|  |  |  |  |
| --- | --- | --- | --- |
| **Course Name:** | **Applied Cryptography (**116U01E628**)** | **Semester:** | **VI** |
| **Date of Performance:** | **15 / 01 / 2025** | **DIV/ Batch No:** | **C - 3** |
| **Student Name:** | **Romil Lodaya** | **Roll No:** | **16010122096** |

**Experiment No: 2**

**Title: Encryption-Decryption programs using classical cryptography**

|  |
| --- |
| **Aim and Objective of the Experiment:** |
| To write a program to convert plain text into cipher text using Transposition cipher |

|  |
| --- |
| **COs to be achieved:** |
| **CO1: Explain the fundamentals of Information Security and cryptography** |

|  |
| --- |
| **Books/ Journals/ Websites referred:** |
| 1. Stallings, W., Cryptography and Network Security: Principles and Practice, Second edition, Person Education 2. “Caesar Cipher in cryptography”, <https://www.geeksforgeeks.org/caesar-cipher-in-cryptography/>, last retrieved on Aug 01, 2023 3. “PlayFair Cipher in cryptography”: <https://www.geeksforgeeks.org/playfair-cipher-with-examples/>, last retrieved on Aug 01, 2023 4. “Transposition cipher in cryptology,  ”<https://www.britannica.com/topic/transposition-cipher>, last retrieved on Aug 01, 2023 |

|  |
| --- |
| **Theory:** |
| Transposition and substitution are the two types of cyphers employed in traditional cryptography. Substitution ciphers encrypt plaintext by changing the plaintext one piece at a time, while Transposition ciphers encrypt plaintext by moving small pieces of the message around. Anagrams are a primitive transposition cipher.  **Transposition cipher**   1. **Columnar Cipher:**   The logic behind transposition cipher is as shown in the given figure:     1. **Rail Fence Cipher:**   Example: We encipher NOTHING IS AS IT SEEMS by first writing it on two lines in a zig-zag pattern  (or rail fence). The ciphertext is produced by transcribing the first row followed by the second row.    Ciphertext: NTIGS STEMO HNIAI SES.  To decrypt, we write half the letters on one line, half on the second. (Note that if there are an odd number of letters, we include the “middle” letter on the top line.) |

|  |
| --- |
| **Code :** |
| **1A] Columnar 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;  }  **1B] Columnar 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;  }  **2] Rail Fence:**  #include <bits/stdc++.h>  using namespace std;  // Function to create the Rail Fence Matrix for encryption  vector<vector<char>> initMatrix(const string& plain, int n) {  int m = plain.size();  vector<vector<char>> matrix(n, vector<char>(m, '-'));  int r = 0, c = 0;  bool down = true;  for (char ch : plain) {  matrix[r][c++] = ch;  if (down) {  if (r == n - 1) {  down = false;  r--;  } else {  r++;  }  } else {  if (r == 0) {  down = true;  r++;  } else {  r--;  }  }  }  return matrix;  }  // Function to extract the cipher text from the Rail Fence Matrix  string cipherText(const vector<vector<char>>& matrix) {  string cipher = "";  for (const auto& row : matrix) {  for (char ch : row) {  if (ch != '-') {  cipher += ch;  }  }  }  return cipher;  }  // Function to create Rail Matrix for decryption  void decryptionMatrix(vector<vector<char>>& matrix, int n, int m) {  int r = 0, c = 0;  bool down = true;  for (int i = 0; i < m; ++i) {  matrix[r][c++] = '\*';  if (down) {  if (r == n - 1) {  down = false;  r--;  } else {  r++;  }  } else {  if (r == 0) {  down = true;  r++;  } else {  r--;  }  }  }  }  // Function to fill the Rail Matrix with the cipher text for decryption  void fillDecryptionMatrix(vector<vector<char>>& matrix, const string& cipher) {  int k = 0;  for (auto& row : matrix) {  for (char& ch : row) {  if (ch == '\*') {  ch = cipher[k++];  }  }  }  }  // Function to decrypt the text from the Rail Matrix  string decryptionText(const vector<vector<char>>& matrix, int n, int m) {  string decryptedText = "";  int r = 0, c = 0;  bool down = true;  for (int i = 0; i < m; ++i) {  decryptedText += matrix[r][c++];  if (down) {  if (r == n - 1) {  down = false;  r--;  } else {  r++;  }  } else {  if (r == 0) {  down = true;  r++;  } else {  r--;  }  }  }  return decryptedText;  }  int main() {  string plain;  cout << "Enter the plain text: ";  cin >> plain;  int n;  cout << "Enter the key: ";  cin >> n;  // Encryption  vector<vector<char>> matrix = initMatrix(plain, n);  cout << "\nRail Fence Matrix (Encryption):\n";  for (const auto& row : matrix) {  for (char ch : row) {  cout << ch;  }  cout << endl;  }  string cipher = cipherText(matrix);  cout << "\nCipher Text: " << cipher << endl;  // Decryption  int m = plain.size();  decryptionMatrix(matrix, n, m);  cout << "\nRail Fence Matrix (Marked for Decryption):\n";  for (const auto& row : matrix) {  for (char ch : row) {  cout << ch;  }  cout << endl;  }  fillDecryptionMatrix(matrix, cipher);  cout << "\nRail Fence Matrix (Filled with Cipher Text):\n";  for (const auto& row : matrix) {  for (char ch : row) {  cout << ch;  }  cout << endl;  }  string decryptedText = decryptionText(matrix, n, m);  cout << "\nDecrypted Text: " << decryptedText << endl;  return 0;  } |

|  |
| --- |
| **Output:** |
| **1A] Columnar without Key:**  **https://lh7-rt.googleusercontent.com/docsz/AD_4nXeiAP9YGxSo2nj5hj9aL88hEoZYWxRdHjik2qiQNaaefY0LucfWAkVsjQXQQ-dKDLG2ruS735VXfYgrjTznm94F7myOdsM8znwjn2WD_OH1-WqNU8rEQD3AGt8TM3ZlBgL00P8_?key=M_OWUKVQ-um5aAYG2z3Z2wkg**  **1B] Columnar with Key:**  https://lh7-rt.googleusercontent.com/docsz/AD_4nXf41CP_dqaEspyVxF7IeBXA_Uy91STm1KPyl8bMAARMjS4kRO-hT9w2admKKytms4bxA-d9tOi67bFdmOIjpPg64pJ6B5UbSbnLB7uAggYt6OMX64TvHEP9Ex8DMJmOVcu3SI_gTg?key=M_OWUKVQ-um5aAYG2z3Z2wkg  **2] Rail Fence:**  https://lh7-rt.googleusercontent.com/docsz/AD_4nXe-aJD0Th3UAqqb3wqvgZ40Qjro58-BUlBBPuZDb8JdEG_Z67j0y6sXogq2eET5GDFFgJSZpRVYTkCZT6KJG72wctw1XEgW0kxTNoD9bGm_MjCWAQN2Jr9TnVryaop01ZF2cINM?key=M_OWUKVQ-um5aAYG2z3Z2wkg |

|  |
| --- |
| **Post Lab Subjective/Objective type Questions:** |
| 1. Explain the key differences between substitution ciphers   Substitution ciphers are encryption methods in which each element of the plaintext (typically a letter) is replaced by another symbol according to a specific rule or key. The key differences among substitution ciphers lie in how the substitutions are carried out:   * **Caesar Cipher**: A type of substitution cipher where each letter in the plaintext is shifted by a fixed number of positions in the alphabet. For example, a Caesar cipher with a shift of 3 would map 'A' to 'D', 'B' to 'E', etc. It is a simple form of substitution and is relatively easy to break. * **Monoalphabetic Substitution Cipher**: In this cipher, each letter of the plaintext is replaced by a unique letter of the alphabet. The mapping is fixed but can be randomly chosen, meaning each letter has a one-to-one correspondence with another letter. Unlike the Caesar cipher, the shift is not fixed and can vary across the message. * **Polyalphabetic Substitution Cipher**: This cipher uses multiple cipher alphabets to encrypt the message, reducing the predictability of letter frequencies and making the cipher more resistant to frequency analysis. The most famous polyalphabetic cipher is the Vigenère cipher, where each letter in the plaintext is substituted according to a key that repeats throughout the message.   The primary difference among these ciphers is the complexity of the substitution system and how easily they can be broken by modern cryptanalysis techniques.   1. In the Playfair cipher, why is it important to replace repeated letters in a digraph   In the Playfair cipher, plaintext is encrypted in pairs of letters, known as digraphs. However, if a digraph contains two identical letters (e.g., "LL" or "EE"), it poses a problem because the Playfair cipher requires two distinct letters for encryption. To address this issue, repeated letters in a digraph are typically replaced with a filler letter (commonly 'X').  For example:   * The digraph "LL" could be replaced with "LX" or "XL."   This step is crucial because:   * It ensures that every digraph consists of two distinct letters, which is required by the encryption algorithm. * It prevents patterns from appearing in the ciphertext, making it more secure and less susceptible to cryptanalysis, which could occur if repeating letters were present.   Without replacing repeated letters, the cipher would lose its effectiveness and introduce weaknesses that attackers could exploit.   1. "Weak security is worse than no security." Do you agree? Justify your answer with example.   Yes, I agree that weak security is worse than no security. The reason is that weak security can create a false sense of safety, leading individuals or organizations to believe their data is protected when, in fact, it is highly vulnerable to attacks. If attackers can easily break into a system with weak encryption, they may gain access to sensitive information.  **Example:** Suppose an organization uses an outdated encryption algorithm like DES (Data Encryption Standard), which uses a 56-bit key. Due to modern computational power, it is possible to break this encryption using brute-force attacks in a short amount of time. This makes the encryption essentially useless. If no encryption is used at all, the system's data is immediately exposed, and the risk is clear to the users or administrators. With weak encryption, however, there is a false impression of security, causing individuals to neglect proper security practices and assume their data is safe when it's not. This is why weak security can be more damaging than having no security at all—it encourages complacency, which can lead to significant data breaches. |

|  |
| --- |
| **Conclusion:** |
| The experiment demonstrates implementing the Transposition cipher, enhancing understanding of classical cryptography and its role in information security. |