| **Course Name:** | **Applied Cryptography (216H02L401)** | **Semester:** | **IV** |
| --- | --- | --- | --- |
| **Date of Performance:** | **06 / 01 / 2025** | **DIV/ Batch No:** | **C - 3** |
| **Student Name:** | **Romil Lodaya** | **Roll No:** | **16010122096** |

**Experiment No: 1**

**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 Caesar cipher and 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 ciphers 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.  **Substitutional Cipher:**   1. **Caesar Cipher**   The earliest known, and the simplest, use of a substitution cipher was by Julius Caesar.  The Caesar cipher involves replacing each letter of the alphabet with the letter standing three places further down the alphabet.  For example,    Note that the alphabet is wrapped around, so that the letter following Z is A.  We can define the transformation by listing all possibilities, as follows:    Let us assign a numerical equivalent to each letter:    The algorithm can be expressed as follows. For each plaintext letter p, substitute the ciphertext letter C:  *C* = E(3, *p*) = (*p* + 3) mod 26  A shift may be of any amount, so the general Caesar algorithm is  C = E(k, p) = (p + k) mod 26  where k takes on a value in the range 1 to 25.  The decryption algorithm is simply  p = D(k, C) = (C - k) mod 26  If it is known that a given cipher text is a Caesar cipher, then a brute-force cryptanalysis is easily performed. Simply try all the 25 possible keys.   1. **Play fair cipher:**   The Playfair cipher was the first practical digraph substitution cipher. The scheme was invented in 1854 by Charles Wheatstone but was named after Lord Playfair who promoted the use of the cipher. In playfair cipher unlike traditional cipher we encrypt a pair of alphabets(digraphs) instead of a single alphabet.  The Playfair Cipher Encryption algorithm consists of 2 steps:   1. **Generate the key Square(5×5):**   - The key square is a 5×5 grid of alphabets that acts as the key for encrypting the plaintext. Each of the 25 alphabets must be unique and one letter of the alphabet (usually J) is omitted from the table (as the table can hold only 25 alphabets). If the plaintext contains J, then it is replaced by I.  -The initial alphabets in the key square are the unique alphabets of the key in the order in which they appear followed by the remaining letters of the alphabet in order.  e.g. consider Key= Monarchy  KeySquare:  Rules for Encryption 3   1. **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(bogus character) is added to the last letter.   Example 1:  PlainText: "instruments"  After Split: 'in' 'st' 'ru' 'me' 'nt' 'sz'  Example 2: (With ‘x’ as the bogus letter.)  Plain Text: “hello”  After Split: ‘he’ ‘lx’ ‘lo’  Rules for Encryption:   1. If both the letters are in the same column: Take the letter below each one (going back to the top if at the bottom).   For example:  **Diagraph:** "me"  **Encrypted Text:** cl  **Encryption:**  m -> c  e -> l  Rules for Encryption 1   1. **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). **For example:**   **Diagraph:** "st"  **Encrypted Text:** tl  **Encryption:**  s -> t  t -> l  Rules for Encryption 2   1. **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. **For example:**   **Diagraph:** "nt"  **Encrypted Text:** rq  **Encryption:**  n -> r  t -> q  **For example:**   **Plain Text:** "instrumentsz"  **Encrypted Text:** gatlmzclrqtx  **Encryption:**  i -> g  n -> a  s -> t  t -> l  r -> m  u -> z  m -> c  e -> l  n -> r  t -> q  s -> t  z -> x  Decryption in Playfair Cipher: |

| **Code :** |
| --- |
| **1] Caeser Cipher:**  #include<bits/stdc++.h>  using namespace std;  int main()  {  cout << "Enter the mode, Encrypt - 0, Decrypt - 1" << endl;  int mode;  cin >> mode;  if(mode == 0)  {  cout << "Enter the Plain Text: ";  string plain;  cin >> plain;  for(int key = 1; key <= 25; ++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;  for(int key = 1; key <= 25; ++key)  {  for(auto &c : cipher)  {  int index = c - 'a';  index = (index - key);  if(index < 0)index += 26;  cout << (char)('a' + index);  }  cout << endl;  }  }  return 0;  }  **2A] Play Fair without Key:**  #include <bits/stdc++.h>  using namespace std;  void initMatrix(vector<vector<char>>& matrix)  {  char c = 'a';  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) {  // Replace 'j' with 'i'  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;  }  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) {  // Find positions of the characters in the matrix  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 << plainText << endl;  if (!plainText.empty() && plainText.back() == padding.back()) {  plainText.erase(plainText.size() - 1);  }  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;  cout << "Decryption" << endl;  vector<pair<char, char>> cipher = divideText(cipherText);  string decryptedText = processText(cipher, matrix, false);  decryptedText = removePadding(decryptedText, padding);  cout << 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;  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) {  // Replace 'j' with 'i'  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;  }  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) {  // Find positions of the characters in the matrix  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 << plainText << endl;  if (!plainText.empty() && plainText.back() == padding.back()) {  plainText.erase(plainText.size() - 1);  }  return plainText;  }  int main() {  vector<vector<char>> matrix(5, vector<char>(5));  string padding = "x";  string key = "monarchy";  initMatrix(matrix, key);  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;  cout << "Decryption" << endl;  vector<pair<char, char>> cipher = divideText(cipherText);  string decryptedText = processText(cipher, matrix, false);  decryptedText = removePadding(decryptedText, padding);  cout << decryptedText << endl;  return 0;  } |

| **Output:** |
| --- |
| **1] Caese Cipher:**      **2A] Play Fair without Key:**    **2B] Play Fair with Key:** |

| **Post Lab Subjective/Objective type Questions:** |
| --- |
| 1. Explain the key differences between substitution ciphers   Substitution ciphers are a type of encryption where each element of the plaintext (typically a letter) is replaced with another symbol according to a certain system or rule. There are different types of substitution ciphers, and the key differences lie in how the substitutions are made:   * **Caesar Cipher**: A simple substitution cipher where each letter in the plaintext is shifted by a fixed number of positions in the alphabet. For example, with a shift of 3, A becomes D, B becomes E, and so on. * **Monoalphabetic Substitution Cipher**: Each letter of the plaintext is substituted by a different letter from the alphabet, but this mapping is fixed for the entire message. Unlike the Caesar cipher, the shift is not constant, and any letter can map to any other letter. For example, A could map to Q, B to Z, and so on. * **Polyalphabetic Substitution Cipher**: Unlike the monoalphabetic cipher, this method uses multiple cipher alphabets to encode the plaintext, making it much harder to crack through frequency analysis. The most famous polyalphabetic cipher is the Vigenère cipher, where different alphabets are used depending on a repeating keyword.   The key differences between these ciphers are the complexity of the substitution and the ease with which they can be broken. A Caesar cipher is quite weak because of its predictability, while a polyalphabetic cipher is more secure due to the use of multiple alphabets.   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 (called digraphs). A critical aspect of this cipher is that if a digraph contains two identical letters (e.g., "LL"), the encryption process would be problematic because there would be no way to map these two identical letters to a different set of letters in the 5x5 matrix. To solve this issue, repeated letters in a digraph are typically replaced with a filler letter (often 'X') to ensure the two letters are distinct. For example, "LL" would become "LX" or "XL."  This step is crucial because:   * It ensures the cipher works as intended by guaranteeing each digraph consists of two different letters. * It prevents simple patterns from appearing in the ciphertext, which could potentially be exploited by cryptanalysts to break the encryption.  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. This is because weak security can create a false sense of protection, leading individuals or organizations to believe their data is safe when, in fact, it is vulnerable to attacks. A system with weak encryption can be easily broken into by attackers, exposing sensitive information.  **Example:** Consider a scenario where an organization uses an outdated encryption algorithm like DES (Data Encryption Standard) to protect its sensitive data. DES is now considered weak because it uses a 56-bit key, which can be cracked relatively quickly using modern computing power (a brute-force attack on DES can be done in a few hours or less). If an attacker breaks the encryption, they gain access to all the sensitive data.  On the other hand, using no encryption at all would expose the data immediately to any potential attacker. While that is clearly insecure, it is at least transparent—users and organizations know that their data is at risk, and they can take precautions accordingly (e.g., avoiding storing sensitive data in the first place). With weak encryption, users might be lulled into a false sense of security, assuming that their data is protected, when in fact, it is vulnerable to attack. This is why weak security can be more dangerous than having no security at all—it can lead to complacency and negligence in properly securing valuable assets. |

| **Conclusion:** |
| --- |
| Implementing Caesar and Substitution ciphers demonstrate fundamental encryption techniques, enhancing understanding of information security and cryptographic principles. |