {
            s += k[j];
        }
        s += k[0];
        k = s;
        s = "";
    }
    return k;
}
```

```
string xor_(string a, string b)
{
    string ans = "";
    for (int i = 0; i < a.size(); i++) {
        if (a[i] == b[i]) {
            ans += "0";
        }
        else {
            ans += "1";
        }
    }
    return ans;
}
string encrypt(string pt, vector<string> rkb,
               vector<string> rk)
{
    // Hexadecimal to binary
    pt = hex2bin(pt);


    // Initial Permutation Table
    int initial_perm[64]
        = { 58, 50, 42, 34, 26, 18, 10, 2,  60, 52, 44, 36, 28, 20, 12, 4,  62, 54, 46,
38, 30, 22, 14, 6,  64, 56, 48, 40, 32, 24, 16, 8,  57 49, 41, 33, 25, 17, 9,  1,  59,
51, 43, 35, 27, 19, 11, 3,  61, 53, 45, 37, 29, 21, 13, 5,  63, 55, 47, 39, 31, 23, 15,
7 };
    // Initial Permutation
    pt = permute(pt, initial_perm, 64);
    cout << "After initial permutation: " << bin2hex(pt)
         << endl;
// Splitting
```

```
    string left = pt.substr(0, 32);
    string right = pt.substr(32, 32);
    cout << "After splitting: L0=" << bin2hex(left)
         << " R0=" << bin2hex(right) << endl;
  // Expansion D-box Table
    int exp_d[48]
        = { 32, 1,  2,  3,  4,  5,  4,  5,  6,  7,  8,  9,
            8,  9,  10, 11, 12, 13, 12, 13, 14, 15, 16, 17,
            16, 17, 18, 19, 20, 21, 20, 21, 22, 23, 24, 25,
            24, 25, 26, 27, 28, 29, 28, 29, 30, 31, 32, 1 };
    // S-box Table
    int s[8][4][16] = {
        { 14, 4,  13, 1, 2,  15, 11, 8,  3,  10, 6,  12, 5, 9,  0,  7,  0, 15, 7,  4,  14, 2,  13,
1,  10, 6, 12, 11, 9,  5, 3,  8,  4,  1,  14, 8,  13, 6,  2, 11, 15, 12, 9, 7,  3,  10, 5,  0,
15, 12, 8,  2, 4,  9,  1,  7, 5,  11, 3,  14, 10, 0,  6,  13 },{ 15, 1,  8,  14, 6,  11, 3, 4,
9,  7,  2,  13, 12, 0,  5,  10, 3,  13, 4,  7, 15, 2,  8,  14, 12, 0, 1,  10, 6,  9,  11, 5,  0,
14, 7,  11, 10, 4,  13, 1,  5,  8,  12, 6,  9,  3, 2,  15, 13, 8,  10, 1,3,  15, 4,  2,  11, 6,
7, 12, 0,  5,  14, 9 }, { 10, 0,  9,  14, 6,  3,  15, 5,  1,  13, 12, 7,  11, 4,  2,  8,  13,
7,  0,  9,  3,  4,  6,  10, 2,  8,  5,  14, 12, 11, 15, 1,  13, 6,  4,  9,  8,  15, 3,  0,  11,
1,  2,  12, 5,  10, 14, 7,  1,  10, 13, 0,  6,  9,  8, 7,  4,  15, 14, 3,  11, 5,  2,  12 }, {
7,  13, 14, 3,  0,  6,  9,  10, 1,  2, 8,  5,  11, , 4,  15, 13, 8,  11, 5,  6,  15, 0, 3,  4,
7, 2,  12, 1,  10, 14, 9,  10, 6,  9,  0, 12, 11, 7, 13, 15, 1,  3,  14, 5,  2,  8,  4,  3, 15,
0,  6,  10, 1,  13, 8,  9,  4,  5,  11, 12, 7, 2,  14 },{ 2,  12, 4, 1,  7,  10, 11, 6, 8,  5,
3,  15, 13, 0,  14, 9, 14, 11, 2,  12, 4, 7,  13, 1,  5,  0, 15, 10, 3, 9,  8,  6,  4,  2, 1,
11, 10, 13, 7 , 15, 9, 12, 5,  6,  3,  0, 14, 11, 8,  12, 7, 1,  14, 2, 13, 6,  15, 0,  9,
10, 4,  5,  3 },{ 12, 1,  10, 15, 9,  2,  6,  8,  0,  13, 3, 4, 14, 7,  5,  11, 10, 15, 4,  2,
7,  12, 9,  5, 6, 1, 13, 14, 0,  11, 3,  8,  9,  14, 15, 5,  2, 8, 12, 3,  7,  0,  4,  10, 1,
13, 11, 6,  4,  3, 2, 12, 9,  5,  15, 10, 11, 14, 1,  7,  6,  0,  8, 13 },{ 4,  11, 2,  14,
15, 0,  8, 13, 3,  12, 9,  7,  5, 10, 6,  1,  13, 0,  11, 7, 4,  9,  1,  10, 14, 3, 5,  12, 2,
15, 8,  6,  1, 4,  11, 13, 12, 3,  7, 14, 10, 15, 6,  8,  0,  5, 9,  2,  6,  11, 13, 8, 1,  4,
10, 7,  9,  5,  0, 15, 14, 2,  3,  12 },{ 13, 2,  8, 4,  6,  15, 11, 1,  10, 9, 3, 14, 5,0,
12, 7, 1,  15, 13, 8,  10, 3,  7, 4, 12, 5, 6,  11, 0, 14, 9,  2,  7,  11, 4,  1, 9, 12, 14,
2,  0,  6, 10, 13, 15, 3,  5,  8,  2, 1, 14, 7, 4,  10, 8, 13, 15, 12, 9,  0,  3,  5, 6, 11 }
    };
  // Straight Permutation Table
```

```
int per[32]
    = { 16, 7, 20, 21, 29, 12, 28, 17, 1,  15, 23,
        26, 5, 18, 31, 10, 2,  8,  24, 14, 32, 27,
        3,  9, 19, 13, 30, 6,  22, 11, 4,  25 };
cout << endl;
for (int i = 0; i < 16; i++) {
    // Expansion D-box
    string right_expanded = permute(right, exp_d, 48);
    // XOR RoundKey[i] and right_expanded
    string x = xor_(rkb[i], right_expanded);
    // S-boxes
    string op = "";
    for (int i = 0; i < 8; i++) {
        int row = 2 * int(x[i * 6] - '0')
                + int(x[i * 6 + 5] - '0');
        int col = 8 * int(x[i * 6 + 1] - '0')
                + 4 * int(x[i * 6 + 2] - '0')
                + 2 * int(x[i * 6 + 3] - '0')
                + int(x[i * 6 + 4] - '0');
        int val = s[i][row][col];
        op += char(val / 8 + '0');
        val = val % 8;
        op += char(val / 4 + '0');
        val = val % 4;
        op += char(val / 2 + '0');
        val = val % 2;
        op += char(val + '0');
    }
    // Straight D-box
```

```
        op = permute(op, per, 32);
        // XOR left and op
        x = xor_(op, left);
        left = x;
        // Swapper
        if (i != 15) {
            swap(left, right);
        }
        cout << "Round " << i + 1 << " " << bin2hex(left)
             << " " << bin2hex(right) << " " << rk[i]
             << endl;
    }
    // Combination
    string combine = left + right;

    // Final Permutation Table
    int final_perm[64]
        = { 40, 8,  48, 16, 56, 24, 64, 32, 39, 7,  47,
            15, 55, 23, 63, 31, 38, 6,  46, 14, 54, 22,
            62, 30, 37, 5,  45, 13, 53, 21, 61, 29, 36,
            4,  44, 12, 52, 20, 60, 28, 35, 3,  43, 11,
            51, 19, 59, 27, 34, 2,  42, 10, 50, 18, 58,
            26, 33, 1,  41, 9,  49, 17, 57, 25 };
    // Final Permutation
    string cipher
        = bin2hex(permute(combine, final_perm, 64));
    return cipher;
}
// Driver code
```

```
int main()
{
    // pt is plain text
    string pt, key;
    /*cout<<"Enter plain text(in hexadecimal): ";
    cin>>pt;
    cout<<"Enter key(in hexadecimal): ";
    cin>>key;*/
    pt = "123456ABCD132536";
    key = "AABB09182736CCDD";
    // Key Generation
    // Hex to binary
    key = hex2bin(key);
    // Parity bit drop table
    int keyp[56]
        = { 57, 49, 41, 33, 25, 17, 9,  1,  58, 50, 42, 34, 26, 18, 10, 2,  59, 51, 43, 35,
27, 19, 11, 3, 60, 52, 44, 36, 63, 55, 47, 39, 31, 23, 15, 7, 62, 54, 46, 38, 30, 22,
14, 6,  61, 53, 45, 37, 29, 21, 13, 5,  28, 20, 12, 4 };
    // getting 56 bit key from 64 bit using the parity bits
    key = permute(key, keyp, 56); // key without parity
    // Number of bit shifts
    int shift_table[16] = { 1, 1, 2, 2, 2, 2, 2, 2,
                            1, 2, 2, 2, 2, 2, 2, 1 };
    // Key- Compression Table
    int key_comp[48] = { 14, 17, 11, 24, 1,  5,  3,  28, 15, 6,  21, 10, 23, 19, 12, 4,
26, 8,  16, 7,  27, 20, 13, 2, 41, 52, 31, 37, 47, 55, 30, 40, 51, 45, 33, 48, 44, 49,
39, 56, 34, 53, 46, 42, 50, 36, 29, 32 };
    // Splitting
    string left = key.substr(0, 28);
    string right = key.substr(28, 28);
```

```
    vector<string> rkb; // rkb for RoundKeys in binary
    vector<string> rk; // rk for RoundKeys in hexadecimal
    for (int i = 0; i < 16; i++) {
        // Shifting
        left = shift_left(left, shift_table[i]);
        right = shift_left(right, shift_table[i]);
        // Combining
        string combine = left + right;
        // Key Compression
        string RoundKey = permute(combine, key_comp, 48);
        rkb.push_back(RoundKey);
        rk.push_back(bin2hex(RoundKey));
    }
    cout << "\nEncryption:\n\n";
    string cipher = encrypt(pt, rkb, rk);
    cout << "\nCipher Text: " << cipher << endl;
    cout << "\nDecryption\n\n";
    reverse(rkb.begin(), rkb.end());
    reverse(rk.begin(), rk.end());
    string text = encrypt(cipher, rkb, rk);
    cout << "\nPlain Text: " << text << endl;
}
```