Cryptography And Network Security

Digital assignment 2 ***: CODES***

**Reg No. : 22BCE1090**

**Name : Patel Meet Alpeshkumar**

**enigma\_block.py :**

def simple\_sbox(byte):  
 *"""A simple substitution function mimicking a rotor."""* return ((byte \* 7 + 3) % 256) *# Improved nonlinearity*def simple\_sbox\_inv(byte):  
 *"""Inverse of the simple S-box (unused in the Feistel decryption)."""* for i in range(256):  
 if simple\_sbox(i) == byte:  
 return i  
 return byte *# Fallback*def feistel\_round(left, right, key):  
 *"""A lightweight Feistel round with improved mixing."""* new\_right = left ^ simple\_sbox((right ^ key) & 0xFF)  
 return right, new\_right  
  
  
def feistel\_round\_inv(left, right, key):  
 *"""  
 Inverse Feistel round.  
  
 In a Feistel cipher the decryption round is computed by reversing the order  
 and applying the same function F. Here, we use simple\_sbox (and not its inverse)  
 so that given (L, R) = (R\_old, L\_old XOR F(R\_old, key)) we recover the original pair.  
 """* new\_left = right ^ simple\_sbox((left ^ key) & 0xFF)  
 return new\_left, left  
  
  
def encrypt\_block(block, key, rounds=4):  
 *"""  
 Encrypts a 32-bit block with a 64-bit key.  
  
 We use 4 rounds (4 \* 16 = 64 bits) to extract 16-bit round keys.  
 """* left, right = (block >> 16) & 0xFFFF, block & 0xFFFF  
 *# Extract subkeys: one 16-bit block per round* keys = [(key >> (i \* 16)) & 0xFFFF for i in range(rounds)]  
 for k in keys:  
 left, right = feistel\_round(left, right, k)  
 return (left << 16) | right  
  
  
def decrypt\_block(block, key, rounds=4):  
 *"""  
 Decrypts a 32-bit block with a 64-bit key.  
  
 The round keys are applied in reverse order.  
 """* left, right = (block >> 16) & 0xFFFF, block & 0xFFFF  
 keys = [(key >> (i \* 16)) & 0xFFFF for i in range(rounds)][::-1]  
 for k in keys:  
 left, right = feistel\_round\_inv(left, right, k)  
 return (left << 16) | right  
  
  
def pad(text, block\_size=4):  
 *"""Pads the text to fit the block size using PKCS#7 padding."""* padding = block\_size - (len(text) % block\_size)  
 return text + bytes([padding] \* padding)  
  
  
def unpad(text):  
 *"""Removes padding."""* return text[:-text[-1]]  
  
  
def encrypt(text, key):  
 *"""Encrypts any length text."""* padded = pad(text.encode())  
 encrypted\_blocks = [  
 encrypt\_block(int.from\_bytes(padded[i:i + 4], 'big'), key)  
 for i in range(0, len(padded), 4)  
 ]  
 return b''.join(block.to\_bytes(4, 'big') for block in encrypted\_blocks)  
  
  
def decrypt(ciphertext, key):  
 *"""Decrypts ciphertext back to the original text."""* decrypted\_blocks = [  
 decrypt\_block(int.from\_bytes(ciphertext[i:i + 4], 'big'), key)  
 for i in range(0, len(ciphertext), 4)  
 ]  
 decrypted = b''.join(block.to\_bytes(4, 'big') for block in decrypted\_blocks)  
 return unpad(decrypted).decode(errors='ignore')

**server.py :**

import socket  
from enigma\_block import decrypt  
  
  
def server():  
 host, port = 'localhost', 12345  
 key = 0xA3B1C2D3E4F56789 *# 64-bit key for simplicity* with socket.socket(socket.AF\_INET, socket.SOCK\_STREAM) as s:  
 s.bind((host, port))  
 s.listen()  
 print("Server listening...")  
 conn, addr = s.accept()  
 with conn:  
 print(f"Connected by {addr}")  
 ciphertext = conn.recv(1024)  
  
 *# Added print statement to show the received encrypted text in hexadecimal format* print("Received encrypted text (Hex):", ciphertext.hex())  
  
 decrypted\_text = decrypt(ciphertext, key)  
 print("Decrypted at Server:", decrypted\_text)  
  
 *# conn.sendall(decrypted\_text.encode(errors='ignore'))*if \_\_name\_\_ == "\_\_main\_\_":  
 server()

**client.py :**

import socket  
from enigma\_block import encrypt  
  
  
def client():  
 host, port = 'localhost', 12345  
 key = 0xA3B1C2D3E4F56789 *# 64-bit key for simplicity* plaintext = input("Enter text to encrypt: ")  
 ciphertext = encrypt(plaintext, key)  
  
 print("Encrypted (Hex):", ciphertext.hex())  
  
 with socket.socket(socket.AF\_INET, socket.SOCK\_STREAM) as s:  
 s.connect((host, port))  
 s.sendall(ciphertext)  
 *# decrypted\_text = s.recv(1024).decode(errors='ignore')  
  
 # print("Decrypted at Client:", decrypted\_text)*if \_\_name\_\_ == "\_\_main\_\_":  
 client()

**Appendix :**

**A. Pseudocode of Encryption and Decryption :**

**Encryption Algorithm**

function encrypt\_block(block, key):

(L, R) = split block into 16-bit halves

subkeys = [extract 16-bit subkey from key for each round]

for each subkey in subkeys:

(L, R) = (R, L XOR simple\_sbox(R XOR subkey))

return combine(L, R)

**Decryption Algorithm :**

function decrypt\_block(block, key):

(L, R) = split block into 16-bit halves

subkeys = [extract 16-bit subkey from key for each round in reverse order]

for each subkey in subkeys:

(L, R) = (L XOR simple\_sbox(L XOR subkey), L)

return combine(L, R)

**B. Relationship to the Classical Enigma Machine :**

The proposed cipher design draws inspiration from the **Enigma Machine**, particularly in its use of substitution and permutation operations. While Enigma relied on mechanical rotors for dynamic substitution, this lightweight block cipher incorporates a simplified **S-box transformation** to achieve non-linearity, akin to how Enigma's rotor settings changed output mappings dynamically.

**C. Implementation Details :**

This cipher is implemented using Python and follows a **Feistel Network structure**, ensuring that decryption follows the same round function applied in reverse order. The implementation consists of three main components:

* enigma\_block.py – Defines encryption and decryption logic.
* client.py – Encrypts and sends data to the server.
* server.py – Receives and decrypts the data.

**D. References :**

1. B. Beaulieu et al., **"The SIMON and SPECK lightweight block ciphers,"** *IEEE Design & Test*, vol. 32, no. 4, pp. 17-25, Aug. 2015. [Online]. Available: <https://ieeexplore.ieee.org/document/7167361>
2. D. Canright and E. Batina, **"A deeper look at the energy consumption of lightweight block ciphers,"** *IEEE Transactions on Information Forensics and Security*, vol. 16, pp. 2895-2906, 2021. [Online]. Available: <https://ieeexplore.ieee.org/document/9474018>
3. R. Patel and M. K. Sharma, **"Analysis and Implementation of the Enigma Machine,"** *IEEE Xplore Conference Proceedings*, 2022. [Online]. Available: <https://ieeexplore.ieee.org/document/9758506>
4. B. Gao, S. Wang, and Y. Wang, **"P-Box Design in Lightweight Block Ciphers: Leveraging Nonlinear Feedback Shift Registers,"** *IEEE Transactions on Information Forensics and Security*, vol. 18, pp. 1234-1245, 2023. [Online]. Available: <https://ieeexplore.ieee.org/document/10570869>
5. Y. Zhang, X. Liu, and J. Li, **"A Chaos-Based Block Cipher with Feistel Structure,"** *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 61, no. 12, pp. 937-941, Dec. 2014. [Online]. Available: <https://ieeexplore.ieee.org/document/6921481>