

### Experiment no. 4

**Learning Objective:** Analyze and implement Polyalphabetic Ciphers

**Tools:** C/C++/Java/Python

**Theory:**

**Autokey Cipher:**

---

- It is a simple polyalphabetic cipher called the autokey cipher.
  - In this cipher, the key is a stream of subkeys, in which each subkey is used to encrypt the corresponding character in the plaintext.
  - The first subkey is a pre-determined value secretly agreed upon by Alice and Bob.
  - The second subkey is the value of the first plaintext character (between 0 and 25).
  - The third subkey is the value of the second plaintext. And so on.
  - The name of the cipher, autokey, implies that the subkeys are automatically created from the plaintext cipher characters during the encryption process.
- 

**Play Fair Cipher:**

---

The secret key in this cipher is made of 25 alphabet letters arranged in 5x5 matrix (letters I and J are considered the same when encrypting).

---

- Different arrangements of the letters in the matrix can create many different secret keys. One of the possible arrangements.
  - The ciphertext is actually the key stream:  $P = P_1P_2P_3\dots$   $C = C_1C_2C_3\dots$   $k = [(k_1,k_2),(k_3,k_4),\dots]$
  - Encryption:  $C_i = k_i$  Decryption:  $P_i = k_i$
  - Before encryption, if the two letters in a pair are the same, a bogus letter is inserted to separate them.
  - After inserting bogus letter, if the number of characters in the plaintext is odd, one extra bogus character is added at the end to make the number of characters even.
- 

The Cipher uses three Rules for Encryption:

---

- If the two letters in a pair are located in the same row of the secret key, the corresponding encrypted character for each letter is the next letter to the right in the same row (with wrapping to the beginning of the row if the plaintext letter is the last character in the row)
- If the two letters in a pair are located in the same column of the secret key, the corresponding encrypted character for each letter is the letter beneath it in the same column (with wrapping to the beginning of the column if the plaintext letter is the last column in the column)
- If the two letters in a pair are not in the same row or column of the secret, the corresponding encrypted character for each letter is a letter that is in its own row but in the same column as the other letter.

**Vignere Cipher:**

A Vigenere cipher uses a different strategy to create the key stream.

- The key stream is a repetition of an initial secret key stream of length  $m$ , where we have  $1 \leq m \leq 26$ .
- The ciphertext is actually the key stream:  $P = P_1P_2P_3\dots$   $C = C_1C_2C_3\dots$   $k = [(k_1, k_2, \dots, k_m), [(k_1, k_2, \dots, k_m), \dots]]$  Encryption:  $C_i = P_i + k_i$  Decryption:  $P_i = C_i - k_i$
- One important difference between the Vigenere cipher and the two other polyalphabetic ciphers is that Vigenere key stream does not depend on the plaintext characters; it depends only on the position of the character in the plaintext.
- In other words, the key stream can be created without knowing what the plaintext is.

**Code:**

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define SIZE 30
void toLowerCase(char plain[], int ps)
{
    int i;
    for (i = 0; i < ps; i++) {
        if (plain[i] > 64 && plain[i] < 91)
            plain[i] += 32;
    }
}
```

```
int removeSpaces(char* plain, int ps)
{
    int i, count = 0;
    for (i = 0; i < ps; i++)
        if (plain[i] != ' ')
            plain[count++] = plain[i];
    plain[count] = '\0';
    return count;
}

void generateKeyTable(char key[], int ks, char keyT[5][5])
{
    int i, j, k, flag = 0, *dicty;

    dicty = (int*)calloc(26, sizeof(int));
    for (i = 0; i < ks; i++) {
        if (key[i] != 'j')
            dicty[key[i] - 97] = 2;
    }

    dicty['j' - 97] = 1;

    i = 0;
    j = 0;

    for (k = 0; k < ks; k++) {
        if (dicty[key[k] - 97] == 2) {
            dicty[key[k] - 97] -= 1;
            keyT[i][j] = key[k];
            j++;
            if (j == 5) {
                i++;
                j = 0;
            }
        }
    }

    for (k = 0; k < 26; k++) {
```

---

```
if (dicty[k] == 0) {  
    keyT[i][j] = (char)(k + 97);  
    j++;  
    if (j == 5) {  
        i++;  
        j = 0;  
    }  
}  
}  
}  
  
void search(char keyT[5][5], char a, char b, int arr[])  
{  
    int i, j;  
  
    if (a == 'j')  
        a = 'i';  
    else if (b == 'j')  
        b = 'i';  
  
    for (i = 0; i < 5; i++) {  
  
        for (j = 0; j < 5; j++) {  
  
            if (keyT[i][j] == a) {  
                arr[0] = i;  
                arr[1] = j;  
            }  
            else if (keyT[i][j] == b) {  
                arr[2] = i;  
                arr[3] = j;  
            }  
        }  
    }  
}
```

---

```
int mod5(int a) { return (a % 5); }
int prepare(char str[], int ptrs)
{
    if (ptrs % 2 != 0) {
        str[ptrs++] = 'z';
        str[ptrs] = '\0';
    }
    return ptrs;
}

void encrypt(char str[], char keyT[5][5], int ps)
{
    int i, a[4];

    for (i = 0; i < ps; i += 2) {

        search(keyT, str[i], str[i + 1], a);

        if (a[0] == a[2]) {
            str[i] = keyT[a[0]][mod5(a[1] + 1)];
            str[i + 1] = keyT[a[0]][mod5(a[3] + 1)];
        }
        else if (a[1] == a[3]) {
            str[i] = keyT[mod5(a[0] + 1)][a[1]];
            str[i + 1] = keyT[mod5(a[2] + 1)][a[1]];
        }
        else {
            str[i] = keyT[a[0]][a[3]];
            str[i + 1] = keyT[a[2]][a[1]];
        }
    }
}

void encryptByPlayfairCipher(char str[], char key[])
{
    char ps, ks, keyT[5][5];
    ks = strlen(key);
    ks = removeSpaces(key, ks);
    toLowerCase(key, ks);
```

---

```
ps = strlen(str);
toLowerCase(str, ps);
ps = removeSpaces(str, ps);

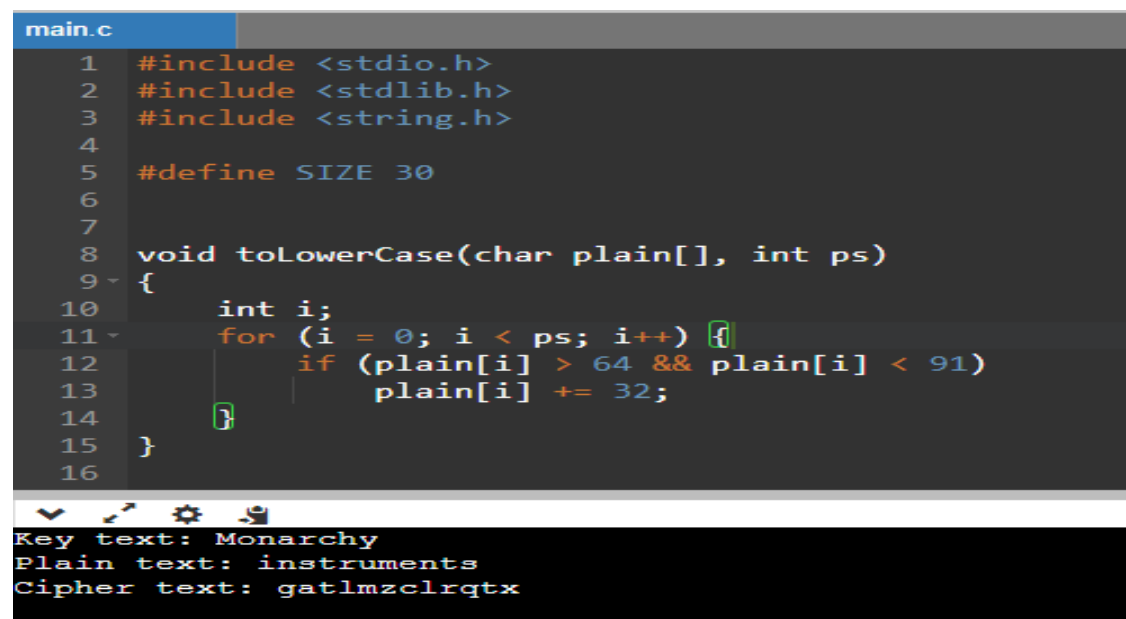
ps = prepare(str, ps);

generateKeyTable(key, ks, keyT);

encrypt(str, keyT, ps);
}
int main()
{
    char str[SIZE], key[SIZE];
    strcpy(key, "Monarchy");
    printf("Key text: %s\n", key);

    strcpy(str, "instruments");
    printf("Plain text: %s\n", str);
    encryptByPlayfairCipher(str, key);
    printf("Cipher text: %s\n", str);
    return 0;
}
```

- **Output:**



```
main.c
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h>
4
5  #define SIZE 30
6
7
8  void toLowerCase(char plain[], int ps)
9  {
10     int i;
11     for (i = 0; i < ps; i++) {
12         if (plain[i] > 64 && plain[i] < 91)
13             plain[i] += 32;
14     }
15 }
16
```

```
Key text: Monarchy
Plain text: instruments
Cipher text: gatlmzclrqtx
```

**Result and Discussion:**

The analysis and implementation of Polyalphabetic Ciphers demonstrate their effectiveness in enhancing encryption security through the use of multiple cipher alphabets. By employing techniques such as the Vigenère Cipher or Playfair Cipher, these ciphers provide resistance against frequency analysis attacks.

**Learning Outcomes:** The student will be able to:

LO1: Understand the Diffie-Hellman Key Exchange Algorithm

LO2: Analyze and implement the Diffie-Hellman Key Exchange Algorithm

**Course Outcomes:** Upon completion of the course students will be able to analyze and implement Diffie-Hellman Key Exchange Algorithm for generation of shared symmetric key.

**Conclusion:** It Reveal their effectiveness in providing stronger encryption compared to monoalphabetic ciphers. By employing multiple alphabets and shifting techniques, they significantly enhance the security of encrypted messages.

---

**Name:** Aryan Tiwari

**Class:** SE – CSE

**RollNo:** 57

**For Faculty Use**

Correction Parameters	Formative Assessment [40%]	Timely completion of Practical [ 40%]	Attendance / Learning Attitude [20%]	
Marks Obtained				