# SIDDAGANGA INSTITUTE OF TECHNOLOGY, TUMKUR-572103

## DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

### CRYPTOGRAPHY AND NETWORK SECURITY LAB (7RCSL01)

| Student Name: Pragati Shankar | USN: 1SI19CS090 | Batch No: B2 | Date: 15-11-2022 |
|---|---|---|---|

**Evaluation:**

| Write Up (10 marks) | Clarity in concepts (10 marks) | Implementation and execution of the algorithms (10 marks) | Viva (05 marks) | Total (35 marks) |
|---|---|---|---|---|
| | | | | |

| Sl.No | Name of the Faculty In-Charge | Signature |
|---|---|---|
| 1. | H K Vedamurthy | |
| 2. | Guruaj S P | |

**Question No: 3**

Write a program to perform the following using Hill cipher:

(i) Encrypt a message M with a given key matrix of size 2X2 and 3X3

(i) Decrypt the cipher text obtained in (i) by computing inverse of the respective key matrix

**Hill Cipher:**

This **encryption** algorithm takes m successive plaintext letters and substitutes for them m ciphertext letters. The substitution is determined by m linear equations in which each character is assigned a numerical value (a = 0, b = 1, , z = 25) . For m = 3, the system can be described as

$$c_1 = (k_{11}p_1 + k_{12}p_2 + k_{13}p_3) \bmod 26$$

$$c_2 = (k_{21}p_1 + k_{22}p_2 + k_{23}p_3) \bmod 26$$

$$c_3 = (k_{31}p_1 + k_{32}p_2 + k_{33}p_3) \bmod 26$$

**C = E(K, P) = PK mod 26** where C and P are row vectors of length 3 representing the plaintext and ciphertext, and K is a 3 X 3 matrix representing the encryption key. Operations are performed mod 26.

**Decryption** requires using the inverse of the matrix K.

$$\mathbf{P = D(K, C) = CK^{-1} \bmod 26 = PKK^{-1} = P}$$

For the 2X2 matrix determinant is $k_{11}k_{22} - k_{12}k_{21}$. For a 3X3 matrix, the value of the determinant is $k_{11}k_{22}k_{33} + k_{21}k_{32}k_{13} + k_{31}k_{12}k_{23} - k_{31}k_{22}k_{13} - k_{21}k_{12}k_{33} - k_{11}k_{32}k_{23}$

If a square matrix A has a nonzero determinant, then the inverse of the matrix is computed as $[A^{-1}]_{ij} = (\det A)^{-1}(-1)^{i+j}(D_{ji})$ , where $(D_{ji})$ is the sub determinant formed by deleting the 'j'th row and the' i'th column of A, det(A) is the determinant of A, and $(\det A)^{-1}$ is the multiplicative inverse of (det A) mod 26.

**CODE:**

```
#include<bits/stdc++.h>
using namespace std ;

int key[3][3];

int mod26(int x)
{
	return x >= 0 ? (x%26) : 26-(abs(x)%26) ;
}
int findDet(int m[3][3],int n)
{
	int det;
	if(n == 2)
	{
		det = m[0][0] * m[1][1] - m[0][1]*m[1][0] ;
	}
	else if (n == 3)
	{
		det = m[0][0]*(m[1][1]*m[2][2] - m[1][2]*m[2][1]) - m[0][1]*(m[1][0]*m[2][2] - m[2][0]*m[1][2]) +
m[0][2]*(m[1][0]*m[2][1] - m[1][1]*m[2][0]);
	}
	else det = 0 ;
	return mod26(det);
}

int findDetInverse(int R , int D = 26)
{
	int i = 0 ;
	int p[100] = {0,1};
	int q[100] = {0} ;
	while(R!=0)
	{
		q[i] = D/R ;
		int oldD = D ;
		D=R;
		R = oldD%R ;
		if(i>1)
		{
			p[i] = mod26(p[i-2] - p[i-1]*q[i-2]) ;
		}
		i++;
	}
	if (i == 1) return 1;
	else return p[i] = mod26(p[i-2] - p[i-1]*q[i-2]) ;
}

void multiplyMatrices(int a[1000][3] , int a_rows , int a_cols , int b[1000][3] , int b_rows , int b_cols, int
res[1000][3])
{
```

```
	for(int i=0 ; i < a_rows ; i++)
	{
		for(int j=0 ; j < b_cols ; j++)
		{
			for(int k=0 ; k < b_rows ; k++)
			{
				res[i][j] += a[i][k]*b[k][j];
			}
			res[i][j] = mod26(res[i][j]);
		}
	}
}

void findInverse(int m[3][3] , int n , int m_inverse[3][3] )
{
	int adj[3][3] = {0};
	int det = findDet(m , n);
	int detInverse = findDetInverse(det);
	if(n==2)
	{
		adj[0][0] = m[1][1];
		adj[1][1] = m[0][0];
		adj[0][1] = -m[0][1];
		adj[1][0] = -m[1][0];
	}
	else if(n==3)
	{
		int temp[5][5] = {0} ;
		for(int i=0; i<5; i++)
		{
			for(int j=0; j<5; j++)
			{
				temp[i][j] = m[i%3][j%3] ;
			}
		}
		for(int i=1; i<=3 ; i++)
		{
			for(int j=1; j<=3 ; j++)
			{
				adj[j-1][i-1] = temp[i][j]*temp[i+1][j+1] - temp[i][j+1]*temp[i+1][j];
			}
		}
	}
	for(int i=0; i<n ; i++){
		for(int j=0; j<n ; j++)
		{
			m_inverse[i][j] = mod26(adj[i][j] * detInverse) ;
		}
	}
}
```

```
string encrypt(string pt, int n)
{
	int P[1000][3] = {0} ;
	int C[1000][3] = {0} ;
	int ptIter = 0 ;
	while(pt.length()%n != 0)
	{
		pt += "x" ;
	}
	int row = (pt.length())/n;
	for(int i=0; i<row ; i++)
	{
		for(int j=0; j<n; j++)
		{
			P[i][j] = pt[ptIter++]-'a' ;
		}
	}
	multiplyMatrices(P, row , n , key , n , n , C) ;
	string ct = "" ;
	for(int i=0 ; i<row ; i++)
	{
		for(int j=0 ; j<n ;j++)
		{
			ct += (C[i][j] + 'a');
		}
	}
	return ct ;
}

string decrypt(string ct, int n)
{
	int P[1000][3] = {0} ;
	int C[1000][3] = {0} ;
	int ctIter = 0 ;
	int row = ct.length()/n;
	for(int i=0; i<row ;i++){
		for(int j=0; j<n; j++)
		{
			C[i][j] = ct[ctIter++]-'a' ;
		}
	}
	int k_inverse[3][3] = {0};
	findInverse(key,n,k_inverse);
	multiplyMatrices(C,row,n,k_inverse,n,n,P) ;
	string pt = "" ;
	for(int i = 0 ; i<row ; i++)
	{
		for(int j=0 ; j<n ; j++)
		{
			pt += (P[i][j] + 'a');
		}
```

```
        }
        return pt ;
}


int main(void)
{
        string pt ;
        int n ;
        cout << "Enter the text to be encrypted :";
        cin >> pt;
        cout << "Enter order of key matrix : ";
        cin >> n ;
        cout<<"Enter key matrix: " <<endl;
        for(int i=0; i<n; i++)
        {
                for(int j=0; j<n; j++)
                {
                        cin >> key[i][j];
                }
        }
        cout << "\nOriginal text : " << pt << endl;
        string ct = encrypt(pt, n) ;
        cout << "Encrypted text : " << ct << endl;
        string dt = decrypt(ct, n);
        cout << "Decrypted text : " << dt << endl;
}
```

**Output Screenshot:**