Experiment 5

5.1 Aim:-Design and Implement a product cipher using Substitution ciphers using Substitution ciphers.

5.2Course Outcome:-Identify the basic cryptographic techniques using classical and block encryption methods.

5.3 Lab Objective:-To design, implement, and evaluate a multi-layered product cipher using substitution techniques for enhanced cryptographic security.

5.4 Requirement:-

- A programming language (Python, Java, or any preferred language).
- Knowledge of substitution ciphers (Caesar cipher, mono-alphabetic cipher, etc.).
- A plaintext message for encryption and decryption.

5.5 Theory:

- **Product Cipher**:
  A product cipher combines two or more simple ciphers (e.g., substitution, permutation) in sequence to achieve stronger encryption. This concept is based on Shannon's principle of "confusion and diffusion."
- **Substitution Cipher**:
  A substitution cipher replaces each letter of plaintext with another letter based on a predefined rule or key. Examples include the Caesar cipher and mono-alphabetic substitution.
- By layering substitution ciphers, the product cipher increases complexity and resilience against frequency analysis attacks.

Design of the Cipher

Key Elements:

1. First Substitution Layer: Caesar cipher with a shift key (e.g., shift = 3).
2. Second Substitution Layer: Mono-alphabetic substitution using a random key.
3. Third Substitution Layer: Affine cipher defined by $E(x)=(a \cdot x+b) \mod 26$ $E(x) = (a \cdot x + b) \mod 26$ $E(x)=(a \cdot x+b) \mod 26$, where aaa and bbb are keys.

5.6 Procedure:-

Encryption Process

1. Apply the Caesar cipher to the plaintext.
2. Use mono-alphabetic substitution to replace the output from Step 1.
3. Apply the affine cipher to further encrypt the output from Step 2.

Decryption Process

1. Reverse the affine cipher on the ciphertext.
2. Apply the reverse of the mono-alphabetic substitution.
3. Decrypt using the Caesar cipher to recover the plaintext.

5.7 Result :-

```python
import random
import string
def caesar_encrypt(text, shift):
    result = ""
    for char in text:
        if char.isalpha():
            shift_base = ord('A') if char.isupper() else ord('a')
            result += chr((ord(char) - shift_base + shift) % 26 + shift_base)
        else:
            result += char
    return result
def caesar_decrypt(text, shift):
    return caesar_encrypt(text, -shift)
def generate_monoalphabetic_key():
    letters = list(string.ascii_lowercase)
    shuffled_letters = letters[:]
    random.shuffle(shuffled_letters)
    return dict(zip(letters, shuffled_letters)), dict(zip(shuffled_letters, letters))

def monoalphabetic_encrypt(text, key_map):
    return "".join(key_map.get(char, char) for char in text.lower())

def monoalphabetic_decrypt(text, rev_key_map):
    return "".join(rev_key_map.get(char, char) for char in text.lower())
def mod_inverse(a, m):
    for i in range(m):
        if (a * i) % m == 1:
            return i
    return None
def affine_encrypt(text, a, b):
    if mod_inverse(a, 26) is None:
        raise ValueError("Invalid 'a' value! Must be coprime to 26.")
    result = ""
    for char in text:
        if char.isalpha():
            shift_base = ord('A') if char.isupper() else ord('a')
            result += chr(((a * (ord(char) - shift_base) + b) % 26) + shift_base)
        else:
            result += char
    return result
```

```python
def affine_decrypt(text, a, b):
    a_inv = mod_inverse(a, 26)
    result = ""
    for char in text:
        if char.isalpha():
            shift_base = ord('A') if char.isupper() else ord('a')
            result += chr(((a_inv * ((ord(char) - shift_base) - b)) % 26) + shift_base)
        else:
            result += char
    return result
def encrypt_product_cipher(plaintext, caesar_shift, a, b, mono_key):
    print(f"Original Text: {plaintext}")
    caesar_encrypted = caesar_encrypt(plaintext, caesar_shift)
    print(f"After Caesar Cipher: {caesar_encrypted}")
    mono_encrypted = monoalphabetic_encrypt(caesar_encrypted, mono_key)
    print(f"After Monoalphabetic Cipher: {mono_encrypted}")
    affine_encrypted = affine_encrypt(mono_encrypted, a, b)
    print(f"After Affine Cipher (Final Ciphertext): {affine_encrypted}")
    return affine_encrypted
def decrypt_product_cipher(ciphertext, caesar_shift, a, b, mono_reverse_key):
    print(f"\nCiphertext Received: {ciphertext}")
    affine_decrypted = affine_decrypt(ciphertext, a, b)
    print(f"After Reversing Affine Cipher: {affine_decrypted}")
    mono_decrypted = monoalphabetic_decrypt(affine_decrypted, mono_reverse_key)
    print(f"After Reversing Monoalphabetic Cipher: {mono_decrypted}")
    caesar_decrypted = caesar_decrypt(mono_decrypted, caesar_s
    print(f"After Reversing Caesar Cipher (Final Plaintext): {caesar_decrypted}")
    return caesar_decrypted
plaintext = "hello product cipher"
caesar_shift = 3
a, b = 5, 8
mono_key, mono_reverse_key = generate_monoalphabetic_key()
ciphertext = encrypt_product_cipher(plaintext, caesar_shift, a, b, mono_key)
decrypted_text = decrypt_product_cipher(ciphertext, caesar_shift, a, b, mono_reverse_key)
print("\n✅ Encryption & Decryption Successful!" if plaintext == decrypted_text else "❌ Error in process!")

```

(variable) caesar_decrypted: str | Any

```
Original Text: hello product cipher
After Caesar Cipher: khoor surgxfw flskhu
After Monoalphabetic Cipher: wpcct betvfsk szbwpe
After Affine Cipher (Final Ciphertext): ofssz nczjhug udnofc

Ciphertext Received: ofssz nczjhug udnofc
After Reversing Affine Cipher: wpcct betvfsk szbwpe
After Reversing Monoalphabetic Cipher: khoor surgxfw flskhu
After Reversing Caesar Cipher (Final Plaintext): hello product cipher

✅ Encryption & Decryption Successful!
```

5.8 Conclusion:-

the implementation of the multi-layered product cipher using Caesar cipher, mono-alphabetic substitution, and affine cipher demonstrated an effective approach to enhancing encryption security. By combining these substitution techniques, the cipher complexity was increased, making it more resistant to attacks like frequency analysis. This practical highlighted the importance of layering simple ciphers to achieve stronger cryptographic security.