# EXPERIMENT-5

**AIM:** To implement Hill-cipher encryption-decryption.

## THEORY:

The Hill Cipher is a polygraphic substitution cipher based on linear algebra that encrypts a block of plaintext letters using matrix multiplication modulo 26. Each letter is represented numerically (A=0, B=1, …, Z=25), and a key matrix is applied to transform plaintext vectors into ciphertext vectors. Decryption involves multiplying the ciphertext by the modular inverse of the key matrix. For the key to be valid, its determinant must have a modular inverse under mod 26. This cipher ensures diffusion and strengthens security compared to monoalphabetic ciphers by encrypting multiple letters simultaneously.

## CODE:

```
// To implement Hill- cipher encryption decryption By Aaditya Bhatia
23/CS/004

#include <iostream>

#include <vector>

#include <string>

using namespace std;


int modInverse(int a, int m) {

   a = a % m;

   for (int x = 1; x < m; x++)

       if ((a * x) % m == 1)

           return x;

   return -1;
```

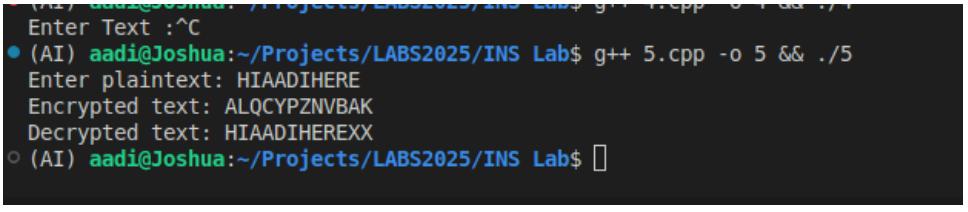
```
}


vector<int> multiplyMatrix(vector<vector<int>> key, vector<int> text) {
   vector<int> result(3);
   for (int i = 0; i < 3; i++)
   {
       result[i] = 0;
       for (int j = 0; j < 3; j++)
           result[i] += key[i][j] * text[j];
       result[i] = result[i] % 26;
   }
   return result;
}


int determinant(vector<vector<int>> key) {
   int det = key[0][0] * (key[1][1] * key[2][2] - key[1][2] * key[2][1])
- key[0][1] * (key[1][0] * key[2][2] - key[1][2] * key[2][0]) +
key[0][2] * (key[1][0] * key[2][1] - key[1][1] * key[2][0]);
   det = (det % 26 + 26) % 26;
   return det;
}
vector<vector<int>> inverseMatrix(vector<vector<int>> key) {
   int det = determinant(key);
   int detInv = modInverse(det, 26);
   if (detInv == -1)
   {
```

```
        cout << "Key matrix not invertible under mod 26!" << endl;
        exit(0);
    }
    vector<vector<int>> adj(3, vector<int>(3));
    adj[0][0] = (key[1][1] * key[2][2] - key[1][2] * key[2][1]);
    adj[0][1] = -(key[1][0] * key[2][2] - key[1][2] * key[2][0]);
    adj[0][2] = (key[1][0] * key[2][1] - key[1][1] * key[2][0]);
    adj[1][0] = -(key[0][1] * key[2][2] - key[0][2] * key[2][1]);
    adj[1][1] = (key[0][0] * key[2][2] - key[0][2] * key[2][0]);
    adj[1][2] = -(key[0][0] * key[2][1] - key[0][1] * key[2][0]);
    adj[2][0] = (key[0][1] * key[1][2] - key[0][2] * key[1][1]);
    adj[2][1] = -(key[0][0] * key[1][2] - key[0][2] * key[1][0]);
    adj[2][2] = (key[0][0] * key[1][1] - key[0][1] * key[1][0]);
    vector<vector<int>> inv(3, vector<int>(3));
    for (int i = 0; i < 3; i++)
        for (int j = 0; j < 3; j++)
        {
            inv[i][j] = adj[j][i];
            inv[i][j] = (inv[i][j] * detInv) % 26;
            inv[i][j] = (inv[i][j] + 26) % 26;
        }
    return inv;
}

string encrypt(string text, vector<vector<int>> key) {
    string result = "";
```

```
    while (text.length() % 3 != 0)
        text += 'X';
    for (int i = 0; i < text.length(); i += 3)
    {
        vector<int> block(3);
        for (int j = 0; j < 3; j++)
            block[j] = text[i + j] - 'A';
        vector<int> enc = multiplyMatrix(key, block);
        for (int j = 0; j < 3; j++)
            result += (enc[j] + 'A');
    }
    return result;
}
string decrypt(string text, vector<vector<int>> key) {
    vector<vector<int>> invKey = inverseMatrix(key);
    string result = "";
    for (int i = 0; i < text.length(); i += 3)
    {
        vector<int> block(3);
        for (int j = 0; j < 3; j++)
            block[j] = text[i + j] - 'A';
        vector<int> dec = multiplyMatrix(invKey, block);
        for (int j = 0; j < 3; j++)
            result += (dec[j] + 'A');
    }
    return result;
```

```cpp
}

int main() {

   string plaintext;

   cout << "Enter plaintext: ";

   cin >> plaintext;

   vector<vector<int>> key = {

       {6, 24, 1},

       {13, 16, 10},

       {20, 17, 15}};

   string encrypted = encrypt(plaintext, key);

   cout << "Encrypted text: " << encrypted << endl;

   string decrypted = decrypt(encrypted, key);

   cout << "Decrypted text: " << decrypted << endl;

   return 0;

}
```

## OUTPUT:




## LEARNING OUTCOME:

To understand and implement Hill cipher encryption and decryption using matrix operations and modular arithmetic for secure data transmission.