MC 208: LINEAR ANALYSIS

PROJECT REPORT

TOPIC: "APPLICATION OF LINEAR ALGEBRA IN HILL CIPHER ENCRYPTION DECRYPTION"

DELHI TECHNOLOGICAL UNIVERSITY

(DEPARTMENT OF APPLIED MATHEMATICS)

SUBMITTED TO:

Ms Kanchan Jangra

Department of Applied Mathematics

SUBMITTED BY:

Shresth Jain 2K19 / MC / 121

Siddharth Bihani 2K19 / MC / 125

ACKNOWLEDGMENT

We would like to express our sincere gratitude to the Delhi Technological University, Delhi for their valuable support and for providing us with a platform to showcase our research and analysis skills in making this project on **'APPLICATION OF LINEAR ALGEBRA IN HILL CIPHER ENCRYPTION DECRYPTION'.**

We would also like to express our deepest appreciation to all those who provided us with the possibility to complete this project review report. Also, we would like to give special gratitude to Ms Kanchan Jangra, who gave us this opportunity to work on this project and gave us all the support and guidance which made us complete the project review duly.

The success and outcome of this progress report required a lot of guidance and assistance from many people and we are extremely privileged to have got this all along with the completion of it. All that we have done is only because of such supervision and assistance and we will not forget to thank them.

Contents

1 OBJECTIVE

In this project, we aim to understand the concept behind the Hill Cipher Encryption and Decryption technique, one of the famous cryptography technique used with the help of our knowledge of Linear Algebra

We start off this project, with understanding what cryptography is .We also apply our knowledge of Linear Algebra to encrypt and decrypt messages using the Hill Cipher method. We will also be writing a code in C++ to do the same. Now as, all techniques come with their own advantages and disadvantages, hence we discuss the advantages and disadvantages of the Hill Cipher method. In the end, we discuss some of the techniques which can be used to improve upon the already existing method.

2 INTRODUCTION

Cryptography is the practice and study of encoding and decoding secret messages, to establish secure communication and preventing third parties from reading the private messages. Codes are called cyphers in the cryptography language, uncoded messages are called plaintext, and coded message is called ciphertext.

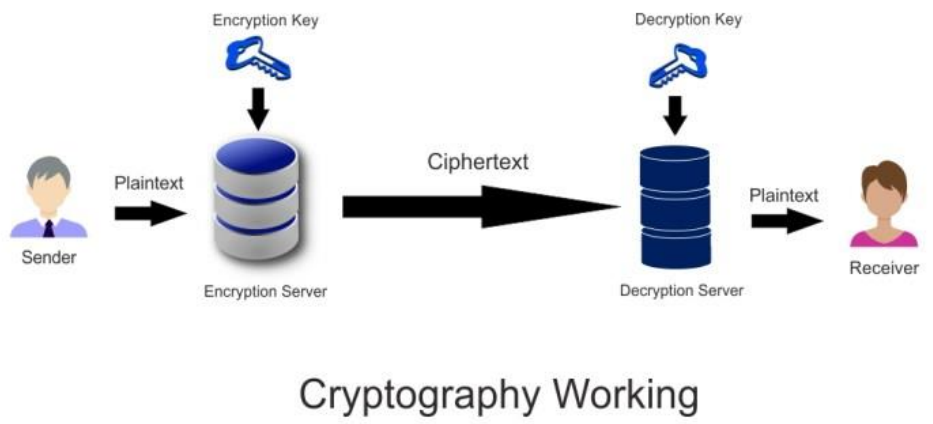


Image 1 – Image depicting the process of cryptography

It has been thousands of years since cryptography, is being practised to protect secret messages. Until recently, it had been the storey of what is known as classic cryptography, or encryption methods that rely on pen and paper or perhaps basic mechanical aids. The invention of complex mechanical and electromechanical machines, such as the Enigma rotor machine, provided a more advanced and efficient encryption method at the beginning of the 20th century and the subsequent advent of electronics and computing allowed for complex systems which were mostly unfit for pen and paper.

Cryptography has long been a significant topic in the computing world. It was originally developed to ensure the protection of passwords, but cryptography has become increasingly relevant as a result of the Internet’s flow of sensitive information such as credit card numbers and other sensitive data, which is relatively simple to track by unintended third parties.

The aim of encoding a message is to render it useless to anyone but the party who has the “key” to decode it.

3 HILL CIPHER

In Classical Cryptography, The Hill Cipher is a polygraphic substitution cypher based on Linear Algebra principles. Lester S. Hill invented it in 1929, and it was the first polygraphic cypher that could work on more than three symbols at the same time. Modulo arithmetic, matrix multiplication, adjoint of matrix, inverse of matrix, and determinants are all used in the Hill Cipher. It is a more mathematical cypher than other cyphers since it employs both of these principles. Since Hill Cipher is a block cypher, it can potentially operate on blocks of any size. Hill Ciphers encrypt and decrypt messages using principles from modular and linear algebra.

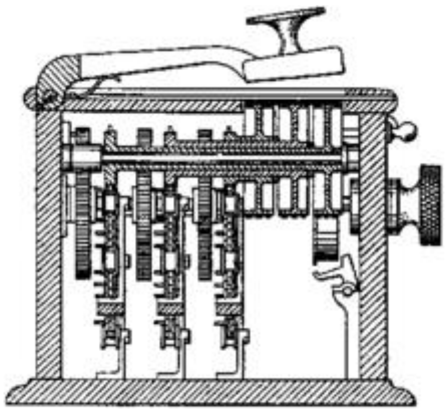


Image 2 – The Hill’s Cipher Machine

Each letter of the alphabet is assigned a numerical value, such as A-0, B-1, E-4, T-19, Z-25, and so on, and modular algebra is used to hold calculations within the letter's range of values. The message is then converted into number matrices, which are then multiplied by a key matrix (the key matrix must be invertible since its inverse is used in the decryption process).

The encoded message is multiplied with the modular of the inverse of the key matrix for decryption, and the modular operation returns the decrypted message.

4 ENCRYPTION USING HILL CIPHER

First we convert all the letters to numbers as:

A	B	C	D	E	F	G	H	I	J	K	L	M
0	1	2	3	4	5	6	7	8	9	10	11	12

N	O	P	Q	R	S	T	U	V	W	X	Y	Z
13	14	15	16	17	18	19	20	21	22	23	24	25

Image 3 – Table showing values assigned to the alphabets used in encrypting and decrypting a message using hill cipher

The Hill cipher relies on the following operation to encrypt data:

$$\mathbf{E(K, P) = (P*K) \bmod 26} \qquad \mathbf{-(1)}$$

Where K is our Key Matrix and P is our message to be encoded in Vector form.

The above method is explained in more detail using the example below.

Suppose our key matrix, K = $\begin{bmatrix} 3 & 3 \\ 2 & 5 \end{bmatrix}$, and we want to encrypt the word "HILL" using it. So, we will be converting the plaintext word, "HILL" into a matrix. Hence the matrix, P = $\begin{bmatrix} 7 & 8 \\ 11 & 11 \end{bmatrix}$, formed using image 3. Now using formula (1), we will form the matrix of the decrypted message and finally using image 3, we will convert the matrix of decrypted message into the final ciphertext.

Hence, $\begin{bmatrix} 7 & 8 \\ 11 & 11 \end{bmatrix} * \begin{bmatrix} 3 & 3 \\ 2 & 5 \end{bmatrix} mod\ 26 = \begin{bmatrix} 37 & 61 \\ 55 & 88 \end{bmatrix} mod\ 26 = \begin{bmatrix} 11 & 9 \\ 3 & 10 \end{bmatrix}$. Hence, using image 3, we find the final cipher text to be "LJDK".

Hence, using key matrix, K = $\begin{bmatrix} 3 & 3 \\ 2 & 5 \end{bmatrix}$, to encrypt the word **"HILL",** we get the final encrypted text to be **"LJDK".**

5 DECRYPTION USING HILL CIPHER

The Hill cipher's decryption is based on the following steps:

$$\mathbf{D(K, C) = (C * K^{-1}) \bmod 26} \qquad \text{-(2)}$$

Where K is our key matrix and C is the ciphertext in vector form. Matrix multiplying the inverse of the key matrix with the matrix of ciphertext produces the decrypted plaintext.

The above method is explained in more detail using the example below.

Let us continue the previous example itself and try to find out the decrypted text, using the key matrix and the matrix of the final cipher text.

Hence, the matrix of the ciphertext, C = $\begin{bmatrix} 11 & 9 \\ 3 & 10 \end{bmatrix}$. First we will find the inverse of the key matrix, using the formula K^{-1} = adj(K)/|K|, we have also applied the Extended Euclidean Algorithm while calculating the inverse,

$$K^{-1} \% \ 26 = \begin{bmatrix} 3 & 3 \\ 2 & 5 \end{bmatrix}^{-1} \% \ 26 = \underbrace{((3x5) - (3x2))^{1} \times \begin{bmatrix} 5 & -3 \\ -2 & 3 \end{bmatrix}}_{\text{solved by Extended Euclidean Algorithm}} \% \ 26 = 3 \begin{bmatrix} 5 & 23 \\ 24 & 3 \end{bmatrix} = \begin{bmatrix} 15 & 69 \\ 72 & 9 \end{bmatrix} \% \ 26 = \begin{bmatrix} 15 & 17 \\ 20 & 9 \end{bmatrix}$$

Now, we will multiply, the matrix C with the inverse of the key matrix and use image 3 to convert the resultant matrix into the original decrypted text.

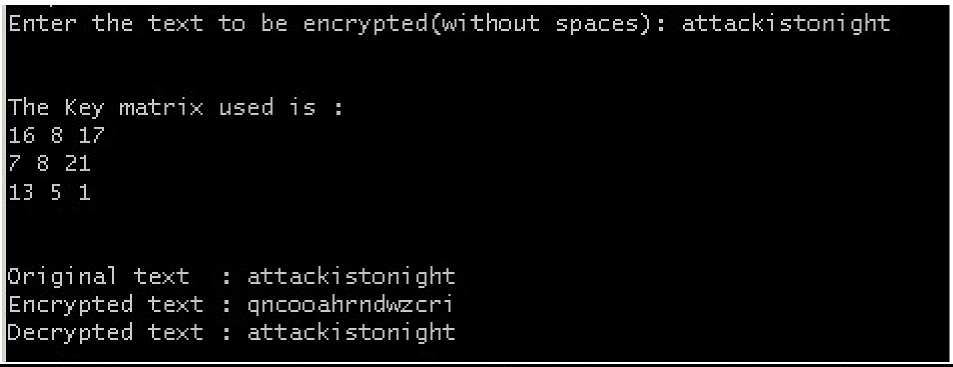
Hence, $\begin{bmatrix} 11 & 9 \\ 3 & 10 \end{bmatrix} * \begin{bmatrix} 15 & 17 \\ 20 & 9 \end{bmatrix} mod\ 26 = \begin{bmatrix} 345 & 268 \\ 245 & 141 \end{bmatrix} mod\ 26 = \begin{bmatrix} 7 & 8 \\ 11 & 11 \end{bmatrix}$.Hence, using image 3, we find the decrypted text to be “HILL”.

Hence, using key matrix, K = $\begin{bmatrix} 3 & 3 \\ 2 & 5 \end{bmatrix}$, to decrypt the word **“LJDK”,** we get the final decrypted text to be **“HILL”.**

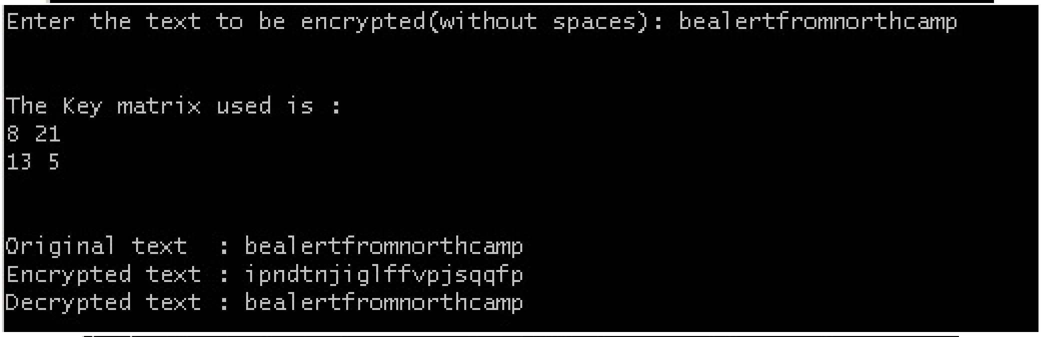
6 C++ IMPLEMENTATION OF HILL CIPHER

For code please refer to Appendix A.

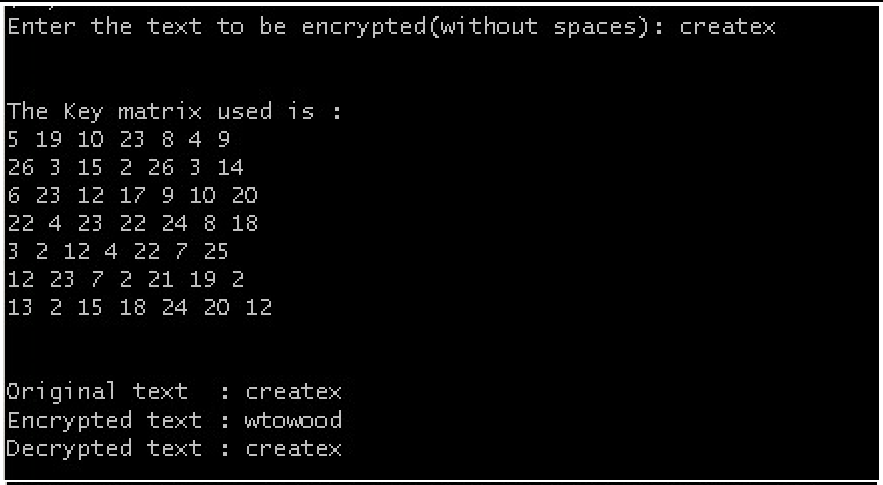
6.1 OUTPUT



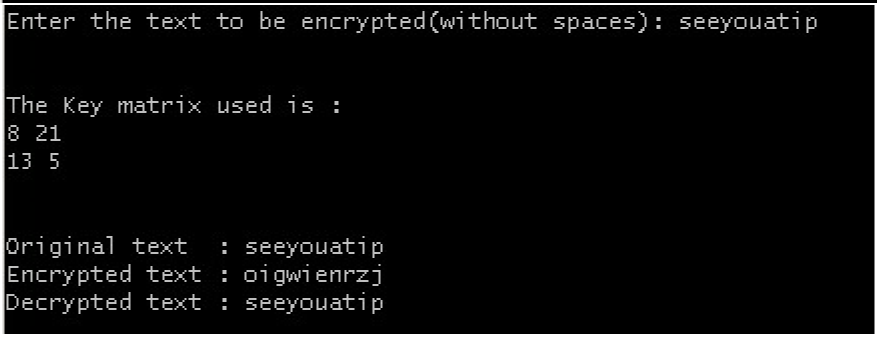
```
Enter the text to be encrypted(without spaces): attackistonight


The Key matrix used is :
16 8 17
7 8 21
13 5 1


Original text  : attackistonight
Encrypted text : qncooahrndwzcri
Decrypted text : attackistonight
```



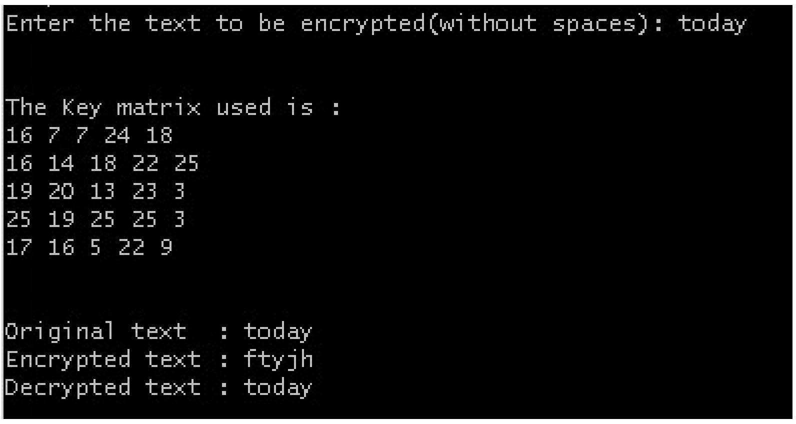
```
Enter the text to be encrypted(without spaces): bealertfromnorthcamp


The Key matrix used is :
8 21
13 5


Original text  : bealertfromnorthcamp
Encrypted text : ipndtnjiglffvpjsqqfp
Decrypted text : bealertfromnorthcamp
```



```
Enter the text to be encrypted(without spaces): createx


The Key matrix used is :
5 19 10 23 8 4 9
26 3 15 2 26 3 14
6 23 12 17 9 10 20
22 4 23 22 24 8 18
3 2 12 4 22 7 25
12 23 7 2 21 19 2
13 2 15 18 24 20 12


Original text  : createx
Encrypted text : wtowood
Decrypted text : createx
```



```
Enter the text to be encrypted(without spaces): seeyouatip


The Key matrix used is :
8 21
13 5


Original text  : seeyouatip
Encrypted text : oigwienrzj
Decrypted text : seeyouatip
```



```
Enter the text to be encrypted(without spaces): today


The Key matrix used is :
16 7 7 24 18
16 14 18 22 25
19 20 13 23 3
25 19 25 25 3
17 16 5 22 9


Original text  : today
Encrypted text : ftyjh
Decrypted text : today
```

7 ADVANTAGES AND DISADVANANTAGES OF HILL CIPHER

7.1 ADVANTAGES

- It does not resemble any pattern between message and encryption.
- It hides all the repetition and occurrences of any letter in the message.
- Encryption and Decryption is way too hard without the knowing the key matrix.
- There are many possibilities of probable key matrix and thus it ensures strong security.
- Matrix multiplication provides diffusion in encryption.

7.2 DISADVANTAGES

- Encryption and decryption process is complicated.
- It is vulnerable to a known-plaintext attack. Anyone who intercepts plain texts can develop a linear system which can usually be solved.
- Completely linear operations increase the chances of data interception.
- The message needs to be broken down to vectors of size n.
- Sending and receiving large messages is fairly complex process.
- With the advancement of computers, it is vulnerable to computer hacking. A computer can test out thousands of keys in a matter of seconds

8 SUGGESESTED IMPROVEMENTS IN HILL CIPHER

- Stronger cyphers can be made by combining symmetric-key cyphers. Simple transformations that are easy to test but fragile on their own can be used to create robust product cyphers.
- Based on Eigenvalues HCM-EE, we may suggest a modification to the Hill cypher. The HCM-EE produces a complex encryption key matrix using exponentiation and Eigenvalues, but it takes a long time.
- We may propose a modified Hill cypher that encrypts all plaintext blocks using a one-time-one key matrix. Per plaintext block is encrypted using its own key in this algorithm. The current key is multiplied by a hidden Initial Vector to generate this unique key (IV). Since the multiplying is done row by row, the algorithm is called Hill Multiplying Rows by Initial Vector (Hill MRIV). This algorithm has been shown to produce higher encryption efficiency.
- We can operate on an affine transformation-based symmetric cryptosystem. It starts with one random number and then generates more random numbers in a chain using HMAC (Hash Message Authentication Code).
- We can use the Hill cypher to build a safe cryptosystem for non-invertible matrices. Converting each plaintext character into two cypher text characters solves the noninvertible key matrix problem. In order to decode a message, two cypher text characters must be converted into one plaintext character. Even though this algorithm solves the non-invertible key problem, it takes a long time to decrypt because the decryption method requires the computation of an inverse key matrix. It will undoubtedly slow down the decryption process, particularly when a high-dimensional key matrix is involved.

9 CONCLUSION

Although the Hill Cypher was a breakthrough in encryption when it was invented in 1929, it does not provide the level of protection necessary to transmit something remotely private or confidential in today's world. It's vulnerable to a variety of attacks, and once the cypher is broken, messages can be intercepted and forwarded without either party knowing. The use of RSA encryption, which is extremely secure and uses extremely large prime numbers as keys, is an effective solution to this issue. These keys are normally 2048 bits long nowadays, which corresponds to a 617-digit decimal number. RSA is based on the assumption that there is no proven algorithm for factoring such large numbers efficiently. While the Hill Cypher is no longer used for encryption due to its weakness, it served as a foundation for a number of others.

10 REFERENCES

- https://en.wikipedia.org/wiki/Hill_cipher
- https://www.educative.io/edpresso/what-is-the-hill-cipher
- https://www.jigsawacademy.com/blogs/cyber-security/hill-cipher/
- https://www.cs.uri.edu/cryptography/classicalhill.htm

11 APPENDIX A

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

vector<vector<int>> key ; // Global variable for key matrix

/****************************************************/
/*Functions used for matrix operations*/
/****************************************************/

int mod26(int x)
{
        return x >= 0 ? (x%26) : 26-(abs(x)%26) ;
}

/*To calculate the cofactor of a nxn matrix
used in calculation of determinant of a matrix*/
vector<vector<int>> getCofactor(vector<vector<int>> mat, int p, int q, int n)
{
    int i = 0, j = 0;
        vector<vector<int>> temp((mat.size()-1),vector<int>(mat.size()-1));
    // Looping for each element of the matrix
    for (int row = 0; row < n; row++){
        for (int col = 0; col < n; col++){
            //  Copying into temporary matrix only those
            //  element which are not in given row and
            //  column
            if (row != p && col != q){
                temp[i][j++] = mat[row][col];
                if (j == n - 1){
                    j = 0;
                    i++;
                }
            }
        }
    }
        return temp;
}

/*To calculate the determinant of a nxn matrix */
int determinantOfMatrix(vector<vector<int>> mat, int n)
{
        int D = 0; // Initialize result
        // Base case : if matrix contains single element
        if (n == 1) return mat[0][0];
        vector<vector<int>> Temp((mat.size()-1),vector<int>(mat.size()-1));
        int sign = 1; // To store sign multiplier
        // Iterate for each element of first row
```

```
            for (int f = 0; f < n; f++){
                        // Getting Cofactor of mat[0][f]
        Temp=getCofactor(mat, 0, f, n);
                        D += sign * mat[0][f]* determinantOfMatrix(Temp, n - 1);
                        // terms are to be added with alternate sign
                        sign = -sign;
            }
            return mod26(D);
}

/*To Generate the key matrix of order n using the rand() function*/
void generatekey(int n)
{
            int det;
            do{
                        key.clear();
        for(int i=0; i<n; i++) {
            vector<int> v1;
            for(int j=0; j<n; j++) v1.push_back(rand()%26+1);   //from 1-26
            key.push_back(v1);
        }
        det=determinantOfMatrix(key,n);
        /*
        The keymatrix must satisfy the following conditions
        1) Matrix should not be singular
        2) Determinant must not have any common factors with modular base ( here =26)
        */
            }while(det==0 || det%2==0 || det%13==0 );
}

/*A function to optimize the determinant of the inverse of the
            key matrix according to the requirement */
int findDetInverse(int R , int D = 26) // R is the remainder or determinant
{
   int i = 0 ;
   int p[100] = {0,1};
   int q[100] = {0} ; // quotient
   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]) ;
}
/*To multiply two matrices a and b of the order a_rows x a_cols
 and b_rows x b_cols respectively*/
```

```
vector<vector<int>> multiplyMatrices(vector<vector<int>> a , int a_rows , int a_cols ,
vector<vector<int>> b, int b_rows , int b_cols)
{
        vector<vector<int>> res(a_rows,vector<int>(b_cols));
        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]) ;
    }
   }
  return res;
}
/*To calculate the adjoint of nxn square matrix */
vector<vector<int>> adjoint(vector<vector<int>> A,int n)
{
        vector<vector<int>> adj(n,vector<int>(n));
        if (n == 1){
     adj[0][0] = 1;
                return adj;
        }
        // temp is used to store cofactors of A[][]
        int sign = 1;
        vector<vector<int>> temp(n,vector<int>(n));
        for (int i=0; i<n; i++){
                for (int j=0; j<n; j++){
                        // Get cofactor of A[i][j]
                        temp=getCofactor(A, i, j, n);
                        // sign of adj[j][i] positive if sum of row
                        // and column indexes is even.
                        sign = ((i+j)%2==0)? 1: -1;
                        // Interchanging rows and columns to get the
                        // transpose of the cofactor matrix
                        adj[j][i] = (sign)*(determinantOfMatrix(temp, n-1));
                }
        }
        return adj;
}
/* To calulate the inverse of a nxn square invertible matrix*/
vector<vector<int>> inverse(vector<vector<int>> A,int n)
{
        // Find determinant of A[][]
        vector<vector<int>> inv(n,vector<int>(n));
        int det=determinantOfMatrix(A,n);
        int detInverse=findDetInverse(det);
        vector<vector<int>> adj(n,vector<int>(n));
        adj=adjoint(A, n);
        // Find Inverse using formula "inverse(A) = adj(A)/det(A)"
        for(int i=0; i<n ; i++)
    for(int j=0; j<n ; j++) inv[i][j] = mod26(adj[i][j] * detInverse) ;
        return inv;
}
/****************************************************/

/****************************************************/
/* To encrypt a string pt applying hill-cipher
  encryption technique*/
/****************************************************/
string encrypt(string pt, int n)
{
        int ptIter = 0  ;
        int row = (pt.length())/n; // number of rows in P
        vector<vector<int>> P(row,vector<int>(n)); // Plain Text
        /*Convert plain text to respective integers matrix*/
        for(int i=0; i<row ; i++)
                for(int j=0; j<n; j++) P[i][j] = pt[ptIter++]-'a' ;
        vector<vector<int>> C;          //Cipher Text
        // multiplyMatrices(mat_a , row_a , col_a ,mat_b,  row_b, col_b)
        C=multiplyMatrices(P, row , n, key,n , n);
        string ct = "" ;
        for(int i=0 ; i<row ; i++)
                for(int j=0 ; j<n ;j++) ct += (C[i][j] + 'a');
        return ct ;
}
/****************************************************/

/****************************************************/
/* To decrypt a string ct using hill-cipher
  decryption algorithms*/
/****************************************************/
string decrypt(string ct, int n)
{
        int ctIter = 0 ;
        int row = ct.length()/n; // number of rows in C
        vector<vector<int>> C (row,vector<int>(n)); // Cipher Text
        for(int i=0; i<row ; i++)
      for(int j=0; j<n; j++) C[i][j] = ct[ctIter++]-'a' ;
        vector<vector<int>> P ;         //Plane Text
        vector<vector<int>> inv (n,vector<int>(n)); // Cipher Text
        inv=inverse(key, n);
  /* multiplyMatrices(mat_a , row_a , col_a , mat_b, row_b, col_b) */
        P=multiplyMatrices(C, row , n, inv,n , n) ;
        string pt = "" ;
        for(int i = 0 ; i<row ; i++)
                for(int j=0 ; j<n ; j++) pt += (P[i][j] + 'a');
        return pt ;
}
/****************************************************/

/****************************************************/
/*Main Function */
```

```
/***************************************************/

int main(void)
{
	string pt ;
	cout << "Enter the text to be encrypted(without spaces): " ;
	getline(cin,pt);
	/*Calculate the order of the key matrix such that
	no extra element is left after making groups*/
	int n=2;
	for (int i = 2; i <= pt.length(); ++i){
		if(pt.length()%i==0){
			n=i;
			break;
		}
	}
	/*Generates a random Key of order n to be used for encryption*/
	generatekey(n);
	cout<<"\n\nThe Key matrix used is :\n";
	for(int i=0; i<n; i++) {
		for(int j=0; j<n; j++) cout<<key[i][j]<<' ';
		cout<<'\n';
	}
	cout<<'\n';
	string ct = encrypt(pt, n);
	cout << "\nOriginal text  : " << pt << endl;
	cout << "Encrypted text : " << ct << endl;
	cout << "Decrypted text : " << decrypt(ct, n) << endl;
	return 0;
}
/***************************************************/
```