**PANDIT DEENDAYAL ENERGY UNIVERSITY**

**SCHOOL OF TECHNOLOGY**



**Course: Information Security**

**Course Code: 20CP304P**

**LAB MANUAL**

**B.Tech. (Computer Science and Engineering)**

**Semester 5**

|  |  |
| --- | --- |
| **Submitted To:**  Dr. Hargeet Kaur | **Submitted By:**  Khushi Yaduka  21BCP417  G-12 batch |

**INDEX**

|  |  |  |  |
| --- | --- | --- | --- |
| **S. No.** | **List of experiments** | **Date** | **Sign** |
| **1** | Download and Practise Cryptool. | 7/8/23 |  |
| **2** | Study and implement Caesar cipher with encryption, decryption. | 7/8/23 |  |
| **3** | Study and implement a program for Transposition(Columnar) cipher to encrypt and decrypt the message . | 14/8/23 |  |
| **4** | Study and implement a program for Rail Fence Transposition Cipher to encrypt and decrypt the message . | 21/8/23 |  |
| **5** | Study and implement a program for Vigenere cipher to encrypt and decrypt the message . | 28/8/23 |  |
| **6** | Study and implement a program for 5\*5 Playfair cipher. | 4/9/23 |  |
| **7** |  | 25/9/23 |  |
| **8** |  |  |  |
| **9** |  |  |  |
| **10** |  |  |  |

**Experiment No : 1**

**Aim** : Download and Practise Cryptool.

**Introduction** : CrypTool is an [open-source](https://en.wikipedia.org/wiki/Open-source_software) project that is a free [e-learning](https://en.wikipedia.org/wiki/E-learning) software for illustrating [cryptographic and cryptanalytic concepts](https://en.wikipedia.org/wiki/Cryptography). According to "Hakin9",CrypTool is worldwide the most widespread e-learning software in the field of [cryptology](https://en.wikipedia.org/wiki/Cryptology).

They contain most [classical ciphers](https://en.wikipedia.org/wiki/Classical_cipher), as well as modern symmetric and [asymmetric cryptography](https://en.wikipedia.org/wiki/Public-key_cryptography) including [RSA](https://en.wikipedia.org/wiki/RSA_(algorithm)), [ECC](https://en.wikipedia.org/wiki/Elliptic_curve_cryptography), [digital signatures](https://en.wikipedia.org/wiki/Digital_signature), hybrid encryption, [homomorphic encryption](https://en.wikipedia.org/wiki/Homomorphic_encryption), and [Diffie–Hellman key exchange](https://en.wikipedia.org/wiki/Diffie%E2%80%93Hellman_key_exchange). Methods from the area of [quantum cryptography](https://en.wikipedia.org/wiki/Quantum_cryptography) (like [BB84 key exchange protocol](https://en.wikipedia.org/wiki/BB84)) and the area of [post-quantum cryptography](https://en.wikipedia.org/wiki/Post-quantum_cryptography) (like [McEliece](https://en.wikipedia.org/wiki/McEliece_cryptosystem), WOTS, [Merkle-Signature-Scheme](https://en.wikipedia.org/wiki/Merkle_signature_scheme), [XMSS, XMSS\_MT, and SPHINCS](https://en.wikipedia.org/wiki/Hash-based_cryptography)) are implemented. In addition to the algorithms, solvers (analyzers) are included, especially for classical ciphers. Other methods (for instance [Huffman code](https://en.wikipedia.org/wiki/Huffman_coding), [AES](https://en.wikipedia.org/wiki/Advanced_Encryption_Standard), [Keccak](https://en.wikipedia.org/wiki/SHA-3), [MSS](https://en.wikipedia.org/wiki/Merkle_signature_scheme)) are visualized.

**1.Caeser Cipher**

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.

**2.Vigenere cipher**

Vigenere Cipher is a method of encrypting alphabetic text. It uses a simple form of polyalphabetic substitution. A polyalphabetic cipher is any cipher based on substitution, using multiple substitution alphabets. The table consists of the alphabets written out 26 times in different rows, each alphabet shifted cyclically to the left compared to the previous alphabet, corresponding to the 26 possible Cea sar Ciphers.At different points in the encryption process, the cipher uses a different alphabet from one of the rows.The alphabet used at each point depends on a repeating keyword.

**3.Vernam Cipher**

Vernam Cipher is a method of encrypting alphabetic text. It is one of the Substitution techniques for converting plain text into cipher text. We take a key to encrypt the plain text whose length should be equal to the length of the plain text.Assign a number to each character of the plain-text and the key according to alphabetical order. Bitwise XOR both the number (Corresponding plain-text character number and Key character number). Subtract the number from 26 if the resulting number is greater than or equal to 26, if it isn’t then leave it.

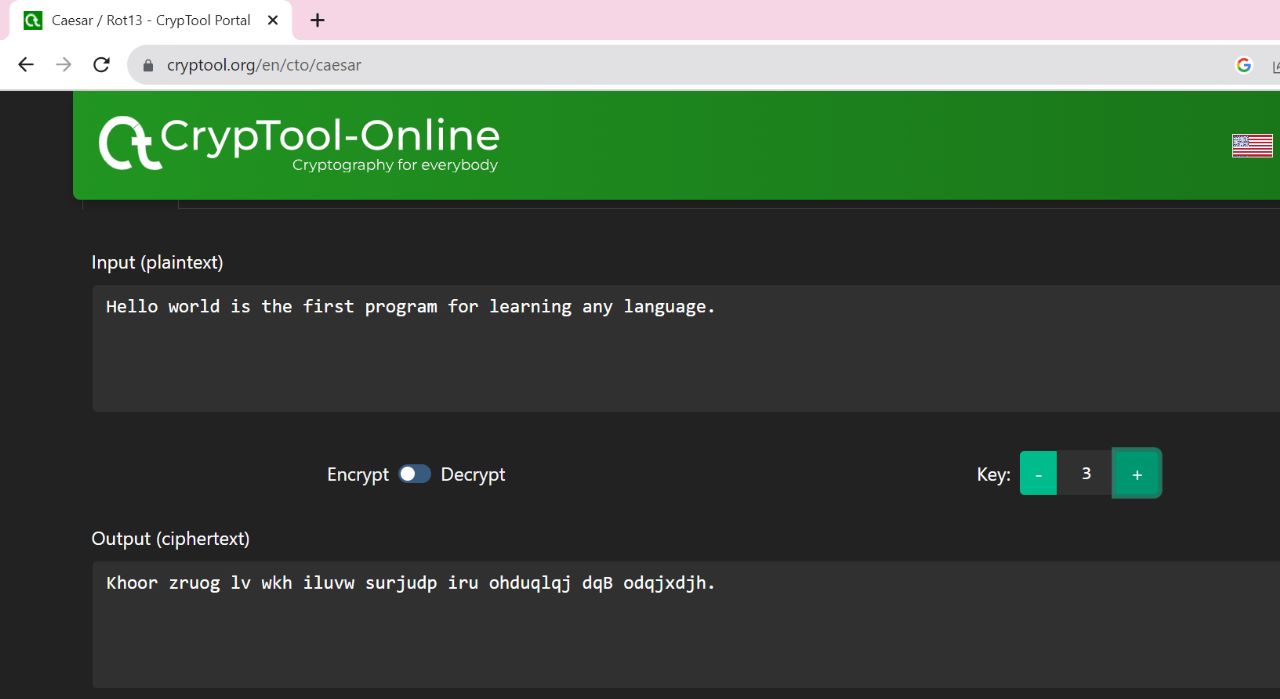
**4.Simple Columnar Transposition**

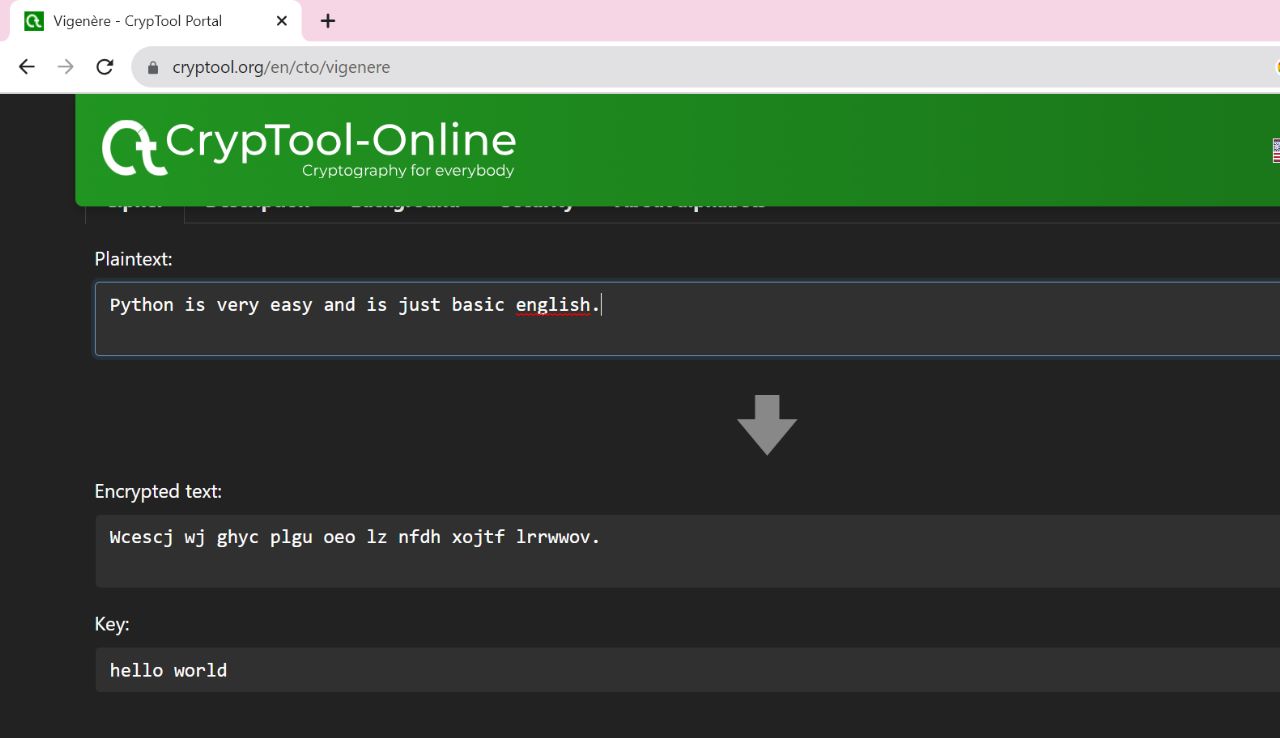
In a transposition cipher, the order of the alphabets is re-arranged to obtain the cipher-text.The message is written out in rows of a fixed length, and then read out again column by column, and the columns are chosen in some scrambled order.Width of the rows and the permutation of the columns are usually defined by a keyword.The permutation is defined by the alphabetical order of the letters in the keyword. Any spare spaces are filled with nulls or left blank or placed by a character .Finally, the message is read off in columns, in the order specified by the keyword.

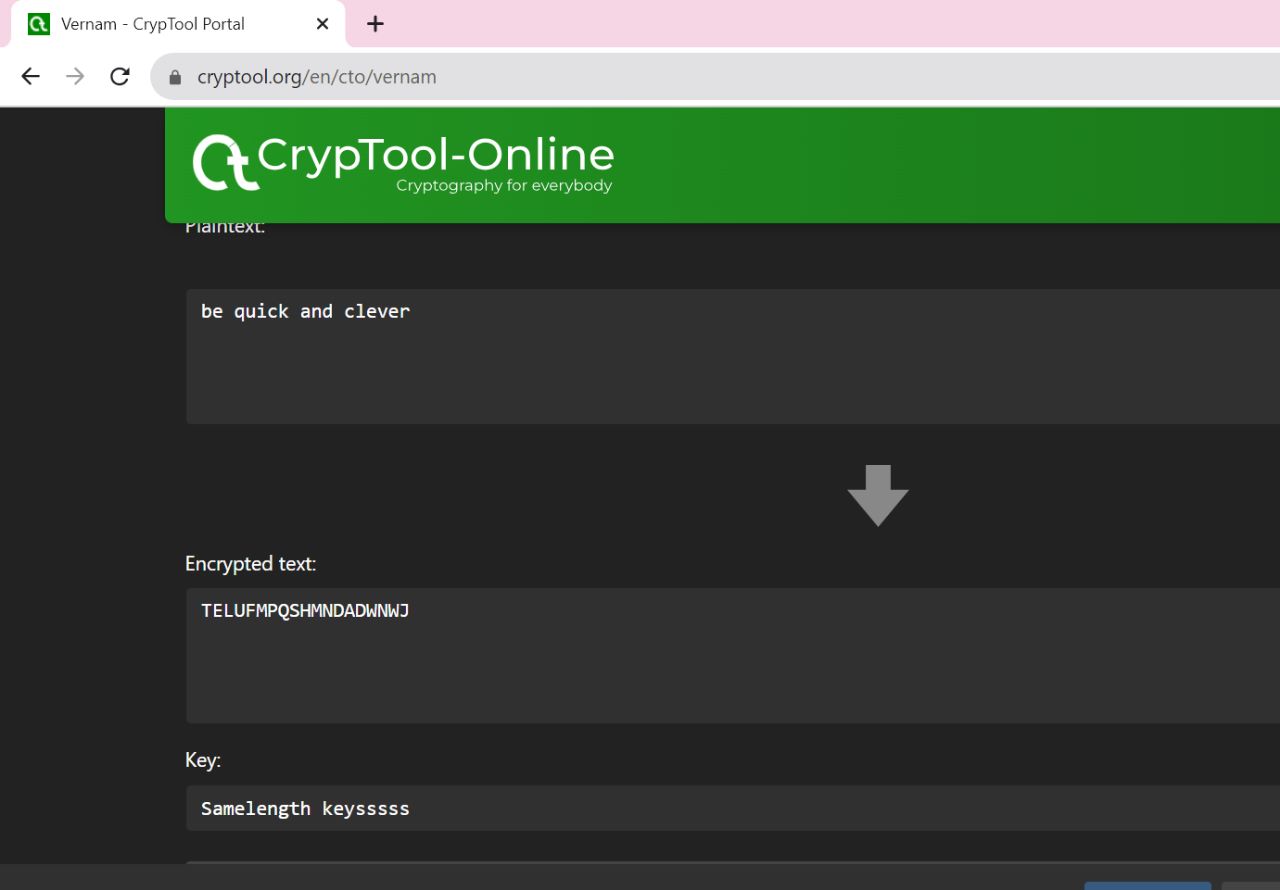
**5.Mono Alphabetic Substition**

A monoalphabetic cipher is any cipher in which the letters of the plain text are mapped to cipher text letters based on a single alphabetic key. Examples of monoalphabetic ciphers would include the Caesar-shift cipher, where each letter is shifted based on a numeric key, and the atbash cipher, where each letter is mapped to the letter symmetric to it about the center of the alphabet.

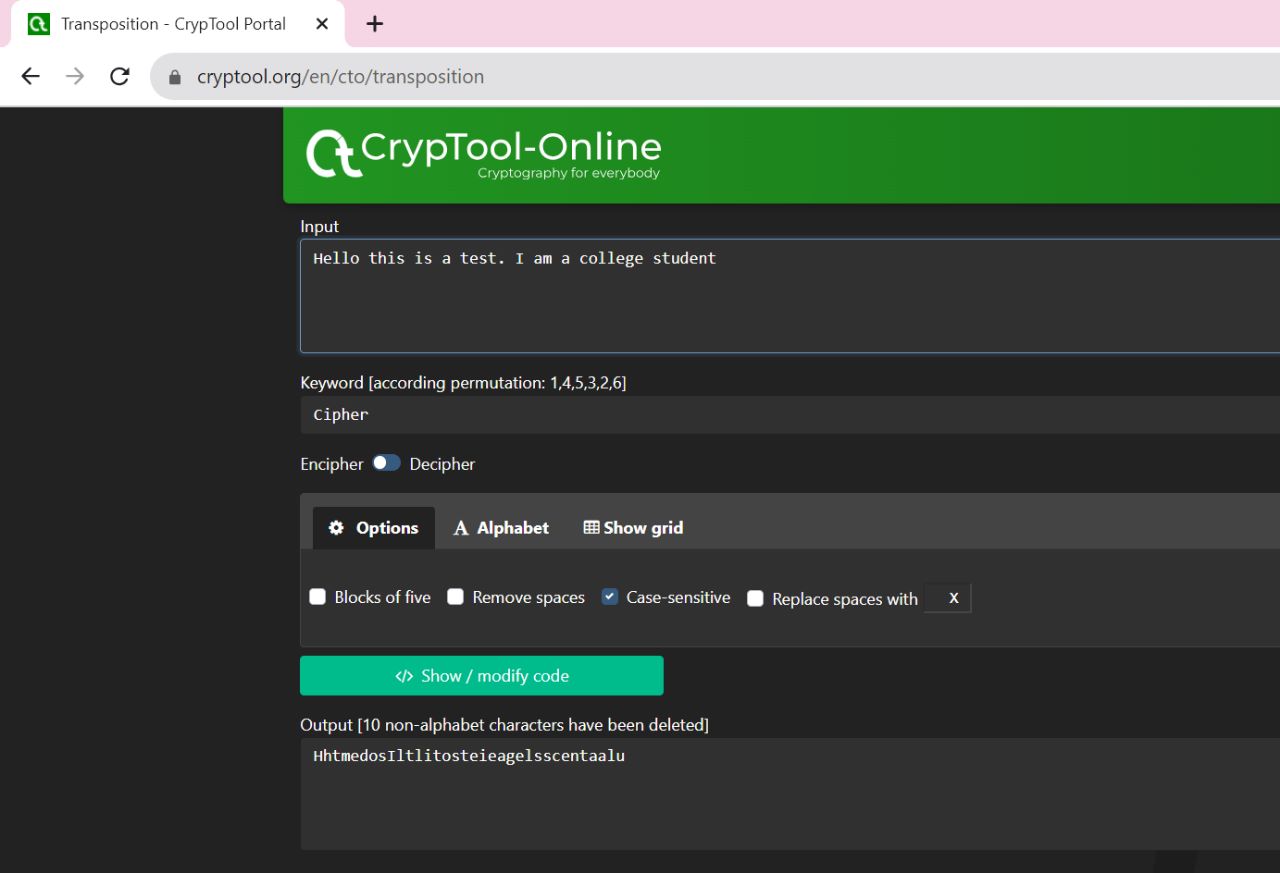
**Output:**

**Caeser Cipher** 

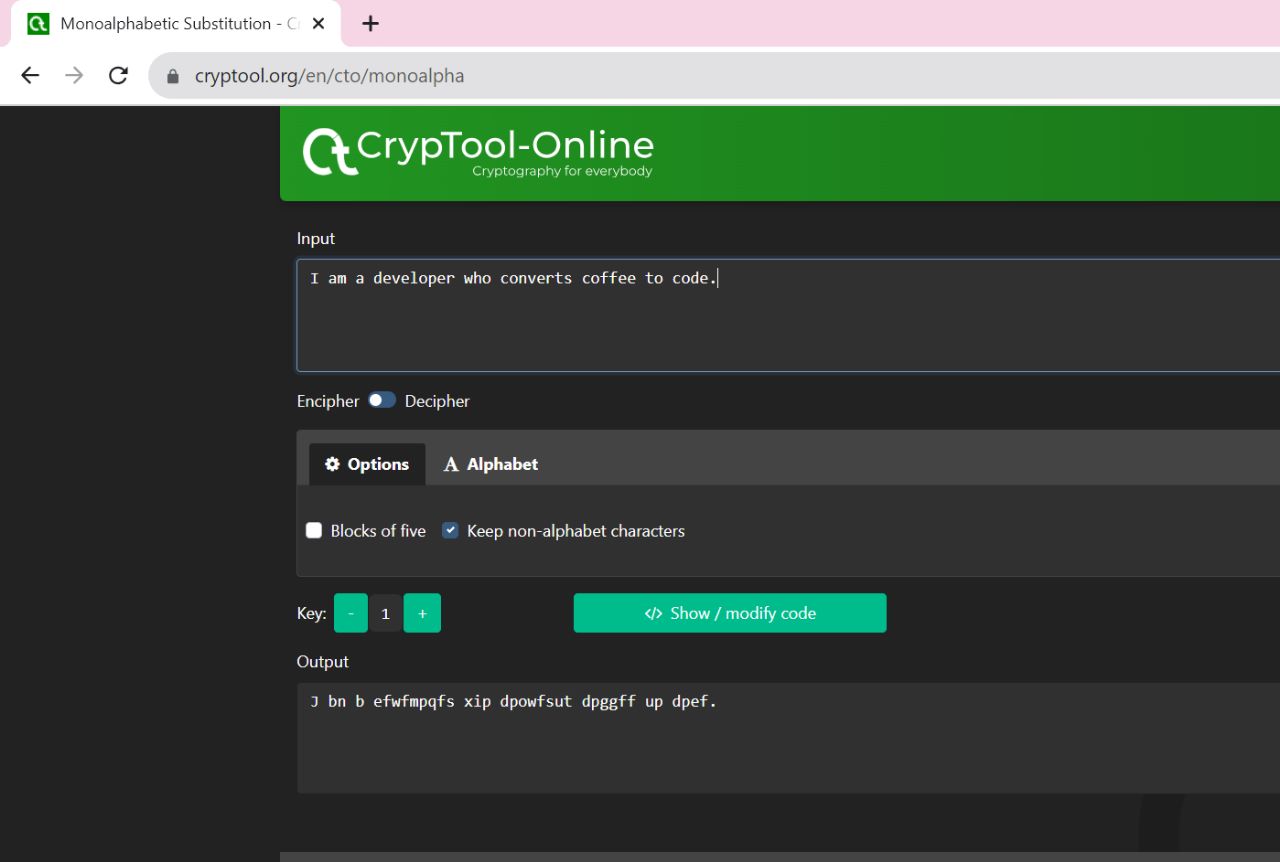
**Vigenere cipher**



**Vernam Cipher**



**Simple Columnar Transposition**



**Mono Alphabetic Substition**

**References :**

1.Wikipedia(https://en.wikipedia.org/wiki/CrypTool#:~:text=CrypTool%20is%20an%20open%2Dsource,in%20the%20field%20of%20cryptology.&text=CrypTool%20implements%20more%20than%20400%20algorithms.)

2. GeeksforGeeks(https://www.geeksforgeeks.org/caesar-cipher-in-cryptography/)

**EXPERIMENT NO: 2**

**Aim:** Study and implement Caesar cipher with encryption, decryption.

**Introduction:** The Caesar cipher is one of the earliest and simplest forms of encryption. This is a type of substitution cipher, where each character in the original message is moved to a number of places below the line.

For example, if there were 3 changes, A would be replaced by D, B by E, C by F, and so on. The first messages will be saved as "HELLO" and "KHOOR".

To encrypt a message using a Caesar cipher:

* Select shift number - This is the number of positions you shift each character in the list. They are typically 3 or 4 inches.
* Write a simple text message.
* Replace each letter in the message with the number you selected. So if the shift is 3, then A becomes D, B E and so on.
* The resulting ciphertext is an encrypted message.

To dig into the Caesar Cipher:

* Displays the shift number originally used. Maybe they know this or maybe you should wonder. Take the cipher text and shift each character back to A by shift number. So if the variable was 3, then D would be A, E would be B, and so on.
* The resulting plaintext is the extracted original message.
* The Caesar cipher is easy to crack because there are only 25 possible changes. It does not offer the worst protection and is of particular historical interest today. However, it represents an early attempt to use alternatives to protect the message.

**Program Code:**

#include <iostream>

using namespace std;

string encryptMessage(const string message, int key) {

    string encryptedMessage = "";

    for (char ch : message) {

        if (islower(ch)) {

            ch = (ch - 'a' + key) % 26 + 'a';

        } else if (ch == ' ') {

            encryptedMessage += ch;

            continue;

        } else {

            cout << "Invalid Message\n";

            return "";

        }

        encryptedMessage += ch;

    }

    return encryptedMessage;

}

string decryptMessage(const string message, int key) {

    return encryptMessage(message, 26 - key);

}

int main() {

    string text;

    int key;

    cout << "Enter a message to encrypt: ";

    getline(cin, text);

    cout << "Enter the key: ";

    cin >> key;

    string encryptedText = encryptMessage(text, key);

    if (!encryptedText.empty()) {

        cout << "Encrypted message: " << encryptedText << endl;

        string decryptedText = decryptMessage(encryptedText, key);

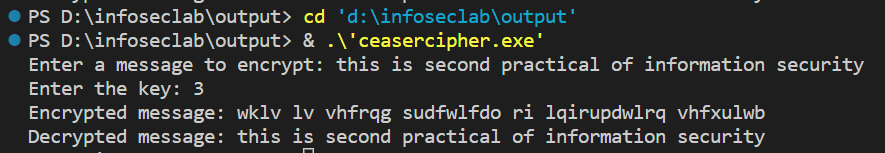
        cout << "Decrypted message: " << decryptedText << endl;

    }

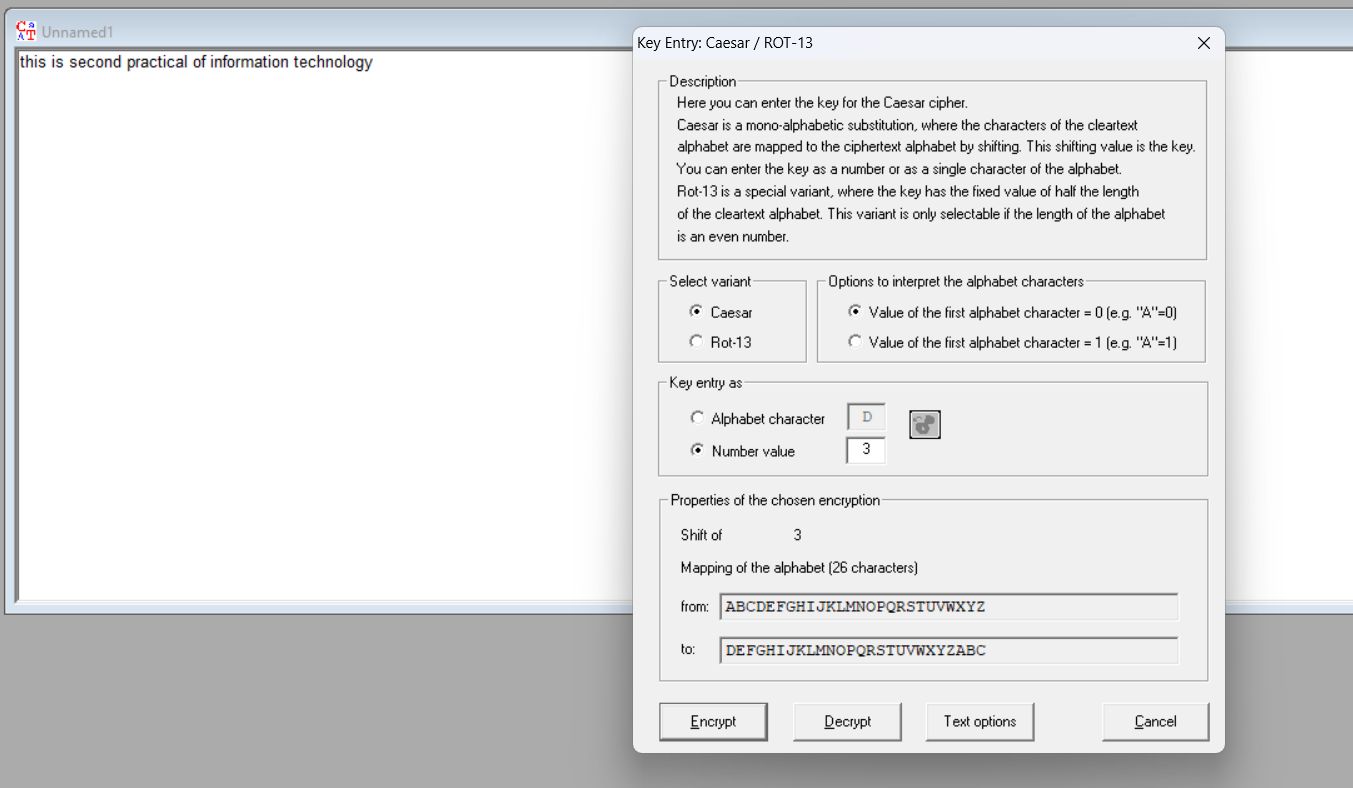
    return 0;

}

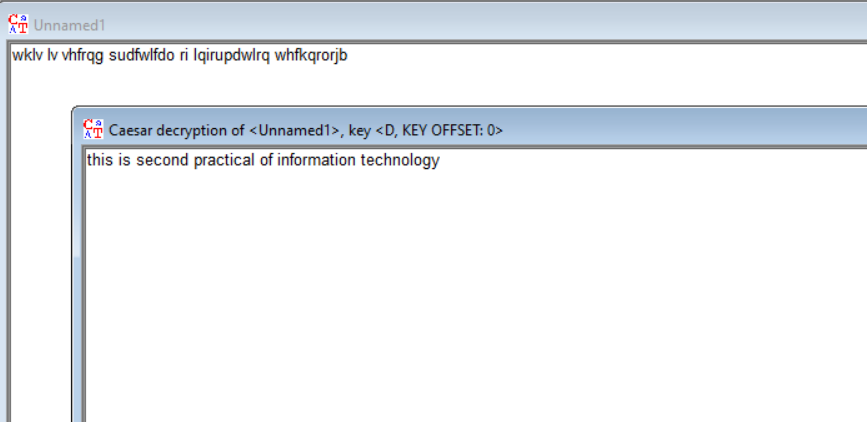
**Program Output(IDE):**

****

**Program Output (Cryptool):**





****

**Cryptanalysis:**

Time complexity:

The time complexity of the Caesar cipher is O(n), where 'n' is the length of the input message (the number of characters in the plaintext or ciphertext). The number of characters processed is directly proportional to the length of the input message, the time complexity is linear, or O(n). Regardless of the shift value, the time complexity remains the same because the same amount of work is done for each character.

Advantages:

* Simple to implement - it just requires substituting each letter with another shifted letter. Easy to do by hand.
* Fast to encrypt and decrypt a message.
* Provides a slight amount of obfuscation of the original message.

Disadvantages:

* Very weak security - there are only 25 possible shifts so it is easy to brute force and crack the cipher.
* The frequency analysis of the cipher text allows the cipher to be cracked easily. More common letters like 'E' and 'T' will still appear frequently.
* The shift is reusable and needs to be protected. If the shift is known, the cipher can be decrypted trivially.
* It only works well for English text, not for other languages or symbols.
* Does not provide confidentiality, integrity, and authenticity in a message.

**Applications:**

* As a simple method to add basic encryption to messages. For example, Julius Caesar is said to have used it to communicate with his generals.
* As a teaching tool to illustrate encryption concepts. It provides a simple substitution cipher to demonstrate core encryption principles.
* To obscure words or passages in media to avoid filters or spoiling plot points.
* To provide a layer of simple obscurity for things like website or product codes.

**References:**

* Wikipedia(https://en.wikipedia.org/wiki/Caesar\_cipher)
* GeeksforGeeks(https://www.geeksforgeeks.org/caesar-cipher-in-cryptography/)

**EXPERIMENT NO: 3**

**Aim:** Study and implement a program for Transposition(Columnar) cipher to encrypt and decrypt the message .

**Introduction:** In a columnar transposition, the message is written out in rows of a fixed length, and then read out again column by column, and the columns are chosen in some scrambled order. Both the width of the rows and the permutation of the columns are usually defined by a keyword.

In a transposition cipher, the order of the alphabets is re-arranged to obtain the cipher-text.

* The message is written out in rows of a fixed length, and then read out again column by column, and the columns are chosen in some scrambled order.
* Width of the rows and the permutation of the columns are usually defined by a keyword.
* Any spare spaces are filled with nulls or left blank or placed by a character (Example: \_).
* Finally, the message is read off in columns, in the order specified by the keyword.

To decipher it, the recipient has to work out the column lengths by dividing the message length by the key length. Then they can write the message out in columns again, then re-order the columns by reforming the key word.

**Program Code:**

#include <bits/stdc++.h>

using namespace std;

string encryptSimpleColumnTransposition(const string& plainText, const string& key) {

    string encryptedText;

    int keyLength = key.length();

    int textLength = plainText.length();

    int numRows = (textLength + keyLength - 1) / keyLength;

    char matrix[numRows][keyLength];

    int index = 0;

    for (int i = 0; i < numRows; i++) {

        for (int j = 0; j < keyLength; j++) {

            if (index < textLength) {

                matrix[i][j] = plainText[index++];

            } else {

                matrix[i][j] = ' ';

            }

        }

    }

    string sortedKey = key;

    sort(sortedKey.begin(), sortedKey.end());

    int permutationIndex[keyLength];

    for (int i = 0; i < keyLength; i++) {

        permutationIndex[i] = key.find(sortedKey[i]);

    }

    for (int i = 0; i < keyLength; i++) {

        for (int j = 0; j < numRows; j++) {

            encryptedText += matrix[j][permutationIndex[i]];

        }

    }

    return encryptedText;

}

string decryptSimpleColumnTransposition(const string& encryptedText, const string& key) {

    string decryptedText;

    int keyLength = key.length();

    int textLength = encryptedText.length();

    int numRows = (textLength + keyLength - 1) / keyLength;

    char matrix[numRows][keyLength];

    string sortedKey = key;

    sort(sortedKey.begin(), sortedKey.end());

    int permutationIndex[keyLength];

    for (int i = 0; i < keyLength; i++) {

        permutationIndex[i] = key.find(sortedKey[i]);

    }

    int index = 0;

    for (int i = 0; i < keyLength; i++) {

        for (int j = 0; j < numRows; j++) {

            matrix[j][permutationIndex[i]] = encryptedText[index++];

        }

    }

    for (int i = 0; i < numRows; i++) {

        for (int j = 0; j < keyLength; j++) {

            decryptedText += matrix[i][j];

        }

    }

    return decryptedText;

}

int main() {

    string plainText;

    string key;

    cout << "Enter the plain text: ";

    getline(cin, plainText);

    cout << "Enter the key: ";

    getline(cin, key);

    string encryptedText = encryptSimpleColumnTransposition(plainText, key);

    cout << "Encrypted Text: " << encryptedText << endl;

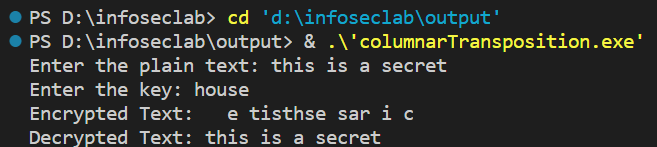
    string decryptedText = decryptSimpleColumnTransposition(encryptedText, key);

    cout << "Decrypted Text: " << decryptedText << endl;

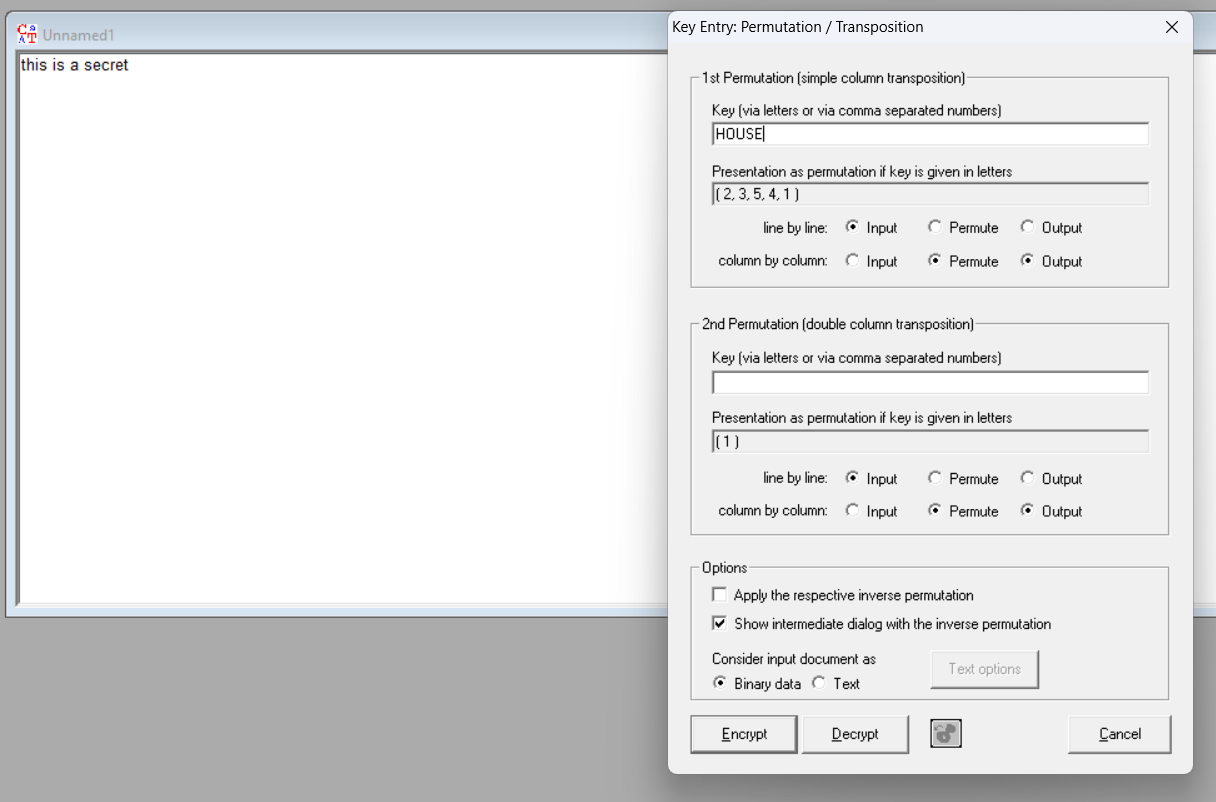
    return 0;

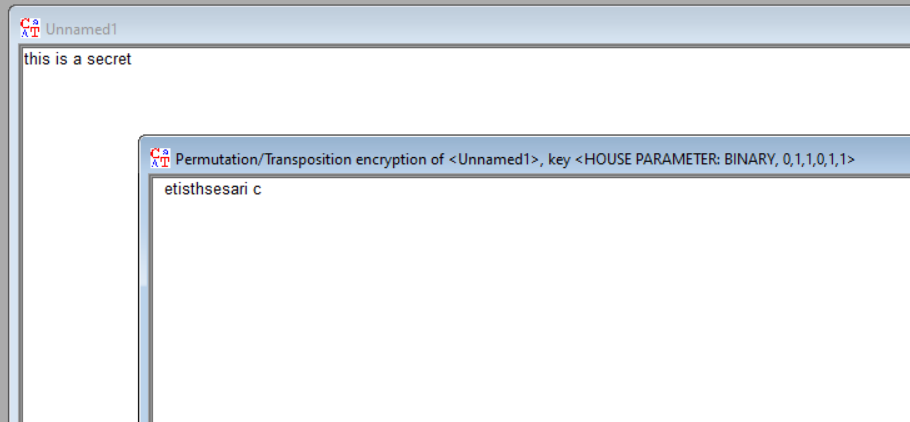
}

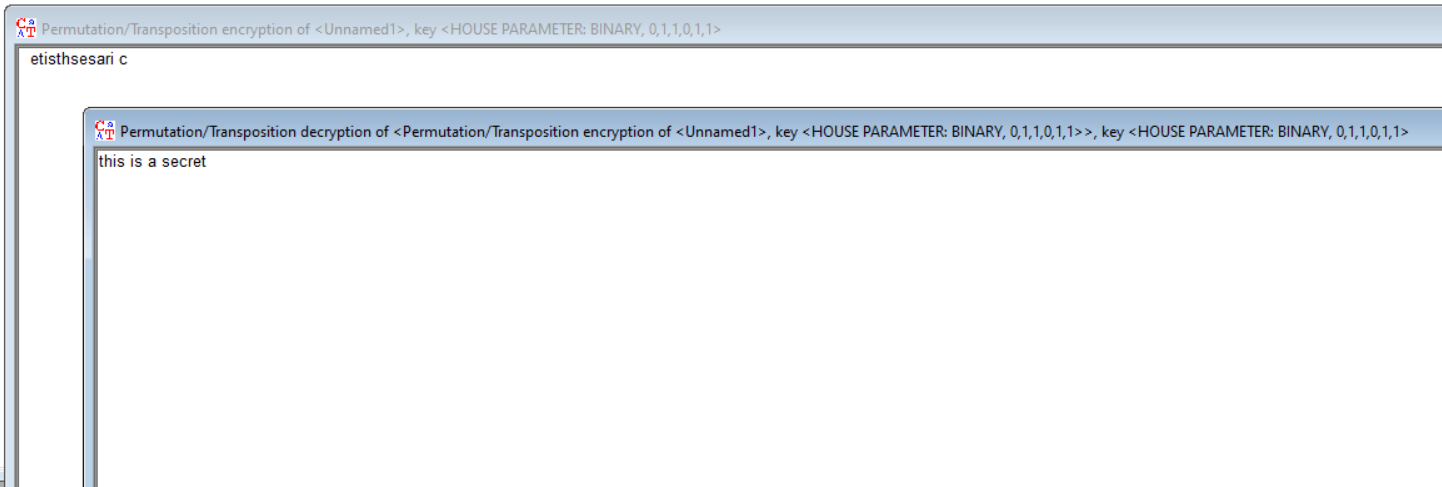
**Program Output(IDE):**

****

**Program Output (Cryptool):**

****

****

****

**Cryptanalysis:**

**Advantages of Columnar Transposition Cipher:**

* **Key Length:** Increasing the key length (number of columns) enhances the security of the cipher. Longer keys make brute force attacks more challenging.
* **Confusion :** The cipher exhibits confusion (hiding the relationship between plaintext and ciphertext) making it resistant to simple attacks.
* **Message Length Hiding:** The use of null characters or spaces to fill in the matrix hides the true length of the message, making it more challenging for attackers to determine the length of the original message.

**Disadvantages of Columnar Transposition Cipher:**

* **Vulnerable to Frequency Analysis:** While the cipher disrupts the frequency analysis of individual letters, longer repeated sequences or known plaintext can still provide enough information for frequency analysis to be effective.
* **Short Key Lengths:** If the key length is short, the number of possible key permutations is limited, making brute force attacks feasible.
* **Patterns in Ciphertext:** Depending on the key arrangement and original plaintext, there might be patterns in the ciphertext that attackers can exploit.

**Time Complexity:**

**Encrypt(encryptSimpleColumnTransposition):**

**Creating a matrix** requires iterating through each cell of the matrix, whose dimensions are **numRows** (indicated by message and key length) and **keyLength** (indicated by key length) This takes **O(numRows \* keyLength)** ) time.

**Formatting a key:** The key formation step takes **O(keyLength \* log(keyLength)**) time, which is the complex time for formatting keyLength objects.

**Creating an encrypted text:** Creating an encrypted text requires iterating through each column of the matrix (**keyLength** column), and for each column, to pass through **numRows** rows this takes **O(keyLength \* numRows)** ) time.

Overall, the main thing is that the matrix is ​​being built, which has a time complexity of **O(numRows \* keyLength).**

**Decrypt ( decryptSimpleColumnChange ):** . Building a matrix: Like encryption, it requires reconstruction of each matrix cell with **numRows** and **keyLength** dimensions, resulting in a time complexity of **O(numRows \* keyLength).**

**Key formation:** Key formation takes **O(keyLength \* log(keyLength)** time.

**Creating decrypted text:** Like encryption, creating decrypted text requires iterating through each row of the matrix (**numRows** rows), and in each row, passing through the **keyLength** columns takes **O(keyLength \* numRows)** time.

Overall, again, the key is building the matrix, which has a time complexity of **O(numRows \* keyLength).**

The time constraint for both encryption and decryption is the creation of the matrix, which takes **O(numRows \* keyLength)** time. Configuring the key and creating encryption or decryption also helps save time, but often.

**Applications:**

* **Steganography:** Steganography is the practice of concealing information within other, seemingly innocuous data. The columnar transposition cipher can be used to create steganographic messages by embedding a secret message within a larger text, making it less conspicuous.
* **Cryptography Education:** While not suitable for secure encryption in modern times, the columnar transposition cipher can be used as a stepping stone to teach basic cryptography concepts before moving on to more complex ciphers and encryption techniques.
* **Basic Message Privacy:** While not suitable for high-security scenarios, the columnar transposition cipher can provide a basic level of privacy for casual communication among individuals who are not well-versed in cryptography.
* **Puzzle and Games:** The cipher can be used in puzzles, escape room challenges, or treasure hunts. Players need to decrypt messages to advance in the game or solve puzzles.

**References:**

* Wikipedia(https://en.wikipedia.org/wiki/Transposition\_cipher)
* GeeksforGeeks(<https://www.geeksforgeeks.org/columnar-transposition-cipher/>

**EXPERIMENT NO: 4**

**Aim:** Study and implement a program for Rail Fence Transposition Cipher to encrypt and decrypt the message .

**Introduction:** The rail fence cipher (also called a zigzag cipher) is a form of transposition cipher. It derives its name from the way in which it is encoded. In a transposition cipher, the order of the alphabets is re-arranged to obtain the cipher-text.

Encryption

In the rail fence cipher, the plain-text is written downwards and diagonally on successive rails of an imaginary fence.

When we reach the bottom rail, we traverse upwards moving diagonally, after reaching the top rail, the direction is changed again. Thus the alphabets of the message are written in a zig-zag manner.

After each alphabet has been written, the individual rows are combined to obtain the cipher-text.

Decryption

As we’ve seen earlier, the number of columns in rail fence cipher remains equal to the length of plain-text message. And the key corresponds to the number of rails.

Hence, rail matrix can be constructed accordingly. Once we’ve got the matrix we can figure-out the spots where texts should be placed (using the same way of moving diagonally up and down alternatively ).

Then, we fill the cipher-text row wise. After filling it, we traverse the matrix in zig-zag manner to obtain the original text.

**Program Code:**

#include <bits/stdc++.h>

using namespace std;

string encrypt(string text, int rails) {

    char railMatrix[rails][text.length()];

    for (int i = 0; i < rails; i++) {

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

            railMatrix[i][j] = '.';

        }

    }

    int row = 0;

    int direction = 1; // 1 for moving down, -1 for moving up

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

        railMatrix[row][i] = text[i];

        if (row == 0) {

            direction = 1;

        } else if (row == rails - 1) {

            direction = -1;

        }

        row += direction;

    }

    string encryptedText = "";

    for (int i = 0; i < rails; i++) {

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

            if (railMatrix[i][j] != '.') {

                encryptedText += railMatrix[i][j];

            }

        }

    }

    return encryptedText;

}

string decrypt(string text, int rails) {

    char railMatrix[rails][text.length()];

    for (int i = 0; i < rails; i++) {

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

            railMatrix[i][j] = '.';

        }

    }

    int row = 0;

    int direction = 1;

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

        railMatrix[row][i] = '\*';

        if (row == 0) {

            direction = 1;

        } else if (row == rails - 1) {

            direction = -1;

        }

        row += direction;

    }

    int index = 0;

    for (int i = 0; i < rails; i++) {

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

            if (railMatrix[i][j] == '\*' && index < text.length()) {

                railMatrix[i][j] = text[index++];

            }

        }

    }

    string decryptedText = "";

    row = 0;

    direction = 1;

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

        decryptedText += railMatrix[row][i];

        if (row == 0) {

            direction = 1;

        } else if (row == rails - 1) {

            direction = -1;

        }

        row += direction;

    }

    return decryptedText;

}

int main() {

    string text;

    int rails;

    cout << "Enter the text to encrypt: ";

    getline(cin, text);

    cout << "Enter the number of rails: ";

    cin >> rails;

    string encryptedText = encrypt(text, rails);

    cout << "Encrypted: " << encryptedText << endl;

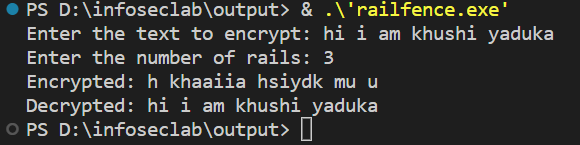
    string decryptedText = decrypt(encryptedText, rails);

    cout << "Decrypted: " << decryptedText << endl;

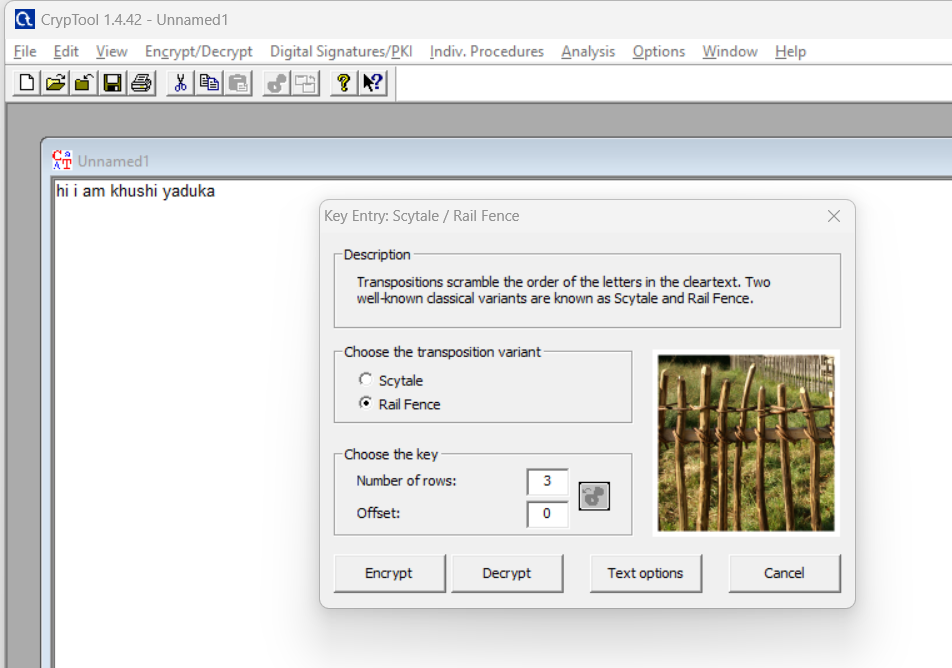
    return 0;

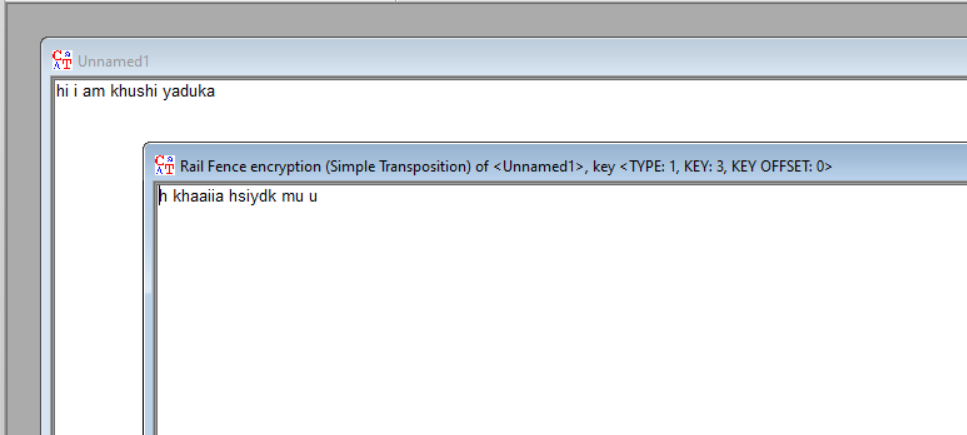
}

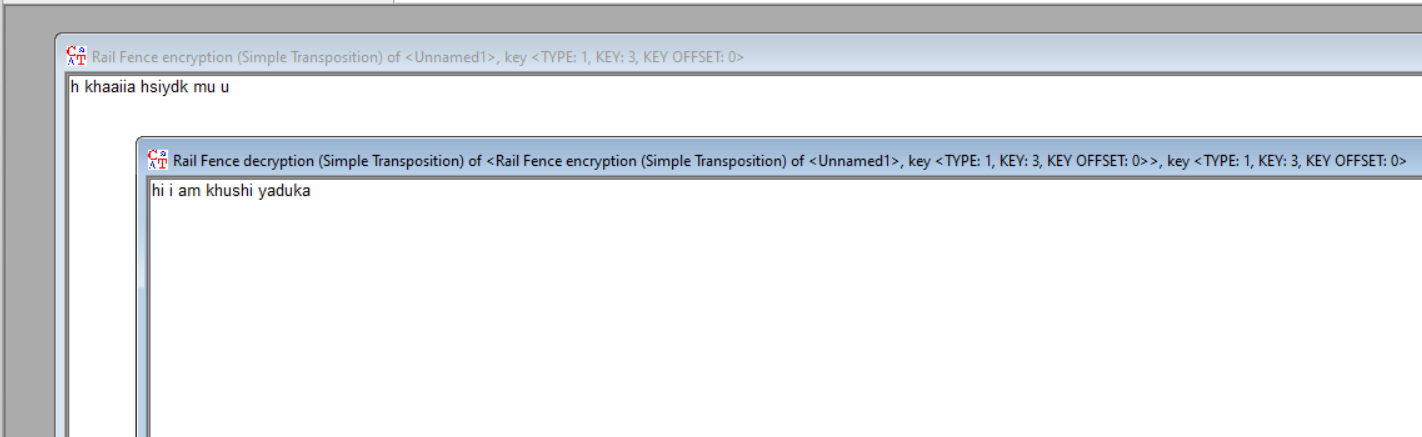
**Program Output(IDE):**

****

**Program Output (Cryptool):**

****

****

****

**Cryptanalysis:**

**Advantages of Railfence Transposition Cipher:**

**Quick Encryption**: Encrypting a message using the Rail Fence cipher is relatively fast and doesn't require complex mathematical operations or extensive computation.

**Visual Complexity**: The encrypted message has a visually complex appearance with diagonal lines that can make it appear more secure than it actually is, which might deter casual eavesdroppers.

**Variable Security**: The security of the Rail Fence cipher can be increased by using a larger number of rails, making it harder to decipher manually.

**Disadvantages of Railfence Transposition Cipher:**

**Weak Security**: The Rail Fence cipher offers weak security because it is vulnerable to various cryptographic attacks, including brute force methods. With modern computing power, it can be easily cracked.

**Low Key Space**: The key space (possible number of keys) for the Rail Fence cipher is relatively small, making it susceptible to exhaustive search attacks.

**Key Sharing Challenges**: Sharing the key securely can be difficult, as there is often no clear indication of what the correct number of rails should be.

**Time Complexity:**

**The complexity of the rail fence technique depends on the number of rows used to write out the message.** The more rows we use, the more complex the encryption will be. Additionally, increasing the number of rows can make it hard for an attacker to determine the original positions of the letters in the message. Time Complexity = O(Row\*Col).

**Applications:**

* **Pen-and-Paper Puzzles:** The Rail Fence Cipher is often used in puzzles and games to challenge participants to decrypt a message as part of a brainteaser or scavenger hunt.
* **Education:** It is used as a teaching tool to introduce students to the concept of cryptography and basic encryption techniques due to its simplicity and ease of understanding.
* **Steganography:** The Rail Fence Cipher can be used as a form of steganography, where it's used to hide messages within other documents or images. It can be a simple way to obscure a message within a larger body of text or data.
* **Recreational Cryptography:** Enthusiasts sometimes use it for recreational or hobbyist cryptography projects and challenges. It can be a fun way to create and solve puzzles.

**References:**

* GeeksforGeeks(<https://www.geeksforgeeks.org/rail-fence-cipher-encryption-decryption/>)
* https://www.baeldung.com/cs/cryptography-rail-fence-technique

**EXPERIMENT NO: 5**

**Aim:** Study and implement a program for Vigenere cipher to encrypt and decrypt the message .

**Introduction:** Vigenere Cipher is a method of encrypting alphabetic text. It uses a simple form of polyalphabetic substitution. A polyalphabetic cipher is any cipher based on substitution, using multiple substitution alphabets.The encryption of the original text is done using the Vigenère square or Vigenère table.

A more easy implementation could be to visualize Vigenère algebraically by converting [A-Z] into numbers [0–25].

**Encryption**

The plaintext(P) and key(K) are added modulo 26.

Ei = (Pi + Ki) mod 26

**Decryption**

Di = (Ei - Ki) mod 26

**Program Code:**

#include <bits/stdc++.h>

using namespace std;

string vigenereEncrypt(const string &plainText, const string &key) {

    string encryptedText = "";

    for (int i = 0, j = 0; i < plainText.length(); ++i) {

        char plainChar = plainText[i];

        char keyChar = key[j % key.length()];

        if (islower(plainChar)) {

            char base = 'a';

            char encryptedChar = (plainChar + keyChar - 2 \* base) % 26 + base;

            encryptedText += encryptedChar;

            ++j;

        } else {

            encryptedText += plainChar;

        }

    }

    return encryptedText;

}

string vigenereDecrypt(const string &encryptedText, const string &key) {

    string decryptedText = "";

    for (int i = 0, j = 0; i < encryptedText.length(); ++i) {

        char encryptedChar = encryptedText[i];

        char keyChar = key[j % key.length()];

        if (islower(encryptedChar)) {

            char base = 'a';

            char decryptedChar = (encryptedChar - keyChar + 26) % 26 + base;

            decryptedText += decryptedChar;

            ++j;

        } else {

            decryptedText += encryptedChar;

        }

    }

    return decryptedText;

}

int main() {

    string plainText, key;

    cout << "Enter the plaintext: ";

    getline(cin, plainText);

    cout << "Enter the key: ";

    getline(cin, key);

    string encryptedText = vigenereEncrypt(plainText, key);

    string decryptedText = vigenereDecrypt(encryptedText, key);

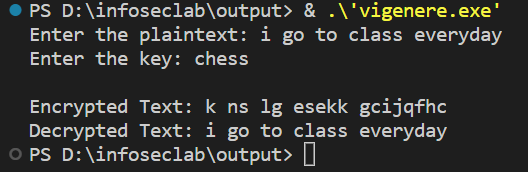
    cout << "\nEncrypted Text: " << encryptedText << endl;

    cout << "Decrypted Text: " << decryptedText << endl;

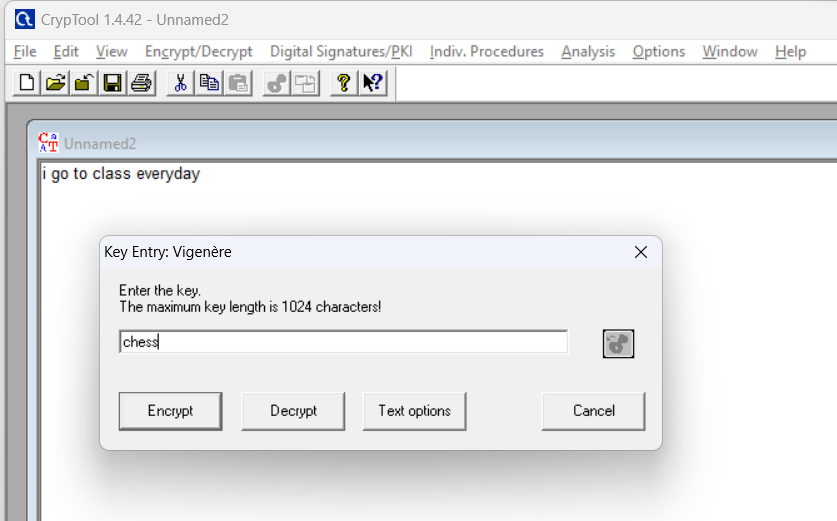
    return 0;

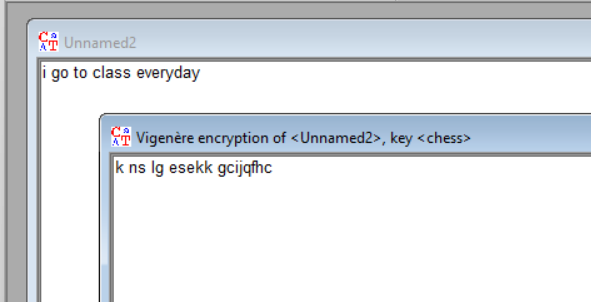
}

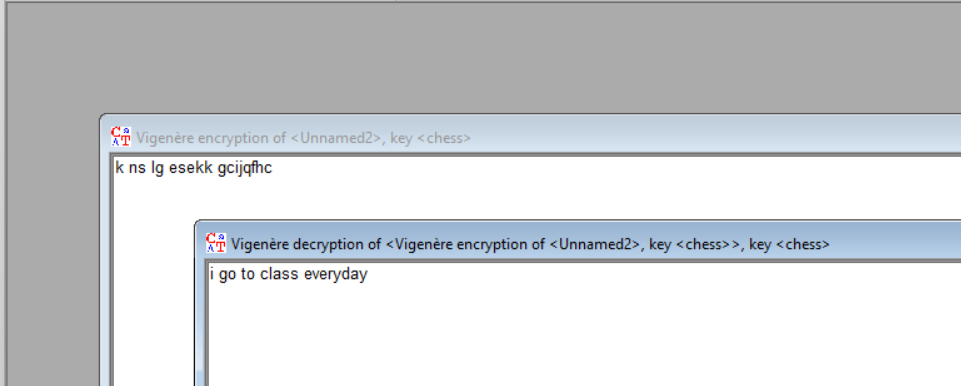
**Program Output(IDE):**

****

**Program Output (Cryptool):**

****

****

****

**Cryptanalysis:**

**Advantages of Vigenere Cipher:**

* **Polyalphabetic Substitution:** Unlike simple Caesar ciphers, which use a single fixed shift value for all characters, the Vigenère cipher uses a keyword to create different shift values for different positions in the plaintext. This makes it more complex and resistant to simple frequency analysis.
* **Variable Key Length:** The Vigenère cipher allows for the use of variable-length keywords, which can make it more adaptable and potentially more secure.
* **Historical Significance:** The Vigenère cipher has historical significance and was used for centuries, including by famous historical figures. It's interesting from a historical and educational perspective.

**Disadvantages of Vigenere Cipher :**

* **Repeating Key length:** The primary weakness of the Vigenère cipher is the repeating nature of its [key](https://en.wikipedia.org/wiki/Key_(cryptography)). If a cryptanalyst correctly guesses the key's length *n*, the cipher text can be treated as *n* interleaved [Caesar ciphers](https://en.wikipedia.org/wiki/Caesar_cipher), which can easily be broken individually. The key length may be discovered by [brute force](https://en.wikipedia.org/wiki/Brute-force_attack) testing each possible value of *n.*
* **Vulnerable to Frequency Analysis:** While the Vigenère cipher is more secure than simple Caesar ciphers, it is still vulnerable to frequency analysis. With a long enough ciphertext, attackers can still analyze patterns in the frequency of letters and potentially crack the code.
* **Weak Security for Modern Purposes:** In the context of modern cryptography, the Vigenère cipher is considered weak and insecure. It can be easily broken with modern computer-based methods, including brute force attacks or using known-plaintext attacks.

**Time Complexity:**

Time complexity of Vigenere Cipher is **O(n),** where n is the length of the string. This is because, for each character in the plaintext, you perform a simple character substitution operation based on the key. Since both the plaintext and key have the same length, you perform a constant-time operation for each character in the message.

**Applications:**

* **Educational Purposes:** The Vigenère cipher can be a useful tool for teaching the concept of polyalphabetic substitution and basic encryption techniques. It helps students understand the principles of cryptography, including key management and substitution ciphers.
* **Puzzles and Games:** The Vigenère cipher can be used to create puzzles and games. Cryptogram puzzles, for example, often use simple substitution ciphers like the Vigenère to encode messages that readers need to decipher.
* **Historical Documents Analysis:** When studying historical documents, researchers may encounter encrypted messages that were encrypted using methods like the Vigenère cipher. Deciphering these messages can provide insights into the history and context of the document.
* **Steganography:** Although not a highly secure method, the Vigenère cipher can be used in steganography, the practice of hiding messages within other media. In this context, it's used more for obfuscation than security.

**References:**

* Wikipedia(https://en.wikipedia.org/wiki/Vigen%C3%A8re\_cipher)
* GeeksforGeeks(<https://www.geeksforgeeks.org/vigenere-cipher/>)

**EXPERIMENT NO: 6**

**Aim:** Study and implement a program for 5\*5 Playfair cipher.

**Introduction:** Playfair cipher is an encryption algorithm to encrypt or encode a message. It is the same as a traditional cipher. The only difference is that it encrypts a **digraph** (a pair of two letters) instead of a single letter.

It initially creates a key-table of 5\*5 matrix. The matrix contains alphabets that act as the key for encryption of the plaintext. Note that any alphabet should not be repeated. Another point to note that there are 26 alphabets and we have only 25 blocks to put a letter inside it. Therefore, one letter is excess so, a letter will be omitted (usually J) from the matrix. Nevertheless, the plaintext contains J, then **J** is replaced by **I**. It means treat I and J as the same letter, accordingly.

Algorithm to encrypt the plain text: The plaintext is split into pairs of two letters (digraphs). If there is an odd number of letters, a Z is added to the last letter.

Pair cannot be made with same letter. Break the letter in single and add a bogus letter to the previous letter. If both the letters are in the same column: Take the letter below each one (going back to the top if at the bottom).

If both the letters are in the same row: Take the letter to the right of each one (going back to the leftmost if at the rightmost position).

If neither of the above rules is true: Form a rectangle with the two letters and take the letters on the horizontal opposite corner of the rectangle.

Decrypting the Playfair cipher is as simple as doing the same process in reverse. The receiver has the same key and can create the same key table, and then decrypt any messages made using that key

**Program Code:**

import java.awt.Point;

import java.util.Scanner;

public class PlayfairCipher {

    private int length = 0;

    private String[][] table;

    public static void main(String args[]) {

        new PlayfairCipher().start();

    }

    private void start() {

        Scanner sc = new Scanner(System.in);

        System.out.print("Enter the key for Playfair cipher: ");

        String key = parseString(sc);

        while (key.equals("")) {

            System.out.println("Key cannot be empty. Please enter a valid key.");

        }

        table = this.cipherTable(key);

        System.out.print("Enter the plaintext to be encipher: ");

        String input = parseString(sc);

        while (input.equals("")) {

            System.out.println("Plaintext cannot be empty. Please enter a valid plaintext.");

        }

        String output = cipher(input);

        String decodedOutput = decode(output);

        keyTable(table);

        printResults(output, decodedOutput);

        sc.close();

    }

    private String parseString(Scanner sc) {

        String parse = sc.nextLine();

        parse = parse.toUpperCase();

        parse = parse.replaceAll("[^A-Z]", "");

        parse = parse.replace("J", "I");

        return parse;

    }

    private String[][] cipherTable(String key) {

        String[][] playfairTable = new String[5][5];

        String keyString = key + "ABCDEFGHIKLMNOPQRSTUVWXYZ";

        for (int i = 0; i < 5; i++)

            for (int j = 0; j < 5; j++)

                playfairTable[i][j] = "";

        for (int k = 0; k < keyString.length(); k++) {

            boolean repeat = false;

            boolean used = false;

            for (int i = 0; i < 5; i++) {

                for (int j = 0; j < 5; j++) {

                    if (playfairTable[i][j].equals("" + keyString.charAt(k))) {

                        repeat = true;

                    } else if (playfairTable[i][j].equals("") && !repeat && !used) {

                        playfairTable[i][j] = "" + keyString.charAt(k);

                        used = true;

                    }

                }

            }

        }

        return playfairTable;

    }

    private String cipher(String in) {

        length = (int) in.length() / 2 + in.length() % 2;

        for (int i = 0; i < (length - 1); i++) {

            if (in.charAt(2 \* i) == in.charAt(2 \* i + 1)) {

                in = new StringBuffer(in).insert(2 \* i + 1, 'X').toString();

                length = (int) in.length() / 2 + in.length() % 2;

            }

        }

        String[] digraph = new String[length];

        for (int j = 0; j < length; j++) {

            if (j == (length - 1) && in.length() / 2 == (length - 1))

                in = in + "X";

            digraph[j] = in.charAt(2 \* j) + "" + in.charAt(2 \* j + 1);

        }

        String out = "";

        String[] encDigraphs = new String[length];

        encDigraphs = encodeDigraph(digraph);

        for (int k = 0; k < length; k++)

            out = out + encDigraphs[k];

        return out;

    }

    private String[] encodeDigraph(String di[]) {

        String[] encipher = new String[length];

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

            char a = di[i].charAt(0);

            char b = di[i].charAt(1);

            int r1 = (int) getPoint(a).getX();

            int r2 = (int) getPoint(b).getX();

            int c1 = (int) getPoint(a).getY();

            int c2 = (int) getPoint(b).getY();

            if (r1 == r2) {

                c1 = (c1 + 1) % 5;

                c2 = (c2 + 1) % 5;

            } else if (c1 == c2) {

                r1 = (r1 + 1) % 5;

                r2 = (r2 + 1) % 5;

            } else {

                int temp = c1;

                c1 = c2;

                c2 = temp;

            }

            encipher[i] = table[r1][c1] + "" + table[r2][c2];

        }

        return encipher;

    }

    private String decode(String out) {

        String decoded = "";

        for (int i = 0; i < out.length() / 2; i++) {

            char a = out.charAt(2 \* i);

            char b = out.charAt(2 \* i + 1);

            int r1 = (int) getPoint(a).getX();

            int r2 = (int) getPoint(b).getX();

            int c1 = (int) getPoint(a).getY();

            int c2 = (int) getPoint(b).getY();

            if (r1 == r2) {

                c1 = (c1 + 4) % 5;

                c2 = (c2 + 4) % 5;

            } else if (c1 == c2) {

                r1 = (r1 + 4) % 5;

                r2 = (r2 + 4) % 5;

            } else {

                int temp = c1;

                c1 = c2;

                c2 = temp;

            }

            decoded = decoded + table[r1][c1] + table[r2][c2];

        }

        return decoded;

    }

    private Point getPoint(char c) {

        Point pt = new Point(0, 0);

        for (int i = 0; i < 5; i++)

            for (int j = 0; j < 5; j++)

                if (c == table[i][j].charAt(0))

                    pt = new Point(i, j);

        return pt;

    }

    private void keyTable(String[][] printTable) {

        System.out.println("Playfair Cipher Key Matrix:");

        System.out.println();

        for (int i = 0; i < 5; i++) {

            for (int j = 0; j < 5; j++) {

                System.out.print(printTable[i][j] + " ");

            }

            System.out.println();

        }

        System.out.println();

    }

    private void printResults(String encipher, String dec) {

        System.out.print("Encrypted Message: ");

        System.out.println(encipher);

        System.out.println();

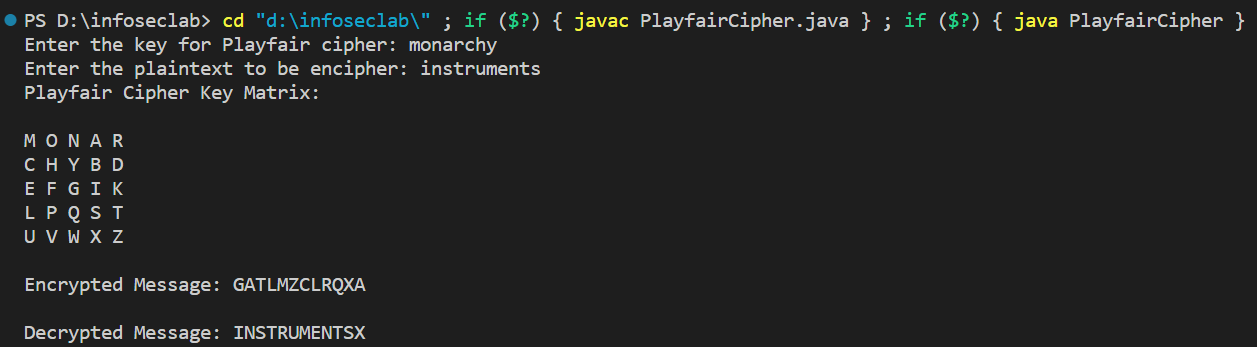
        System.out.print("Decrypted Message: ");

        System.out.println(dec);

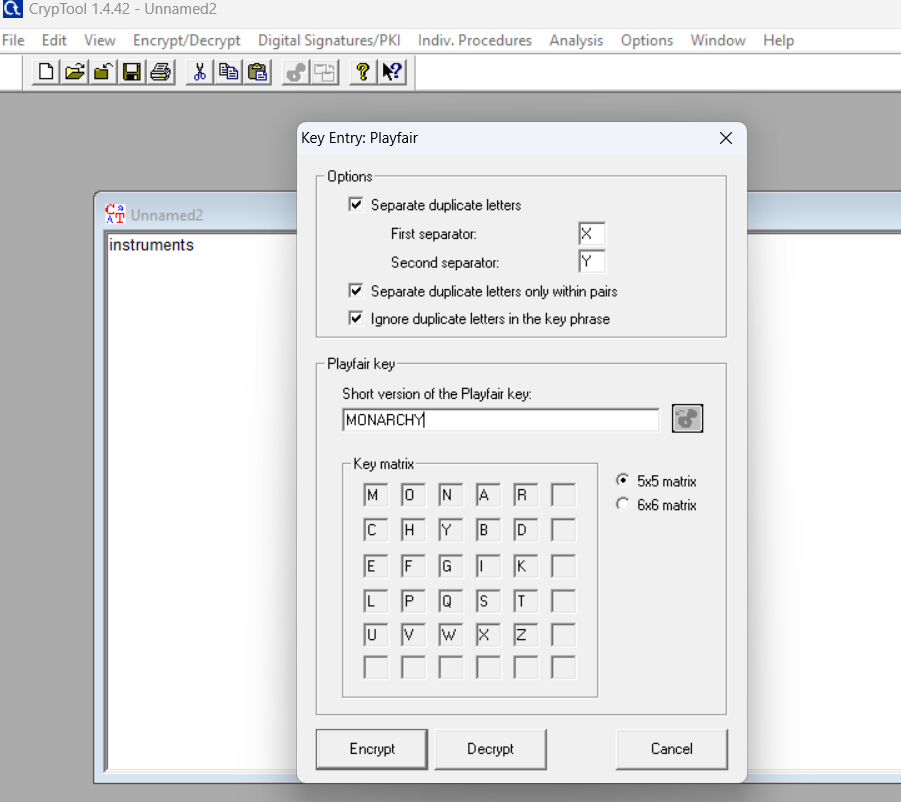
    }

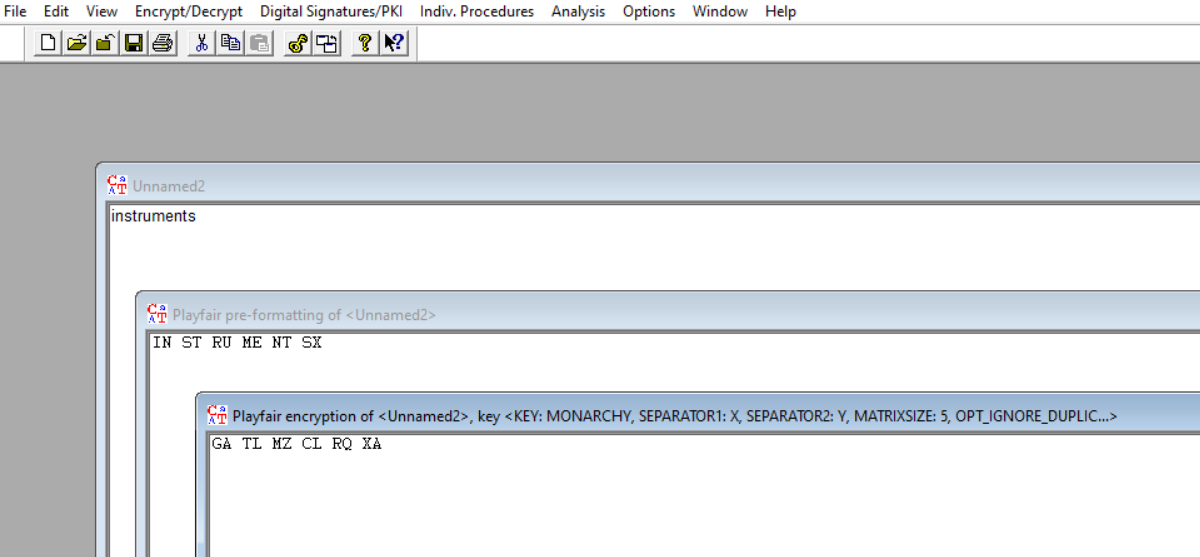
}

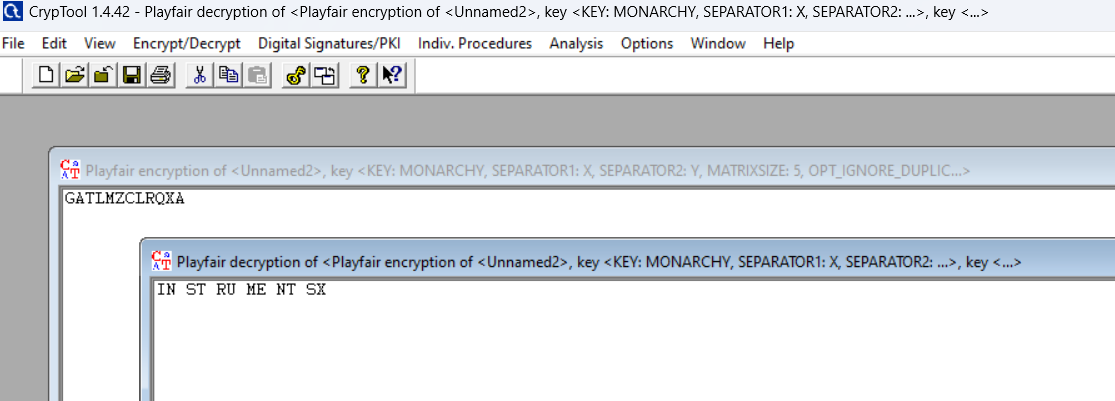
**Program Output(IDE):**

****

**Program Output (Cryptool):**

****

****

****

**Cryptanalysis:**

**Advantages of Playfair Cipher:**

* **Polygram Substitution:** Encrypts pairs of letters together, increasing complexity.
* **Enhanced Security:** Offers more security compared to simple substitution ciphers with moderate complexity.
* **Key Management:** The 5x5 matrix provides a clear visual representation of the key.
* **Suitable for Pen-and-Paper:** Can be implemented manually without special tools.

**Disadvantages of Playfair Cipher:**

* **Limited Character Set:** Cannot encrypt numbers, symbols, or non-alphabetic characters.
* **Lack of Perfect Secrecy:** Doesn't achieve perfect secrecy, making it less secure.
* **Unsuitable for Modern Cryptography:** Not recommended for highly sensitive data.

**Time Complexity:**

**Encryption:** The time complexity for encrypting a message using the Playfair cipher is typically O(n), where n is the length of the message. This is because you need to process each pair of letters in the message, look up their positions in the key matrix, and apply the encryption rules.

**Decryption:** Decryption using the Playfair cipher also has a time complexity of O(n), as you need to process each pair of letters in the ciphertext, find their positions in the key matrix, and apply the decryption rules.

**Applications:**

* Applications of the Playfair cipher include military communications, cryptography puzzles, and cryptanalysis studies.
* It was used for tactical purposes by British forces in the Second Boer War and in World War I and for the same purpose by the Australians during World War II.
* It has also been used in digital encryption systems, such as the CAST-128 and A5/1 encryption algorithms.

**References:**

* Wikipedia()
* GeeksforGeeks(https://www.geeksforgeeks.org/playfair-cipher-with-examples/)

**EXPERIMENT NO: 7**

**Aim:** Study and implement a program for n-gram Hill cipher.

**Introduction:**

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).

Decryption

To decrypt the message, we turn the ciphertext back into a vector, then simply multiply by the inverse matrix of the key matrix.

**Program Code:**

#include <bits/stdc++.h>

using namespace std;

void performEncryption(){

    cout << "Enter the encryption key: ";

    string key;

    cin >> key;

    int n = sqrt(key.length());

    vector<vector<int>> matrix(n, vector<int>(n));

    // filling the key matrix

    int iter = 0;

    for (int i = 0; i < n; i++){

        for (int j = 0; j < n; j++){

            int val = key[iter] - 97;

            matrix[i][j] = val;

            iter++;

        }

    }

    // printing key matrix

    cout << "Key matrix:" << endl;

    for (int i = 0; i < n; i++)

    {

        for (int j = 0; j < n; j++)

        {

            cout << matrix[i][j] << " ";

        }

        cout << endl;

    }

    cout << "Enter the message to encrypt: ";

    string message;

    cin >> message;

    // adding padding in message to fill in extra spaces

    int padding = (n - message.size() % n) % n;

    for (int i = 0; i < padding; i++){

        message += 'x';

    }

    // forming encrypted text

    int k = 0;

    string encryptedText = "";

    while (k < message.size()){

        for (int i = 0; i < n; i++){

            int sum = 0;

            int temp = k;

            for (int j = 0; j < n; j++){

                sum += (matrix[i][j] % 26 \* (message[temp++] - 'a') % 26) % 26;

                sum = sum % 26;

            }

            encryptedText += (sum + 'a');

        }

        k += n;

    }

    cout << "\nEncrypted text is: " << encryptedText << '\n';

}

// function to calculate a mod(m)

int modInverse(int a, int m){

    a = a % m;

    for (int x = -m; x < m; x++)

        if ((a \* x) % m == 1)

            return x;

    return 0;

}

// function that calculates a cofactor matrix 'temp' of matrix 'a' excluding the row 'p' and column 'q'

void getCofactor(vector<vector<int>> &a, vector<vector<int>> &temp, int p, int q, int n){

    int i = 0, j = 0;

    for (int row = 0; row < n; row++){

        for (int col = 0; col < n; col++){

            if (row != p && col != q){

                temp[i][j++] = a[row][col];

                if (j == n - 1){

                    j = 0;

                    i++;

                }

            }

        }

    }

}

// calculates determinant of a matrix 'a'

int calculateDeterminant(vector<vector<int>> &a, int n, int N){

    int D = 0;

    if (n == 1)

        return a[0][0];

    vector<vector<int>> temp(N, vector<int>(N));

    int sign = 1;

    for (int f = 0; f < n; f++){

        getCofactor(a, temp, 0, f, n);

        D += sign \* a[0][f] \* calculateDeterminant(temp, n - 1, N);

        sign = -sign;

    }

    return D;

}

// calculates adjoint matrix 'adj' of a matrix 'a' using recursion

void getAdjoint(vector<vector<int>> &a, vector<vector<int>> &adj, int N){

    if (N == 1){

        adj[0][0] = 1;

        return;

    }

    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++){

            getCofactor(a, temp, i, j, N);

            sign = ((i + j) % 2 == 0) ? 1 : -1;

            adj[j][i] = (sign) \* (calculateDeterminant(temp, N - 1, N));

        }

    }

}

bool calculateInverse(vector<vector<int>> &a, vector<vector<int>> &inv, int N)

{

    int det = calculateDeterminant(a, N, N);

    if (det == 0){

        cout << "Inverse does not exist";

        return false;

    }

    int invDet = modInverse(det, 26);

    cout << det % 26 << " #### " << invDet << '\n';

    vector<vector<int>> adj(N, vector<int>(N));

    getAdjoint(a, adj, N);

    for (int i = 0; i < N; i++)

        for (int j = 0; j < N; j++)

            inv[i][j] = (adj[i][j] \* invDet) % 26;

    return true;

}

void performDecryption(){

    int x, y, i, j, k, n;

    cout << "Enter the decryption key: ";

    string key;

    cin >> key;

    n = sqrt(key.length());

    vector<vector<int>> a(n, vector<int>(n));

    vector<vector<int>> adj(n, vector<int>(n));

    vector<vector<int>> inv(n, vector<int>(n));

    int iter = 0;

    for (i = 0; i < n; i++){

        for (j = 0; j < n; j++){

            int val = key[iter] - 97;

            a[i][j] = val;

            iter++;

        }

    }

    if (calculateInverse(a, inv, n)){

        cout << "Inverse exists\n";

    }

    cout << "Enter the message to decrypt\n";

    string message;

    cin >> message;

    k = 0;

    string decryptedText = "";

    while (k < message.size()){

        for (i = 0; i < n; i++){

            int sum = 0;

            int temp = k;

            for (j = 0; j < n; j++){

                sum += ((inv[i][j] + 26) % 26 \* (message[temp++] - 'a') % 26) % 26;

                sum = sum % 26;

            }

            sum += 'a';

            decryptedText += sum;

        }

        k += n;

    }

    int lastCharIndex = decryptedText.size() - 1;

    while (decryptedText[lastCharIndex] == 'x'){

        lastCharIndex--;

    }

    cout << "\nDecrypted text is: ";

    for (i = 0; i <= lastCharIndex; i++){

        cout << decryptedText[i];

    }

    cout << '\n';

}

int main(){

    int choice;

    cout << "Enter your choice :\n";

    cout << "1. Encryption:\n2.Decryption:\n";

    cin >> choice;

    switch (choice){

    case 1:

        performEncryption();

        break;

    case 2:

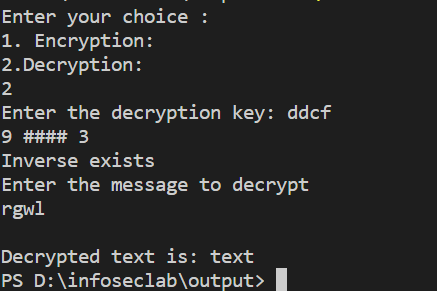
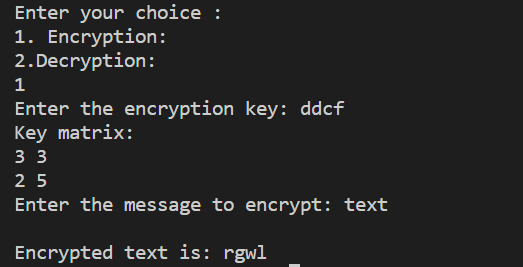
        performDecryption();

        break;

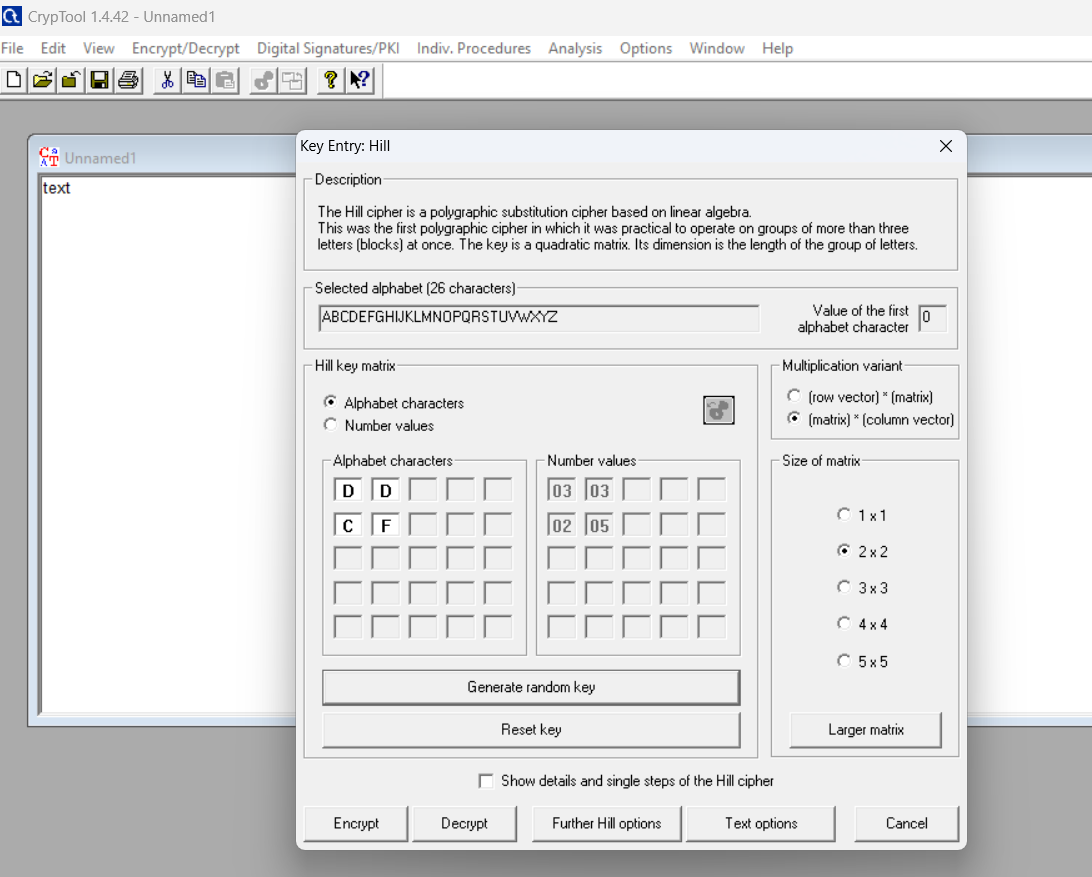
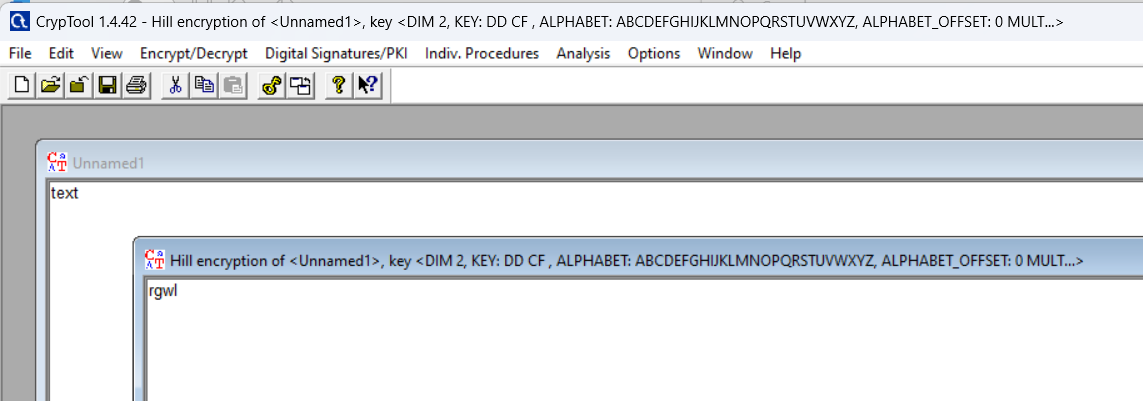
    }

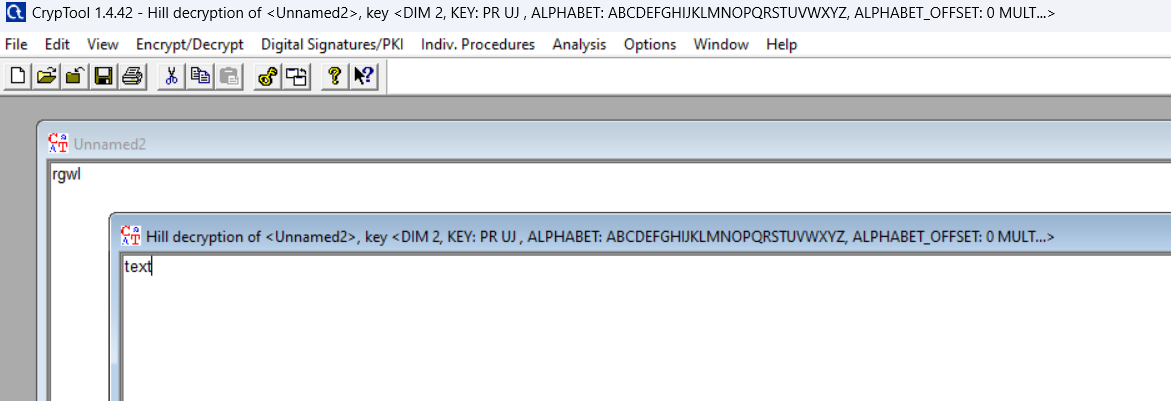
}

**Program Output(IDE):**

****

**Program Output (Cryptool):**

**** ****

****

**Cryptanalysis:**

**Key length:** An important factor in the security of a Hill cipher is the size of its key matrix. In a Hill Cipher with a key matrix of size n x n, with a dimension of dimension n, the key space grows exponentially with n. Consequently, larger key matrices are generally more secure, but they also increase computational complexity.

**Known plaintext attack:** If the attacker has access to the plaintext pairs and associated ciphertext, a known plaintext attack can be used to recover the key. The complexity of this attack depends on the size of the key matrix.

**Time Complexity:**

The time complexity in Hill cipher encryption and decryption is O(n^3), where n is the dimension of the key matrix.

**Applications:**

* **Image Encryption:**While not as common as other encryption techniques, Hill cipher can be adapted for encrypting images. Each pixel's color values can be treated as numerical values and encrypted using the Hill cipher.
* **Securing Text:**The primary application of the Hill Cipher is in securing text messages. It uses a matrix as a key to encrypt and decrypt messages, making it a useful tool for ensuring the confidentiality of sensitive information transmitted over open networks.
* **Multimedia security:** Hill Cipher is employed in multimedia security as an inexpensive and robust tool. It is particularly useful in protecting multimedia content from unauthorized access or tampering due to its simplicity and efficiency.

**References:**

* CodingNinjas (https://www.codingninjas.com/studio/library/polyalphabetic-hill-cipher)
* GeeksforGeeks(https://www.geeksforgeeks.org/hill-cipher/)