Name	Atharva Tamhankar
UID no.	2022300127
Experiment No.	1

AIM:	Understanding and implementing hill cipher, caesar cipher and playfair cipher.
DISCUSSION:	# Overview of Implementation This report presents the implementation of three classical cryptographic algorithms: Caesar Cipher, Playfair Cipher, and Hill Cipher. Each algorithm represents different approaches to symmetric encryption with varying levels of complexity and security strength. # Algorithm Analysis ### Caesar Cipher The Caesar cipher implementation demonstrates the simplest form of substitution cipher. The program successfully handles both encryption and decryption operations across all possible shift values (0-25). Key observations include: - **Simplicity**: The algorithm is straightforward to implement and understand - **Brute Force Vulnerability**: With only 26 possible keys, the cipher is easily broken through exhaustive search - **Character Handling**: The implementation properly preserves non-alphabetic characters while converting all letters to lowercase - **Practical Application**: While cryptographically weak, it serves as an excellent educational tool for understanding basic encryption concepts ### Playfair Cipher The Playfair cipher represents a significant improvement over simple

	substitution ciphers by encrypting digrams (pairs of letters) rather than individual characters. Implementation highlights: ● **Key Matrix Generation**: The program correctly constructs a 5×5 matrix from the provided key, handling duplicate characters and the traditional i/j equivalence ● **Text Preprocessing**: Proper handling of odd-length plaintexts by padding with 'x' character ● **Encryption Rules**: Accurate implementation of the three Playfair rules (same row, same column, rectangle) ● **Security Enhancement**: The digram approach significantly increases cryptographic strength compared to monoalphabetic ciphers ● **Limitations**: Still vulnerable to frequency analysis of digrams and known plaintext attacks **Hill Cipher** The Hill cipher implementation showcases matrix-based cryptography, representing the most mathematically sophisticated approach among the three algorithms: ● **Matrix Operations**: Successful implementation of 3×3 matrix multiplication for encryption ● **Input Constraints**: The program enforces the requirement for exactly 3-character plaintext and 9-character keys ● **Mathematical Foundation**: Demonstrates linear algebra applications in cryptography ● **Security Considerations**: While more secure than previous methods, it remains vulnerable to known plaintext attacks due to its linear nature
CODE:	#include <cctype> #include <iostream> #include <string> #include <vector> using namespace std;

```
// --- Caesar Cipher Implementation ---
string caesarEncrypt(string plaintext, int shift) {
    string ciphertext = "";
    for (char character : plaintext) {
        char ch = tolower(character);
        if (isalpha(ch)) {
            int shifted = (ch - 'a' + shift) % 26;
            ciphertext += char('a' + shifted);
        } else {
            ciphertext += ch;
        }
    }
    return ciphertext;
}

string caesarDecrypt(string ciphertext, int shift) {
    string plaintext = "";
    for (char character : ciphertext) {
        char ch = tolower(character);
        if (isalpha(ch)) {
            int shifted = (ch - 'a' - shift + 26) % 26;
            plaintext += char('a' + shifted);
        } else {
            plaintext += ch;
        }
    }
    return plaintext;
}

// --- Playfair Cipher Implementation ---
string convertToLowerCase(string input) {
    int length = input.length();
    string output = "";
    for (int i = 0; i < length; i++) {
        if (input[i] >= 'A' && input[i] <= 'Z') {
            output += char(input[i] + 32);
        } else {
            output += input[i];
        }
```

```
    }
    return output;
}

string eliminateSpaces(string input) {
    int length = input.length();
    string result = "";
    for (int i = 0; i < length; i++) {
        if (input[i] != ' ') {
            result += input[i];
        }
    }
    return result;
}

void createKeyMatrix(string key, vector<vector<char>> &matrix) {
    int keyLength = key.length();
    matrix.clear();
    matrix.resize(5, vector<char>(5));
    vector<int> charMap(26, 0);

    // Mark key characters as used
    for (int i = 0; i < keyLength; i++) {
        if (key[i] != 'j') {
            charMap[key[i] - 'a'] = 2;
        }
    }
    charMap['j' - 'a'] = 1; // 'j' is treated as 'i'

    int row = 0, col = 0;
    // Fill matrix with key characters first
    for (int k = 0; k < keyLength; k++) {
        if (charMap[key[k] - 'a'] == 2) {
            charMap[key[k] - 'a']--;
            matrix[row][col] = key[k];
            col++;
            if (col == 5) {
                row++;
                col = 0;
            }
```

```
        }
    }

    // Fill remaining positions with unused letters
    for (int k = 0; k < 26; k++) {
        if (charMap[k] == 0) {
            matrix[row][col] = char(k + 'a');
            col++;
            if (col == 5) {
                row++;
                col = 0;
            }
        }
    }

    // Display the Playfair matrix
    cout << "Playfair Key Matrix:" << endl;
    for (int r = 0; r < 5; r++) {
        for (int c = 0; c < 5; c++) {
            cout << matrix[r][c] << " ";
        }
        cout << endl;
    }
}

void findPositions(vector<vector<char>> &matrix, char first, char second,
vector<int> &positions) {
    if (first == 'j') first = 'i';
    if (second == 'j') second = 'i';

    for (int i = 0; i < 5; i++) {
        for (int j = 0; j < 5; j++) {
            if (matrix[i][j] == first) {
                positions[0] = i;
                positions[1] = j;
            } else if (matrix[i][j] == second) {
                positions[2] = i;
                positions[3] = j;
            }
        }
```

```
    }
}

string prepareText(string text) {
    if (text.length() % 2 != 0) {
        text += 'x'; // Changed from 'z' to 'x'
    }
    return text;
}

string playfairEncryption(string text, vector<vector<char>> &matrix) {
    int textLength = text.length();
    vector<int> positions(4);
    string encrypted = text;

    for (int i = 0; i < textLength; i += 2) {
        findPositions(matrix, encrypted[i], encrypted[i + 1], positions);

        if (positions[0] == positions[2]) { // Same row
            encrypted[i] = matrix[positions[0]][(positions[1] + 1) % 5];
            encrypted[i + 1] = matrix[positions[0]][(positions[3] + 1) % 5];
        } else if (positions[1] == positions[3]) { // Same column
            encrypted[i] = matrix[(positions[0] + 1) % 5][positions[1]];
            encrypted[i + 1] = matrix[(positions[2] + 1) % 5][positions[1]];
        } else { // Rectangle
            encrypted[i] = matrix[positions[0]][positions[3]];
            encrypted[i + 1] = matrix[positions[2]][positions[1]];
        }
    }
    return encrypted;
}

string playfairCipherEncrypt(string plaintext, string key) {
    vector<vector<char>> keyMatrix;
    key = convertToLowerCase(eliminateSpaces(key));
    plaintext = convertToLowerCase(eliminateSpaces(plaintext));
    plaintext = prepareText(plaintext);
    createKeyMatrix(key, keyMatrix);
    return playfairEncryption(plaintext, keyMatrix);
}
```

```
// --- Hill Cipher Implementation ---
string hillCipherEncrypt(string plaintext, string key) {
    if (plaintext.length() != 3 || key.length() != 9) {
        return "Error: Hill cipher requires exactly 3 characters for plaintext and
9 characters for key.";
    }

    // Create key matrix (3x3)
    vector<vector<int>> keyMatrix(3, vector<int>(3));
    int index = 0;
    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 3; j++) {
            keyMatrix[i][j] = toupper(key[index]) - 'A';
            index++;
        }
    }

    // Create message vector
    vector<vector<int>> messageVector(3, vector<int>(1));
    for (int i = 0; i < 3; i++) {
        messageVector[i][0] = toupper(plaintext[i]) - 'A';
    }

    // Perform matrix multiplication
    vector<vector<int>> resultMatrix(3, vector<int>(1));
    for (int i = 0; i < 3; i++) {
        resultMatrix[i][0] = 0;
        for (int j = 0; j < 3; j++) {
            resultMatrix[i][0] += keyMatrix[i][j] * messageVector[j][0];
        }
        resultMatrix[i][0] = resultMatrix[i][0] % 26;
    }

    // Convert result to string
    string encryptedText = "";
    for (int i = 0; i < 3; i++) {
        encryptedText += char(resultMatrix[i][0] + 'A');
    }
    return encryptedText;
```

```
}

// --- Main Function ---
int main() {
    cout << "=== Cryptography Program ===" << endl;
    cout << "Available cipher methods:" << endl;
    cout << "1. Caesar Cipher" << endl;
    cout << "2. Playfair Cipher" << endl;
    cout << "3. Hill Cipher" << endl;

    int option;
    cout << "Please select your choice (1, 2, or 3): ";
    cin >> option;
    cin.ignore(); // Clear input buffer

    switch (option) {
        case 1: {
            string inputText;
            cout << "Enter text for Caesar cipher: ";
            getline(cin, inputText);

            cout << "\n=== CAESAR ENCRYPTION (shift right) ===" <<
endl;
            for (int shift = 0; shift < 26; shift++) {
                string encrypted = caesarEncrypt(inputText, shift);
                cout << "Shift " << shift << ": " << encrypted << endl;
            }

            cout << "\n=== CAESAR DECRYPTION (shift left) ===" << endl;
            for (int shift = 0; shift < 26; shift++) {
                string decrypted = caesarDecrypt(inputText, shift);
                cout << "Shift " << shift << ": " << decrypted << endl;
            }
            break;
        }
        case 2: {
            string cipherKey, plainText;
            cout << "Enter key for Playfair cipher: ";
            getline(cin, cipherKey);
            cout << "Enter plaintext: ";
```

```
            getline(cin, plainText);

            string encryptedText = playfairCipherEncrypt(plainText,
cipherKey);
            cout << "Encrypted text: " << encryptedText << endl;
            break;
        }
        case 3: {
            string message, cipherKey;
            cout << "Enter 3-character plaintext: ";
            getline(cin, message);
            cout << "Enter 9-character key: ";
            getline(cin, cipherKey);

            string encryptedText = hillCipherEncrypt(message, cipherKey);
            cout << "Encrypted text: " << encryptedText << endl;
            break;
        }
        default:
            cout << "Error: Invalid selection. Please choose 1, 2, or 3." << endl;
            break;
    }

    return 0;
}
```

RESULT:

```
=== Cryptography Program ===
Available cipher methods:
1. Caesar Cipher
2. Playfair Cipher
3. Hill Cipher
Please select your choice (1, 2, or 3): 3
Enter 3-character plaintext: tus
Enter 9-character key: rajatusha
Encrypted text: RMO
```

```
=== Cryptography Program ===
Available cipher methods:
1. Caesar Cipher
2. Playfair Cipher
3. Hill Cipher
Please select your choice (1, 2, or 3): 3
Enter 3-character plaintext: att
Enter 9-character key: johndoeoe
Encrypted text: JLE
```

```
=== Cryptography Program ===
Available cipher methods:
1. Caesar Cipher
2. Playfair Cipher
3. Hill Cipher
Please select your choice (1, 2, or 3): 2
Enter key for Playfair cipher: man
Enter plaintext: woman
Playfair Key Matrix:
m a n b c
d e f g h
i k l o p
q r s t u
v w x y z
Encrypted text: ykanfn
```

```
=== Cryptography Program ===
Available cipher methods:
1. Caesar Cipher
2. Playfair Cipher
3. Hill Cipher
Please select your choice (1, 2, or 3): 2
Enter key for Playfair cipher: rohit
Enter plaintext: shubham
Playfair Key Matrix:
r o h i t
a b c d e
f g k l m
n p q s u
v w x y z
Encrypted text: qiperckz
```

```
=== Cryptography Program ===
Available cipher methods:
1. Caesar Cipher
2. Playfair Cipher
3. Hill Cipher
Please select your choice (1, 2, or 3): 1
Enter text for Caesar cipher: visa

=== CAESAR ENCRYPTION (shift right) ===
Shift 0: visa
Shift 1: wjtb
Shift 2: xkuc
Shift 3: ylvd
Shift 4: zmwe
Shift 5: anxf
Shift 6: boyg
Shift 7: cpzh
Shift 8: dqai
Shift 9: erbj
Shift 10: fsck
Shift 11: gtdl
Shift 12: huem
Shift 13: ivfn
Shift 14: jwgo
Shift 15: kxhp
Shift 16: lyiq
Shift 17: mzjr
Shift 18: naks
```

```
Shift 19: oblt
Shift 20: pcmu
Shift 21: qdnv
Shift 22: reow
Shift 23: sfpx
Shift 24: tgqy
Shift 25: uhrz

=== CAESAR DECRYPTION (shift left) ===
Shift 0: visa
Shift 1: uhrz
Shift 2: tgqy
Shift 3: sfpx
Shift 4: reow
Shift 5: qdnv
Shift 6: pcmu
Shift 7: oblt
Shift 8: naks
Shift 9: mzjr
Shift 10: lyiq
Shift 11: kxhp
Shift 12: jwgo
Shift 13: ivfn
Shift 14: huem
Shift 15: gtdl
Shift 16: fsck
Shift 17: erbj
Shift 18: dqai
```

CONCLUSION:	The successful implementation of these three classical ciphers provides valuable insights into fundamental cryptographic principles. While these algorithms are no longer suitable for practical security applications, they serve as excellent stepping stones for understanding modern cryptographic concepts. The modular program design facilitates easy extension and modification, making it an effective educational tool for cryptography students. The project demonstrates the evolution from simple character substitution to more complex mathematical transformations, illustrating how cryptographic strength has developed over time. This foundation prepares students for advanced topics in modern cryptography, including symmetric and asymmetric encryption algorithms used in contemporary security systems.