| **Course Name:** | **Information Security (116U01L602)** | **Semester:** | **VI** |
| --- | --- | --- | --- |
| **Date of Performance:** | **16 / 01 / 2025** | **DIV/ Batch No:** | **C - 3** |
| **Student Name:** | **Romil Lodaya** | **Roll No:** | **16010122096** |

| **Title: Encryption-Decryption programs using classical cryptography**  **(Caeser Cipher, Playfair cipher, Transposition cipher)** |
| --- |

| **Objectives:** |
| --- |
| To write a program to convert plain text into cipher text using Caesar cipher, Playfair Cipher and Transposition cipher |

| **Expected Outcome of Experiment:** |
| --- |
| **CO1: Explain various security goals, threats, vulnerabilities and controls**  **CO2: Apply various cryptographic algorithms for software security** |

| **Books/ Journals/ Websites referred:** |
| --- |
| GeeksforGeeks  TutorialsPoint |

| **Pre Lab/ Prior Concepts:** |
| --- |
| * **Classical Cryptography**: This refers to the traditional methods used for encryption and decryption of text before the advent of modern algorithms. Classical ciphers typically use basic mathematical transformations to encrypt plaintext into ciphertext and vice versa. * **Caesar Cipher**: A simple substitution cipher where each letter of the plaintext is shifted by a certain number of positions down or up the alphabet. The key for the Caesar cipher is the number of positions each letter is shifted. * **Transposition Cipher**: This is a type of cipher where the positions of the characters in the plaintext are shifted according to a specific system or algorithm. Unlike substitution ciphers (like Caesar Cipher), which replace characters, transposition ciphers rearrange the order of characters. * **Playfair Cipher**: The Playfair cipher is a digraph cipher, meaning it encrypts pairs of letters at a time. It uses a 5x5 grid of letters, typically constructed using a keyword. The plaintext is divided into pairs of letters, and each pair is encrypted based on their positions in the grid. If both letters in a pair are the same, or only one letter remains at the end, it is adjusted (by inserting a filler letter, typically 'X'). * **Security Goals**: Cryptography is implemented to achieve certain goals in a security context. These include: * **Confidentiality**: Ensuring that data is only accessible by authorized entities. * **Integrity**: Ensuring that data is not altered or tampered with. * **Authentication**: Verifying the identity of the sender or receiver of data. * **Non-repudiation**: Ensuring that the sender cannot deny the authenticity of the data sent. * **Threats, Vulnerabilities, and Controls**: * **Threats**: Potential risks or attacks that could compromise the confidentiality, integrity, or availability of information. * **Vulnerabilities**: Weaknesses in a system that can be exploited by threats. * **Controls**: Mechanisms or strategies to mitigate or prevent risks or threats to system security. |

| **New Concepts to be learned:** |
| --- |
| * **Caesar Cipher**: A substitution cipher where each letter is shifted by a fixed number. You'll implement encryption and decryption of text using this cipher and explore its strengths and weaknesses. * **Transposition Cipher**: A cipher that rearranges the positions of characters in a message based on a particular key or algorithm. You will implement both encryption and decryption methods and study the effect of key lengths and arrangement on security. * **Playfair Cipher**: This cipher encrypts plaintext in pairs, based on a 5x5 grid constructed using a keyword. You will understand how the cipher handles pairs of letters, what adjustments are made for repeated letters, and how to use a keyword to construct the grid. This cipher provides a higher level of security compared to simpler ciphers like Caesar and Transposition ciphers. * **Security and Cryptographic Algorithms**: You'll gain insights into how classical ciphers, like Caesar, Transposition, and Playfair ciphers, are used to protect data and understand their place in software security. You'll also learn the difference between these traditional methods and more modern encryption techniques like AES. * **Key Management**: You'll understand the importance of the key in all these ciphers and how securely handling the key is essential for maintaining the confidentiality and integrity of the data. |

| **Abstract:** |
| --- |
| This experiment explores classical cryptographic algorithms, specifically the Caesar cipher, Transposition cipher, and Playfair cipher. These algorithms are implemented to convert plaintext into ciphertext and vice versa. The Caesar cipher uses a simple substitution method by shifting the alphabet, while the Transposition cipher rearranges the characters in a specified order. The Playfair cipher encrypts pairs of letters using a 5x5 grid based on a keyword. By using these ciphers, the experiment demonstrates basic cryptographic techniques and their applications in software security. |

| **Related Theory:** |
| --- |
| **Caesar Cipher:**   * The Caesar cipher is one of the simplest and oldest encryption techniques. * Each letter in the plaintext is replaced by a letter a fixed number of positions down or up the alphabet. * It is a symmetric key algorithm, meaning the same key is used for both encryption and decryption. * Strengths: Simple to implement. * Weaknesses: Vulnerable to frequency analysis and brute-force attacks, especially for short keys.   **Transposition Cipher:**   * The Transposition cipher does not alter the characters in the message but changes their positions. * It operates on the principle of shuffling the positions of characters based on a key, which makes it harder to decrypt without the key. * Common methods include the Rail Fence cipher and Columnar Transposition cipher. * Strengths: More complex than simple substitution ciphers. * Weaknesses: Vulnerable to attacks like brute-force and known-plaintext attacks if not properly implemented.   **Playfair Cipher:**   * The Playfair cipher is a digraph cipher, meaning it encrypts pairs of letters instead of individual letters. * A 5x5 matrix of letters is created using a keyword. Letters of the keyword are placed in the grid, filling in the remaining spaces with the rest of the alphabet, omitting duplicates. * The plaintext is divided into pairs of letters, and these pairs are encrypted based on their positions in the grid. * If a pair of letters is the same, or if there is an odd number of letters, a filler letter (usually 'X') is added to make pairs. * The encryption process follows specific rules for pairs of letters in the grid: if the letters are in the same row, replace them with the letters to their immediate right; if in the same column, replace them with the letters directly below them; if they form a rectangle, swap the corners. * Strengths: Offers better security than simpler ciphers by handling pairs of letters. * Weaknesses: Still vulnerable to frequency analysis and can be cracked with modern techniques, though more secure than the Caesar cipher.   **Cryptographic Security:**   * Classical ciphers like Caesar, Transposition, and Playfair are important to understand because they form the foundation of modern cryptographic algorithms. * They also help demonstrate how easily encrypted information can be compromised with basic cryptanalysis techniques. * Modern cryptographic algorithms (such as AES, RSA) have advanced security mechanisms, but classical ciphers are still useful for educational purposes to understand the basic concepts of encryption, decryption, and key management. |

| **Implementation Details:** |
| --- |
| **1] Caesar Cipher:**  *#include*<bits/stdc++.h>  using namespace std;  int main()  {      cout << "Enter the mode, Encrypt - 0, Decrypt - 1: ";      int mode;      cin >> mode;  *if*(mode == 0)      {          cout << "Enter the Plain Text: ";          string plain;          cin >> plain;          cout << "Encryption Results:" << endl;  *for*(int key = 1; key <= 25; ++key)          {              cout << "Key " << key << ": ";  *for*(auto &c : plain)              {                  int index = c - 'a';                  index = (index + key) % 26;                  cout << (char)('a' + index);              }              cout << endl;          }      }  *else* *if*(mode == 1){          cout << "Enter the Cipher Text: ";          string cipher;          cin >> cipher;          cout << "Decryption Results:" << endl;  *for*(int key = 1; key <= 25; ++key)          {              cout << "Key " << key << ": ";  *for*(auto &c : cipher)              {                  int index = c - 'a';                  index = (index - key);  *if*(index < 0) index += 26;                  cout << (char)('a' + index);              }              cout << endl;          }      }  *else*      {          cout << "Invalid mode. Please enter 0 for Encrypt or 1 for Decrypt." << endl;      }  *return* 0;  }  **2A] Play Fair (Without key):**  *#include* <bits/stdc++.h>  using namespace std;  void initMatrix(vector<vector<char>> &*matrix*)  {     char c = 'a';     cout << "Playfair Cipher Matrix:" << endl;  *for* (int i = 0; i < 5; i++)     {  *for* (int j = 0; j < 5; j++)        {  *if* (c == 'j')              c++;  *matrix*[i][j] = c;           c++;           cout << *matrix*[i][j] << " ";        }        cout << endl;     }  }  string preprocessPlainText(string *plain*, const string &*padding*)  {     cout << "Preprocessing Plain Text: " << *plain* << endl;  *for* (int i = 0; i < *plain*.size(); i++)     {  *if* (*plain*[i] == 'j')        {  *plain*[i] = 'i';        }     }  *for* (int i = 0; i < *plain*.size() - 1; i++)     {  *if* (*plain*[i] == *plain*[i + 1])        {  *plain*.insert(i + 1, *padding*);           i++;        }     }  *if* (*plain*.size() % 2 == 1)     {  *plain* += *padding*;     }     cout << "Processed Plain Text: " << *plain* << endl;  *return* *plain*;  }  vector<pair<char, char>> divideText(const string &*plain*)  {     vector<pair<char, char>> text;  *for* (int i = 0; i < *plain*.size(); i += 2)     {        text.push\_back({*plain*[i], *plain*[i + 1]});     }  *return* text;  }  string processText(const vector<pair<char, char>> &*text*, const vector<vector<char>> &*matrix*, bool *encrypt*)  {     string resultText = "";     int r1, r2, c1, c2;  *for* (const auto &pair : *text*)     {  *for* (int a = 0; a < 5; ++a)        {  *for* (int b = 0; b < 5; ++b)           {  *if* (*matrix*[a][b] == pair.first)              {                 r1 = a;                 c1 = b;              }  *if* (*matrix*[a][b] == pair.second)              {                 r2 = a;                 c2 = b;              }           }        }  *if* (r1 == r2)        {           resultText += *matrix*[r1][(c1 + (*encrypt* ? 1 : 4)) % 5];           resultText += *matrix*[r2][(c2 + (*encrypt* ? 1 : 4)) % 5];        }  *else* *if* (c1 == c2)        {           resultText += *matrix*[(r1 + (*encrypt* ? 1 : 4)) % 5][c1];           resultText += *matrix*[(r2 + (*encrypt* ? 1 : 4)) % 5][c2];        }  *else*        {           resultText += *matrix*[r1][c2];           resultText += *matrix*[r2][c1];        }     }  *return* resultText;  }  string removePadding(string &*plainText*, const string &*padding*)  {     cout << "Decrypted Text Before Padding Removal: " << *plainText* << endl;  *if* (!*plainText*.empty() && *plainText*.back() == *padding*.back())     {  *plainText*.erase(*plainText*.size() - 1);     }     cout << "Decrypted Text After Padding Removal: " << *plainText* << endl;  *return* *plainText*;  }  int main()  {     vector<vector<char>> matrix(5, vector<char>(5));     string padding = "x";     initMatrix(matrix);     cout << endl          << "Enter the plainText: ";     string plain;     cin >> plain;     plain = preprocessPlainText(plain, padding);     cout << endl          << "Plain Text: " << plain << endl;     vector<pair<char, char>> text = divideText(plain);     string cipherText = processText(text, matrix, true);     cout << endl          << "Cipher Text: " << cipherText << endl << endl;     cout << "Decryption" << endl << endl;     vector<pair<char, char>> cipher = divideText(cipherText);     string decryptedText = processText(cipher, matrix, false);     decryptedText = removePadding(decryptedText, padding);     cout << "Decrypted Text: " << decryptedText << endl;  *return* 0;  }  **2B] Play Fair (With key):**  *#include* <bits/stdc++.h>  using namespace std;  void initMatrix(vector<vector<char>> &*matrix*, string &*key*)  {     set<char> usedChars;     string uniqueChars;  *for* (char c : *key*)     {  *if* (c == 'j')           c = 'i';  *if* (usedChars.find(c) == usedChars.end() && c >= 'a' && c <= 'z')        {           usedChars.insert(c);           uniqueChars += c;        }     }  *for* (char c = 'a'; c <= 'z'; ++c)     {  *if* (c == 'j')  *continue*;  *if* (usedChars.find(c) == usedChars.end())        {           usedChars.insert(c);           uniqueChars += c;        }     }     int index = 0;     cout << "Generated Playfair Cipher Matrix:" << endl;  *for* (int i = 0; i < 5; i++)     {  *for* (int j = 0; j < 5; j++)        {  *matrix*[i][j] = uniqueChars[index++];           cout << *matrix*[i][j] << " ";        }        cout << endl;     }  }  string preprocessPlainText(string *plain*, const string &*padding*)  {     cout << endl          << "Preprocessing Plain Text: " << *plain* << endl;  *for* (int i = 0; i < *plain*.size(); i++)     {  *if* (*plain*[i] == 'j')        {  *plain*[i] = 'i';        }     }  *for* (int i = 0; i < *plain*.size() - 1; i++)     {  *if* (*plain*[i] == *plain*[i + 1])        {  *plain*.insert(i + 1, *padding*);           i++;        }     }  *if* (*plain*.size() % 2 == 1)     {  *plain* += *padding*;     }     cout << endl          << "Processed Plain Text: " << *plain* << endl;  *return* *plain*;  }  vector<pair<char, char>> divideText(const string &*plain*)  {     vector<pair<char, char>> text;     cout << endl          << "Divided Text into Pairs:" << endl;  *for* (int i = 0; i < *plain*.size(); i += 2)     {        text.push\_back({*plain*[i], *plain*[i + 1]});        cout << "(" << *plain*[i] << ", " << *plain*[i + 1] << ") ";     }     cout << endl;  *return* text;  }  string processText(const vector<pair<char, char>> &*text*, const vector<vector<char>> &*matrix*, bool *encrypt*)  {     string resultText = "";     int r1, r2, c1, c2;  *for* (const auto &pair : *text*)     {  *for* (int a = 0; a < 5; ++a)        {  *for* (int b = 0; b < 5; ++b)           {  *if* (*matrix*[a][b] == pair.first)              {                 r1 = a;                 c1 = b;              }  *if* (*matrix*[a][b] == pair.second)              {                 r2 = a;                 c2 = b;              }           }        }  *if* (r1 == r2)        {           resultText += *matrix*[r1][(c1 + (*encrypt* ? 1 : 4)) % 5];           resultText += *matrix*[r2][(c2 + (*encrypt* ? 1 : 4)) % 5];        }  *else* *if* (c1 == c2)        {           resultText += *matrix*[(r1 + (*encrypt* ? 1 : 4)) % 5][c1];           resultText += *matrix*[(r2 + (*encrypt* ? 1 : 4)) % 5][c2];        }  *else*        {           resultText += *matrix*[r1][c2];           resultText += *matrix*[r2][c1];        }     }     cout << endl          << (*encrypt* ? "Cipher Text after Encryption: " : "Text after Decryption: ") << resultText << endl;  *return* resultText;  }  string removePadding(const string &*plainText*, const string &*padding*)  {     string result = *plainText*;  *if* (!result.empty() && result.back() == *padding*.back())     {        result.pop\_back();     }  *return* result;  }  int main()  {     vector<vector<char>> matrix(5, vector<char>(5));     string padding = "x";     string key = "security";     initMatrix(matrix, key);     cout << endl          << "Enter the plainText: ";     string plain;     cin >> plain;     plain = preprocessPlainText(plain, padding);     vector<pair<char, char>> text = divideText(plain);     string cipherText = processText(text, matrix, true);     cout << endl          << "Decryption Process Begins..." << endl;     vector<pair<char, char>> cipher = divideText(cipherText);     string decryptedText = processText(cipher, matrix, false);     decryptedText = removePadding(decryptedText, padding);     cout << endl          << "Final Decrypted Text: " << decryptedText << endl;  *return* 0;  }  **3A] Transposition without key:**  *#include* <bits/stdc++.h>  using namespace std;  *// Function to create the Columnar Matrix for encryption*  vector<vector<char>> initMatrix(const string &*plain*, int *n*)  {     int p = *plain*.size();     vector<vector<char>> v(*n*, vector<char>(*n*, '\_'));  *for* (int i = 0, j = 0, k = 0; k < p; ++j)     {  *if* (j == *n*)        {           j = 0;           i++;        }        v[i][j] = *plain*[k++];     }  *return* v;  }  string encryptedText(vector<vector<char>> &*matrix*, int *n*)  {     string cipherText = "";  *for* (int i = 0; i < *n*; ++i)     {  *for* (int j = 0; j < *n*; ++j)        {           cipherText += *matrix*[j][i];        }     }  *return* cipherText;  }  vector<vector<char>> cipherTextToDmatrix(string &*cipherText*, int *n*)  {     int c = *cipherText*.size();     vector<vector<char>> v(5, vector<char>(5, '\_'));  *for* (int i = 0, k = 0; i < *n*; ++i)     {  *for* (int j = 0; j < *n* && k < *cipherText*.size(); ++j)        {           v[j][i] = *cipherText*[k++];        }     }  *return* v;  }  string decryptionText(vector<vector<char>> &*decryptionMatrix*)  {     string decryptedText = "";  *for* (auto &i : *decryptionMatrix*)     {  *for* (auto j : i)        {  *if* (j != '\_')              decryptedText += j;        }     }  *return* decryptedText;  }  int main()  {     string plain;     cout << "Enter the plain text: ";     cin >> plain;     int n = 5;  *// Encryption*     vector<vector<char>> matrix = initMatrix(plain, n);     cout << endl          << "Encryption:" << endl          << endl          << "Plain text to encryption matrix: " << endl;  *for* (auto &i : matrix)     {  *for* (auto &j : i)        {           cout << j << " ";        }        cout << endl;     }     string cipherText = encryptedText(matrix, n);     cout << endl          << "Cipher Text: " << cipherText << endl;     cout << endl          << "-----------------------------------------------------" << endl;  *// Decryption*     vector<vector<char>> decryptionMatrix = cipherTextToDmatrix(cipherText, n);     cout << endl          << "Decryption:" << endl;     cout << endl          << "Cipher text to decryption matrix: " << endl;  *for* (auto &i : decryptionMatrix)     {  *for* (auto &j : i)        {           cout << j << " ";        }        cout << endl;     }     string decryptedText = decryptionText(decryptionMatrix);     cout << endl          << "Decrypted Text: " << decryptedText << endl;  *return* 0;  }  **3B] Transposition with key:**  *#include* <bits/stdc++.h>  using namespace std;  *// Function to create the Columnar Matrix for encryption*  vector<vector<char>> initMatrix(const string &*plain*, string &*keyword*)  {     int index = 0;     int p = *plain*.size();     int k = *keyword*.size();     int n = (p + k - 1) / k;     vector<vector<char>> v(n, vector<char>(k, '\_'));  *for* (int i = 0; i < n; ++i)     {  *for* (int j = 0; j < k; ++j)        {  *if* (index < p)              v[i][j] = *plain*[index++];        }     }  *return* v;  }  vector<pair<char, int>> indices(string &*keyword*, bool *decrypt*)  {     vector<pair<char, int>> v;  *for* (int i = 0; i < *keyword*.size(); ++i)     {        v.push\_back({*keyword*[i], i});     }     sort(v.begin(), v.end(), [](const pair<char, int> &*a*, const pair<char, int> &*b*)          { *return* *a*.first < *b*.first; });  *return* v;  }  vector<vector<char>> rearrange(const vector<pair<char, int>> &*indx*, vector<vector<char>> &*matrix*, bool *decrypt*)  {     int row = *matrix*.size();     int col = *matrix*[0].size();     vector<vector<char>> rearranged(row, vector<char>(col, '\_'));  *for* (int c = 0; c < col; ++c)     {        int sortedIndex = *indx*[c].second;  *for* (int r = 0; r < row; ++r)        {  *if* (!*decrypt*)           {              rearranged[r][c] = *matrix*[r][sortedIndex];           }  *else*           {              rearranged[r][sortedIndex] = *matrix*[r][c];           }        }     }  *return* rearranged;  }  string encryptedText(vector<vector<char>> &*matrix*)  {     string cipherText = "";  *for* (auto &i : *matrix*)     {  *for* (auto &j : i)        {           cipherText += j;        }     }  *return* cipherText;  }  vector<vector<char>> cipherTextToDmatrix(string &*cipherText*, string &*keyword*)  {     int index = 0;     int k = *keyword*.size();     int c = *cipherText*.size();     int n = (c + k - 1) / k;     vector<vector<char>> v(n, vector<char>(k, '\_'));  *for* (int i = 0; i < n; ++i)     {  *for* (int j = 0; j < k; ++j)        {  *if* (index < c)              v[i][j] = *cipherText*[index++];        }     }  *return* v;  }  string decryptionText(vector<vector<char>> &*decryptionMatrix*)  {     string decryptedText = "";  *for* (auto &i : *decryptionMatrix*)     {  *for* (auto j : i)        {  *if* (j != '\_')              decryptedText += j;        }     }  *return* decryptedText;  }  int main()  {     string plain;     cout << "Enter the plain text: ";     cin >> plain;     string keyword;     cout << "Enter the keyword: ";     cin >> keyword;  *// Encryption*     vector<vector<char>> matrix = initMatrix(plain, keyword);     cout << endl          << "Encryption:" << endl          << endl          << "Plain text to encryption matrix: " << endl;  *for* (auto &i : matrix)     {  *for* (auto &j : i)        {           cout << j << " ";        }        cout << endl;     }     bool decrypt = false;     vector<pair<char, int>> indx = indices(keyword, decrypt);     matrix = rearrange(indx, matrix, decrypt);     cout << endl          << "Rearranged encryption matrix: " << endl;  *for* (auto &i : matrix)     {  *for* (auto &j : i)        {           cout << j << " ";        }        cout << endl;     }     string cipherText = encryptedText(matrix);     cout << endl          << "Cipher Text: " << cipherText << endl;     cout << endl          << "-----------------------------------------------------" << endl;  *// Decryption*     vector<vector<char>> decryptionMatrix = cipherTextToDmatrix(cipherText, keyword);     cout << endl          << "Decryption:" << endl;     cout << endl          << "Cipher text to decryption matrix: " << endl;  *for* (auto &i : decryptionMatrix)     {  *for* (auto &j : i)        {           cout << j << " ";        }        cout << endl;     }     decrypt = true;     decryptionMatrix = rearrange(indx, decryptionMatrix, decrypt);     cout << endl          << "Rearranged decryption matrix: " << endl;  *for* (auto &i : decryptionMatrix)     {  *for* (auto &j : i)        {           cout << j << " ";        }        cout << endl;     }     string decryptedText = decryptionText(decryptionMatrix);     cout << endl          << "Decrypted Text: " << decryptedText << endl;  *return* 0;  } |

| **Results/Output:**  **1] Caeser Cipher**    **2A] PlayFair without key**    **2B] PlayFair with key**    **3A] Transposition without key**    **3B] Transposition with key** |
| --- |

| **Conclusion:** |
| --- |
| This experiment demonstrated secure text transformation using Playfair Cipher, highlighting encryption, decryption, preprocessing, and matrix-based substitution techniques. |

| **Vlab:** |
| --- |
| **Breaking The Shift Cipher:**      **Breaking the Mono-alphabetic Substitution Cipher:** |

| **Post-Lab Questions:** |
| --- |
| **Q1] Points of Difference Between Mono-Alphabetic Cipher and Poly-Alphabetic Cipher:**   1. **Substitution Type**:    * Mono-alphabetic cipher uses a single substitution for all characters.    * Poly-alphabetic cipher uses multiple substitutions based on the position of characters. 2. **Complexity**:    * Mono-alphabetic ciphers are simpler and easier to break using frequency analysis.    * Poly-alphabetic ciphers are more complex and resist frequency analysis better. 3. **Key Usage**:    * Mono-alphabetic cipher has a fixed key for substitution.    * Poly-alphabetic cipher uses a repeating or varying key.   **Q2] Working of a Rail-Fence Cipher with Example:**  The Rail-Fence cipher is a transposition cipher that writes the plain text in a zigzag pattern across multiple rows and then reads it row by row.  **Example**: **PlainText:** GEEKSFORGEEKS Key = 3   * Number of columns in matrix = len(cipher-text) = 13 * Number of rows = key = 3  1. Arrange in a zigzag pattern:      1. Read row by row: GSGS EKFREK EOE   **Cipher Text**: GSGSEKFREKEOE  **Q3] Applications of Cryptography:**   1. **Data Security**: Cryptography secures sensitive data, ensuring its confidentiality and integrity during transmission or storage. 2. **Digital Signatures**: Used for authentication and verifying the origin of messages in emails, contracts, and documents. 3. **Secure Communications**: Ensures safe communication in applications like WhatsApp, banking systems, and VPNs using encryption algorithms. |