|  |  |  |  |
| --- | --- | --- | --- |
| **Course Name:** | **Information Theory and Coding Techniques** | **Semester:** | **V** |
| **Date of Performance:** | **05 / 11 / 2024** | **Batch No:** | **B - 1** |
| **Faculty Name:** | **Prof. Makarand Kulkarni** | **Roll No:** | **16014022050** |
| **Faculty Sign & Date:** |  | **Grade/Marks:** | **\_\_\_ / 25** |

**Experiment No.: 8**

**Title:** Encryption and Decryption Techniques

|  |
| --- |
| **Aim and Objective of the Experiment:** |
| To study and implement different encryption and decryption algorithm. |

|  |
| --- |
| **COs to be achieved:** |
| **CO5:** Students will be able to compare and contrast symmetric and asymmetric encryption systems and respective key management. |

|  |
| --- |
| **Theory:** |
| **Encryption and Decryption Algorithms**  **Substitution Methods of Encryption**  Substitution methods replace characters in plaintext with other characters or symbols to obscure the original message. Each letter or group of letters is systematically replaced, based on a defined rule.   1. Caesar Cipher   The Caesar Cipher is one of the simplest and most well-known substitution ciphers, attributed to Julius Caesar. This cipher involves shifting each letter in the plaintext by a fixed number of positions along the alphabet   * How It Works: If the shift is 3, the letter 'A' becomes 'D', 'B' becomes 'E', and so on. The alphabet wraps around, so after 'Z', it cycles back to 'A'. * Encryption Formula: E(x)=(x+k) mod 26, where x is the position of the letter in the alphabet, and k is the fixed shift. * Weakness: Since there are only 26 possible shifts in the English alphabet, the Caesar Cipher is easily broken by brute force.  1. Vigenère Square Cipher   The Vigenère Cipher is a polyalphabetic substitution cipher that uses a keyword to shift each letter in the plaintext by varying amounts.   * How It Works: A Vigenère table or "Vigenère square" is created, with each row representing a Caesar Cipher shifted by one position relative to the row above. The keyword is repeated to match the length of the plaintext. * Encryption Formula: For each letter 𝑃𝑖 in the plaintext, the corresponding letter 𝐾𝑖 in the keyword determines the shift. The encrypted character 𝐶𝑖 is found by shifting 𝑃𝑖 by the alphabetical position of 𝐾𝑖. * Strengths and Weaknesses: The Vigenère Cipher is more resistant to frequency analysis, as it uses multiple Caesar Ciphers in sequence. However, if the keyword is short or repetitive, patterns may emerge, making it vulnerable to attacks like the Kasiski examination.   **Transpositional Methods of Encryption**  Transpositional methods rearrange the characters in the plaintext according to a specific system, rather than substituting them with other characters.   1. Multiplication Cipher   The Multiplication Cipher is a form of transpositional cipher that encrypts each character in the plaintext by multiplying its position in the alphabet by a fixed number.   * How It Works: Each letter is mapped to a number (e.g., A=1, B=2, etc.), then multiplied by a fixed integer key. The result is taken modulo 26 to find the encrypted character's position in the alphabet. * Encryption Formula: E(x)=(k × x)mod 26, where x is the numerical position of the letter, and k is the multiplier. * Weakness: The cipher can be broken if the multiplicative key is not coprime to 26 (for English), as it will not generate a unique mapping for each letter.  1. Affine Cipher   The Affine Cipher combines both multiplication and addition to transform each character's position in the plaintext. It’s a combination of a multiplicative cipher and Caesar Cipher.   * How It Works: Each character is first multiplied by a key and then shifted by an additional key. The result is then taken modulo 26. * Encryption Formula: E(x)=(ax + b)mod 26, where a and b are keys, and a must be coprime to 26 to ensure each letter has a unique mapping. * Strengths and Weaknesses: This combination of two transformations increases complexity and security compared to Caesar and Multiplication ciphers. However, if an attacker determines the keys, it can be cracked using mathematical analysis. |

|  |
| --- |
| **Observations:** |
| **Codes for different encryption methods**   1. **Substitution Methods of Encryption**    1. **Caesar Cipher**   def caesar\_cipher(text, key):      encrypted\_text = ""        for char in text:          if char.isalpha():              is\_upper = char.isupper()              char = char.lower()                original\_position = ord(char) - ord('a') + 1                new\_position = (original\_position + key) % 26              if new\_position == 0:                  new\_position = 26                new\_char = chr(new\_position + ord('a') - 1)                if is\_upper:                  new\_char = new\_char.upper()                encrypted\_text += new\_char          else:              encrypted\_text += char      return encrypted\_text  text = input("Enter the text to encrypt: ")  key = int(input("Enter the key (integer): "))  encrypted\_text = caesar\_cipher(text, key)  print("Encrypted text:", encrypted\_text)     * 1. **Vignere Square Cipher**   # vigenere square  def print\_vigenere\_square(plaintext, key):      key\_length = len(key)      extended\_key = (key \* (len(plaintext) // key\_length)) + key[:len(plaintext) % key\_length]        print("\nVigenère Square:")      print("Plaintext:  ", " ".join(plaintext))      print("Key        :", " ".join(extended\_key))      print("Matrix:")        for i, char in enumerate(plaintext):          if char.isalpha():              plain\_position = ord(char.lower()) - ord('a')              key\_position = ord(extended\_key[i].lower()) - ord('a')                row = [(chr((j + key\_position) % 26 + ord('a'))) for j in range(26)]                print(f"{char.upper()} (Key: {extended\_key[i].upper()}): {' '.join(row)}")          else:              print(f"{char} (No encryption)")  plaintext = input("Enter the text to encrypt: ").replace(" ", "").upper()  # Remove spaces for simplicity  key = input("Enter the encryption key (string): ").upper()  print\_vigenere\_square(plaintext, key)     1. **Transpositional Methods of Encryption**    1. **Multiplication Cipher**   # multiplicative cipher  def multiplicative\_encrypt(plaintext, key):      alphabet = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'      letter\_to\_num = {alphabet[i]: i+1 for i in range(26)}      num\_to\_letter = {i+1: alphabet[i] for i in range(26)}      num\_to\_letter[0] = 'Z'      plaintext = plaintext.upper()      encrypted\_text = ""      for char in plaintext:          if char in letter\_to\_num:              num = letter\_to\_num[char]              encrypted\_num = (num \* key) % 26              encrypted\_char = num\_to\_letter[encrypted\_num]              encrypted\_text += encrypted\_char          else:              encrypted\_text += char      return encrypted\_text  plaintext = input("Enter the plaintext: ")  key = int(input("Enter the key (integer): "))  encrypted\_text = multiplicative\_encrypt(plaintext, key)  print("Encrypted text:", encrypted\_text)     * 1. **Affine Cipher**   # affine cipher  def affine\_encrypt(plaintext):      alphabet = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'      letter\_to\_num = {alphabet[i]: i + 1 for i in range(26)}      num\_to\_letter = {i + 1: alphabet[i] for i in range(26)}      num\_to\_letter[0] = 'Z'      plaintext = plaintext.upper()      encrypted\_text = ""      for char in plaintext:          if char in letter\_to\_num:              num = letter\_to\_num[char]              num = num + 3              encrypted\_num = (num \* 5) % 26              encrypted\_char = num\_to\_letter[encrypted\_num]              encrypted\_text += encrypted\_char          else:              encrypted\_text += char      return encrypted\_text  plaintext = input("Enter the plaintext: ")  encrypted\_text = affine\_encrypt(plaintext)  print("Encrypted text:", encrypted\_text) |

|  |
| --- |
| **Post Lab Subjective/Objective type Questions:** |
| **Verifying methods theoretically.** |

|  |
| --- |
| **Conclusion:** |
| Successfully learnt and implemented different encryption and decryption techniques to make data being transferred more secure. |

|  |
| --- |
| **Signature of faculty in-charge with Date:** |