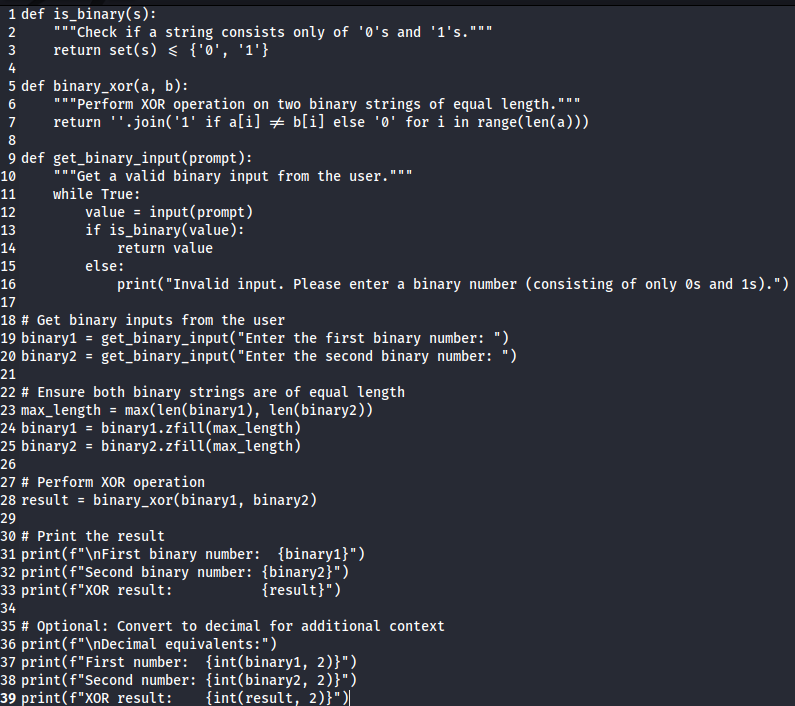
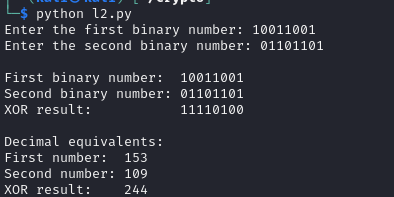
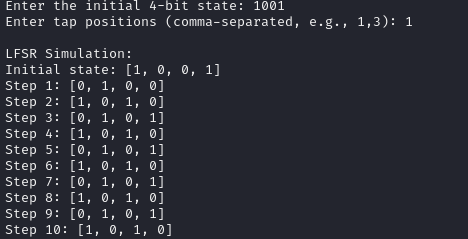
1. Write a python script to get the binary values from the user and perform XOR operation.

2. Write a Python script that implements a simple 4-bit LFSR. The initial state of the register and the tap positions should be user inputs.

Simulate 10 steps of the LFSR, displaying the state of the register at each step.

A screenshot of a computer program

Description automatically generated 

1. Write a report on attacks on LFSR. Explain any one attack in detail.
2. **# Attacks on Linear Feedback Shift Registers (LFSRs)**
3. **## Introduction**
4. Linear Feedback Shift Registers (LFSRs) are widely used in cryptography, particularly in stream ciphers and pseudo-random number generators. However, their linear nature makes them vulnerable to several types of attacks. This report outlines the major attacks on LFSRs and provides a detailed explanation of one particularly effective attack.
5. **## Common Attacks on LFSRs**
6. 1. **\*\*Berlekamp-Massey Algorithm Attack\*\***: Used to determine the shortest LFSR that can produce a given output sequence.
7. 2. **\*\*Algebraic Attack\*\***: Exploits the algebraic structure of the LFSR to set up and solve a system of linear equations.
8. 3. **\*\*Correlation Attack\*\***: Targets stream ciphers that use multiple LFSRs, exploiting statistical weaknesses.
9. 4. **\*\*Fast Correlation Attack\*\***: An improved version of the correlation attack, using error-correcting codes.
10. 5. **\*\*Known Plaintext Attack\*\***: Uses known plaintext-ciphertext pairs to deduce the LFSR state.
11. 6. **\*\*Divide-and-Conquer Attack\*\***: Applies to stream ciphers using multiple LFSRs, attacking each LFSR separately.
12. **## Detailed Analysis: The Berlekamp-Massey Algorithm Attack**
13. The Berlekamp-Massey algorithm is a powerful method for cryptanalyzing LFSR-based systems. It can determine the shortest LFSR capable of generating a given output sequence, effectively reverse-engineering the LFSR structure.
14. **### How the Attack Works**
15. 1. **\*\*Observation\*\***: The attacker observes a portion of the LFSR's output sequence.
16. 2. **\*\*Algorithm Application\*\***: The Berlekamp-Massey algorithm is applied to this sequence.
17. 3. **\*\*LFSR Reconstruction\*\***: The algorithm determines:
18. - The length of the shortest LFSR that could produce the sequence
19. - The feedback polynomial of this LFSR
20. 4. **\*\*State Recovery\*\***: Once the LFSR structure is known, the initial state can often be determined by solving a system of linear equations.
21. **### Mathematical Basis**
22. The Berlekamp-Massey algorithm works by iteratively constructing a Linear Recurrence Relation (LRR) that generates the observed sequence. At each step, it either:
23. - Increases the length of the LFSR if the current one can't generate the next bit, or
24. - Adjusts the feedback taps without increasing length if possible
25. The algorithm's complexity is O(n^2), where n is the length of the observed sequence.
26. **### Example**
27. Consider a 4-bit LFSR with the output sequence: 1011010110...
28. 1. The algorithm starts with a 1-bit LFSR and progressively increases its size.
29. 2. It might determine that a 4-bit LFSR with feedback polynomial x^4 + x^3 + 1 generates this sequence.
30. 3. The attacker can then deduce that the original LFSR likely had this structure.
31. **### Implications for Cryptography**
32. 1. **\*\*Vulnerability of Simple LFSRs\*\***: This attack demonstrates why single, short LFSRs are inadequate for secure cryptographic systems.
33. 2. **\*\*Required Sequence Length\*\***: To break an n-bit LFSR, approximately 2n bits of output are needed.
34. 3. **\*\*Countermeasures\*\***:
35. - Use longer LFSRs (e.g., 128 bits or more)
36. - Combine multiple LFSRs in non-linear ways
37. - Introduce irregular clocking or output functions
38. **## Conclusion**
39. The Berlekamp-Massey algorithm attack, along with other attacks on LFSRs, highlights the importance of using more complex structures in cryptographic systems. While LFSRs remain useful components, they must be employed carefully and in combination with other techniques to ensure cryptographic security.
40. Understanding these attacks is crucial for cryptographers to design more robust systems and for security analysts to assess the strength of existing cryptographic implementations.

BONUS POINT:

4. write a python script to break hill cipher (2X2) using known plain text attack.

Known Plaintext: "MEET"

Corresponding Ciphertext: "URRG"

import numpy as np

def letter\_to\_number(letter):

return ord(letter) - ord('A')

def number\_to\_letter(number):

return chr(number % 26 + ord('A'))

def matrix\_mod\_inv(matrix, modulus):

"""Calculate the modular inverse of a 2x2 matrix"""

det = int(np.round(np.linalg.det(matrix)))

det\_inv = pow(det % modulus, -1, modulus)

adjugate = np.array([[matrix[1, 1], -matrix[0, 1]],

[-matrix[1, 0], matrix[0, 0]]])

return (adjugate \* det\_inv) % modulus

# Known plaintext and ciphertext

plaintext = "MEET"

ciphertext = "URRG"

# Convert plaintext and ciphertext to number matrices

P = np.array([[letter\_to\_number(plaintext[0]), letter\_to\_number(plaintext[2])],

[letter\_to\_number(plaintext[1]), letter\_to\_number(plaintext[3])]])

C = np.array([[letter\_to\_number(ciphertext[0]), letter\_to\_number(ciphertext[2])],

[letter\_to\_number(ciphertext[1]), letter\_to\_number(ciphertext[3])]])

# Calculate the key matrix

P\_inv = matrix\_mod\_inv(P, 26)

K = (C @ P\_inv) % 26

print("Recovered Key Matrix:")

print(K)

# Verify the key by encrypting the plaintext

def encrypt(plaintext, key):

result = ""

for i in range(0, len(plaintext), 2):

chunk = np.array([letter\_to\_number(plaintext[i]),

letter\_to\_number(plaintext[i+1])])

encrypted = (key @ chunk) % 26

result += number\_to\_letter(encrypted[0]) + number\_to\_letter(encrypted[1])

return result

encrypted = encrypt(plaintext, K)

print(f"\nVerification:")

print(f"Original Plaintext: {plaintext}")

print(f"Known Ciphertext: {ciphertext}")

print(f"Encrypted with recovered key: {encrypted}")

if encrypted == ciphertext:

print("\nSuccess! The recovered key correctly encrypts the plaintext to the known ciphertext.")

else:

print("\nError: The recovered key does not produce the expected ciphertext.")

=========================================================================================================================

“Dream, Dream, Dream. Dreams transform into thoughts and thoughts result in action.” - Dr. APJ

=========================================================================================================================