PANDIT DEENDAYAL ENERGY UNIVERSITY

SCHOOL OF TECHNOLOGY

Course: Information Security

Course Code: 20CP304P

LAB MANUAL

B.Tech. (Computer Engineering)

Semester 5

Submitted To:

Dr. Hargeet Kaur

Submitted By:

Aryan Randeriya

22BCP469D

G3

INDEX

S. No.	List of experiments	Date	Sign
1	Study and Implement a program for Caesar Cipher		
2	Study and Implement a program for 5x5 Playfair Cipher to encrypt and decrypt the message.		
3	Study and Implement a program for Rail Fence Cipher with columnar transposition		
4	Study and implement a program for columnar Transposition Cipher		
5	Study and implement a program for Vigenère Cipher		
6	Study and Implement a program for n-gram Hill Cipher		
7	Study and Use of RSA algorithm (encryption and decryption)		
8	Study and implement a program of the Digital Signature with RSA algorithm (Reverse RSA)		
9	Study and Use of Diffie-Hellman Key Exchange		
10	Design cipher with detailed explanation. Sample as follows. a) Write a program to encrypt the plaintext with the given key. E.g. plaintext GRONSFELD with the key 1234. Add 1 to G to get H (the letter 1 rank after G is H in the alphabet), then add 2 to C or E (the letter 2 ranks after C is E), and so on. Use smallest letter from plaintext as filler. b) Encrypt the input words PLAINTEXT= RAG BABY to obtain CIPHERTEXT = SCJ DDFD		

Experiment 4

AIM: Study and Implement a program for Vigenere Cipher

Introduction:

The Vigenère cipher is a method of encrypting alphabetic text where each letter of the plaintext is encoded with a different Caesar cipher, whose increment is determined by the corresponding letter of another text, the key.

Program:

```
def generate_key(message, key):
    key = list(key)
    if len(message) == len(key):
        return key
    else:
        for i in range(len(message) - len(key)):
            key.append(key[i % len(key)])
    return ''.join(key)


def encrypt(message, key):
    cipher_text = []
    for i in range(len(message)):
        char = (ord(message[i]) + ord(key[i])) % 26
        char += ord('A')
        cipher_text.append(chr(char))
    return ''.join(cipher_text)


def decrypt(cipher_text, key):
    original_text = []
    for i in range(len(cipher_text)):
        char = (ord(cipher_text[i]) - ord(key[i]) + 26) % 26
        char += ord('A')
        original_text.append(chr(char))
    return ''.join(original_text)


def main():
    choice = input("Do you want to (E)ncrypt or (D)ecrypt? ").upper()

    if choice == 'E':
        plaintext = input("Enter the plaintext (A-Z only): ").upper()
        key = input("Enter the key (A-Z only): ").upper()
        key = generate_key(plaintext, key)
        ciphertext = encrypt(plaintext, key)
        print("Encrypted Message:", ciphertext)

    elif choice == 'D':
        ciphertext = input("Enter the ciphertext (A-Z only): ").upper()
```

```
        key = input("Enter the key (A-Z only): ").upper()
        key = generate_key(ciphertext, key)
        plaintext = decrypt(ciphertext, key)
        print("Decrypted Message:", plaintext)

    else:
        print("Invalid choice! Please choose either E or D.")


if __name__ == "__main__":
    main()
```

Output:

o

```
Do you want to (E)ncrypt or (D)ecrypt? E
Enter the plaintext (A-Z only): aryan
Enter the key (A-Z only): hello
Encrypted Message: HVJLB
```

```
Do you want to (E)ncrypt or (D)ecrypt? D
Enter the ciphertext (A-Z only): hvjlb
Enter the key (A-Z only): hello
Decrypted Message: ARYAN
```

Revised Approach

The revised approach of Vigenère Cipher uses a key to shift letters in a message, with the key's length adjusted to match the message length. It then applies random shifts to each letter based on the key, both for encryption and decryption. The key is extended if needed, and the shifts are generated using a seeded random function to ensure consistency.

Code:

```
import random


def generate_key(message, key):
    key = list(key)
    if len(message) == len(key):
        return key
    else:
        for i in range(len(message) - len(key)):
            key.append(key[i % len(key)])
    return ''.join(key)


def random_shift(key):
    random.seed(sum(ord(char) for char in key))
    shifts = [random.randint(1, 25) for _ in range(len(key))]
    return shifts


def encrypt(message, key):
    key = generate_key(message, key)
    shifts = random_shift(key)
    cipher_text = []

    for i in range(len(message)):
        char = (ord(message[i]) - ord('A') + shifts[i]) % 26
        char += ord('A')
        cipher_text.append(chr(char))

    return ''.join(cipher_text)


def decrypt(cipher_text, key):
    key = generate_key(cipher_text, key)
    shifts = random_shift(key)
    original_text = []

    for i in range(len(cipher_text)):
        char = (ord(cipher_text[i]) - ord('A') - shifts[i]) % 26
        char += ord('A')
        original_text.append(chr(char))

    return ''.join(original_text)
```

```python
def main():
    choice = input("Do you want to (E)ncrypt or (D)ecrypt? ").upper()

    if choice == 'E':
        plaintext = input("Enter the plaintext (A-Z only): ").upper()
        key = input("Enter the key (A-Z only): ").upper()
        ciphertext = encrypt(plaintext, key)
        print("Encrypted Message with random shifts:", ciphertext)

    elif choice == 'D':
        ciphertext = input("Enter the ciphertext (A-Z only): ").upper()
        key = input("Enter the key (A-Z only): ").upper()
        plaintext = decrypt(ciphertext, key)
        print("Decrypted Message with random shifts:", plaintext)

    else:
        print("Invalid choice! Please choose either E or D.")


if __name__ == "__main__":
    main()
```

Output:

```
Do you want to (E)ncrypt or (D)ecrypt? E
Enter the plaintext (A-Z only): aryan
Enter the key (A-Z only): hello
Encrypted Message with random shifts: EOWMV
```

```
Do you want to (E)ncrypt or (D)ecrypt? D
Enter the ciphertext (A-Z only): eowmv
Enter the key (A-Z only): hello
Decrypted Message with random shifts: ARYAN
```

Comparative Analysis of original and revised approach:

Original Approach: The original method uses a simple shift based on the ASCII values of characters from the key. While it shifts letters predictably, making it easy to understand and use, it also means the encryption is less secure and can be more easily broken.

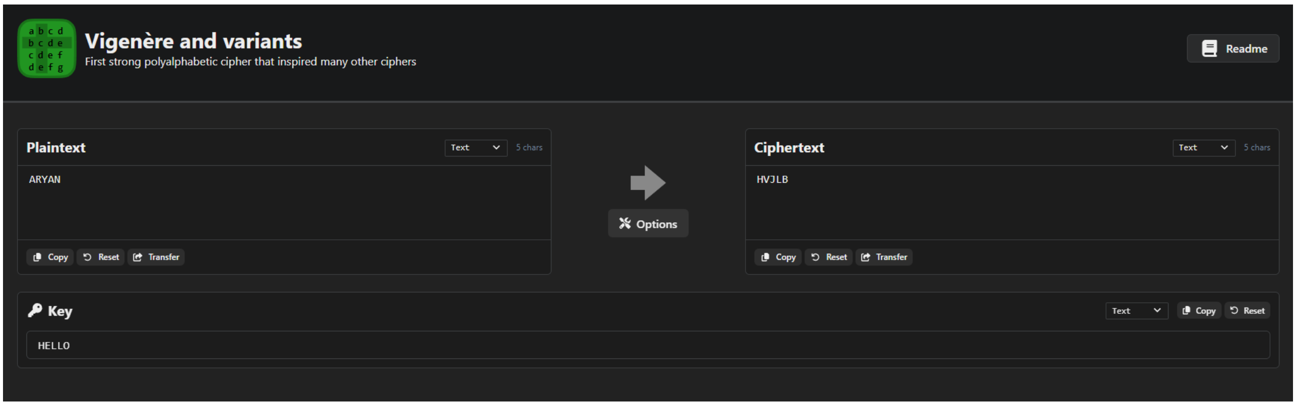
Revised Approach: The revised method also shifts letters using the key but is more straightforward and consistent. It lacks the randomness of the original method, which makes it simpler but potentially less secure. Overall, the revised approach is easier to use but not as strong in terms of security.

Crypt Analysis:

Brute Force Attack:
For short keywords, all possible combinations can be tried until the correct one is found. With longer keywords, this becomes less feasible, but for weak keys or smaller key lengths, it can still be effective.

Known Plaintext Attack:
If part of the plaintext and the corresponding ciphertext are known, the keyword can be derived by comparing these segments. Once the keyword is partially known, it can be used to decrypt other parts of the ciphertext.

CrypTool Output:



Conclusion:
The Vigenère Cipher, in its original form, provides basic encryption by shifting letters based on a repeating keyword. This method, while more secure than simple ciphers like the Caesar Cipher, can still be relatively easy to break using techniques such as frequency analysis or known plaintext attacks. The revised approach enhances the original by incorporating random shifts, which adds a layer of complexity and variability.

References:

- https://pages.mtu.edu/~shene/NSF-4/Tutorial/VIG/Vig-Base.html
- https://brilliant.org/wiki/vigenere-cipher/