CRYPTOGRAPHY AND NETWORK SECURITY

DIGITAL ASSIGNMENT 2

Reg No. : 22BCE1090

Name : Patel Meet Alpeshkumar

Assignment Question:

Lightweight cryptography is essential for resource-constrained environments such as IoT devices, RFID tags, and embedded systems. Your task is to **design a lightweight block cipher** that balances security, efficiency, and low computational cost.

ENIGLITE: A LIGHTWEIGHT BLOCK CIPHER :

I have designed **Eniglite**, a lightweight block cipher **inspired by the classical Enigma machine**, specifically for resource-constrained environments. This cipher follows a **Feistel network** structure, ensuring both efficiency and ease of decryption by applying the same round function in reverse.

Key Features:

- **Feistel Structure:** Simplifies decryption while maintaining strong diffusion.
- **Optimized Block & Key Size:** Uses a **32-bit block size** and a **64-bit key**, striking a balance between security and computational efficiency.
- **Nonlinear S-box:** The S-box introduces nonlinearity through a simple mathematical transformation, enhancing confusion.
- **Lightweight Computation:** Each round consists of XOR operations, modular arithmetic, and a substitution step, making it suitable for low-power devices.

1. CIPHER DESIGN

- **Block Size:** 32-bit (4-byte) blocks for low memory and processing cost.
- **Key Size:** 64-bit key, split into four 16-bit subkeys for a balance of security and efficiency.
- **Structure:** Feistel Network, simplifying decryption by using the same round function in reverse.
- **Rounds:** 4 rounds, ensuring diffusion and confusion with minimal computational overhead.

Block & Key Size :

- **Block Size:** Encrypts **32-bit blocks**, minimizing processing time and memory use while fitting within lightweight cryptographic constraints.
- **Key Size:** Uses a **64-bit key**, divided into four 16-bit subkeys. This choice prioritizes efficiency over extreme security, making it ideal for low-resource devices.

Structure & Rounds :

- **Feistel Network:** Enables simple decryption using the same round function in reverse, even if the round function itself is non-invertible. The design remains efficient while integrating non-linear transformations.
- **Rounds:** Four rounds balance security and computational efficiency. Additional rounds could enhance security but would increase processing overhead.

Round Function Details :

- **S-box Function:**
  The function `*simple_sbox(byte)*` is defined as:
- *simple_sbox(byte) = (byte * 7 + 3) mod 256*

This function introduces the necessary nonlinearity for cryptographic strength.

- **Feistel Round:**
  In each round, the two 16-bit halves are processed as follows:

---

new_right = left XOR simple_sbox((right XOR key) & 0xFF)

(left, right) = (right, new_right)

For decryption, the same structure is applied in reverse order with the corresponding subkeys.

PSEUDOCODE SUMMARY

```
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)

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)
```

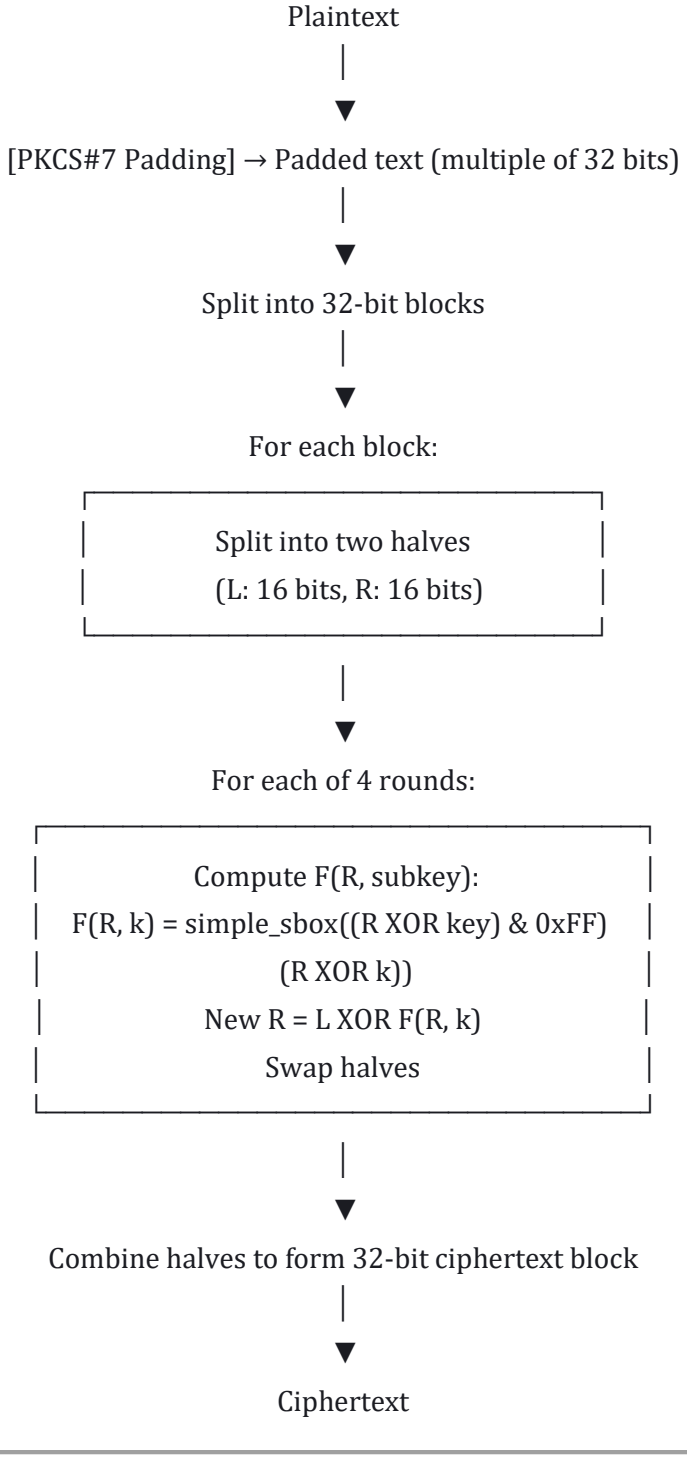
Detailed Encryption Process :

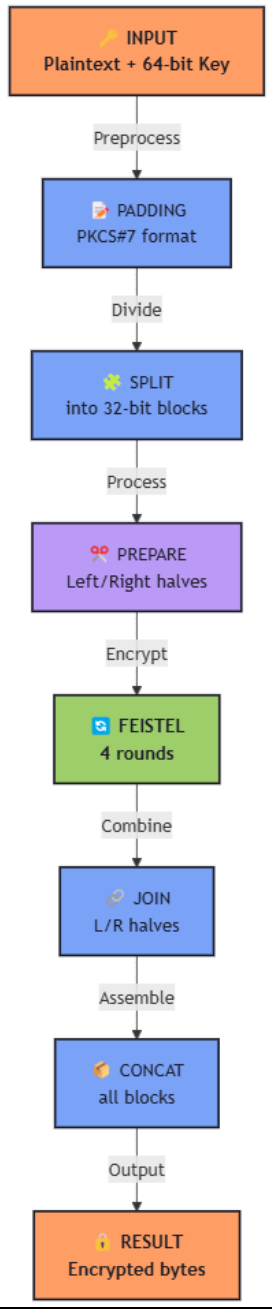
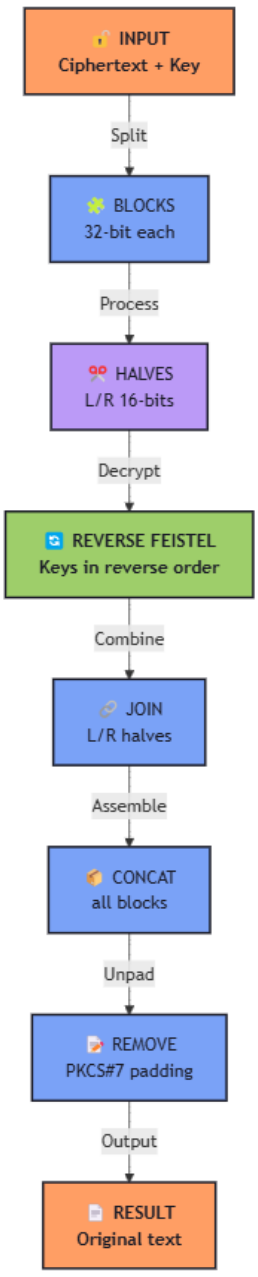
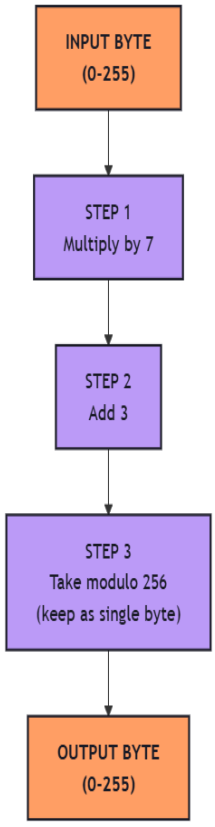
1. **Plaintext Preparation:**
    - **Padding:**
      The input plaintext is padded using PKCS#7 padding. This ensures that the total length is a multiple of the block size (32 bits or 4 bytes). For example, if the plaintext is not a multiple of 4 bytes, additional bytes (each containing the padding length) are appended.
2. **Block Processing:**
    - **Splitting into Blocks:**
      The padded plaintext is divided into 32-bit (4-byte) blocks.
    - **Splitting Each Block:**
      Each 32-bit block is split into two 16-bit halves:
        - **Left Half (L)**
        - **Right Half (R)**
3. **Feistel Rounds (4 Rounds):**
    - **Subkey Extraction:**
      The 64-bit key is divided into four 16-bit subkeys. Each round uses one of these subkeys.
    - **For Each Round:**
        - **Key Mixing:**
          The right half is XORed with the current subkey.
        - **Non-linear Substitution (S-box):**
          The result of the XOR is masked to 8 bits using & 0xFF and then passed through the simple_sbox function, which computes:
          simple_sbox(byte) = (byte * 7 + 3) mod 256
        - **Combining with Left Half:**
          The output of the S-box is XORed with the left half to produce the new right half.
        - **Swap Halves:**
          For the next round, the old right half becomes the new left half and the newly computed value becomes the new right half.
4. **Finalizing the Block:**
    - After all 4 rounds, the two 16-bit halves are combined to form a 32-bit ciphertext block.
5. **Output:**
    - All ciphertext blocks are concatenated to produce the final encrypted message.

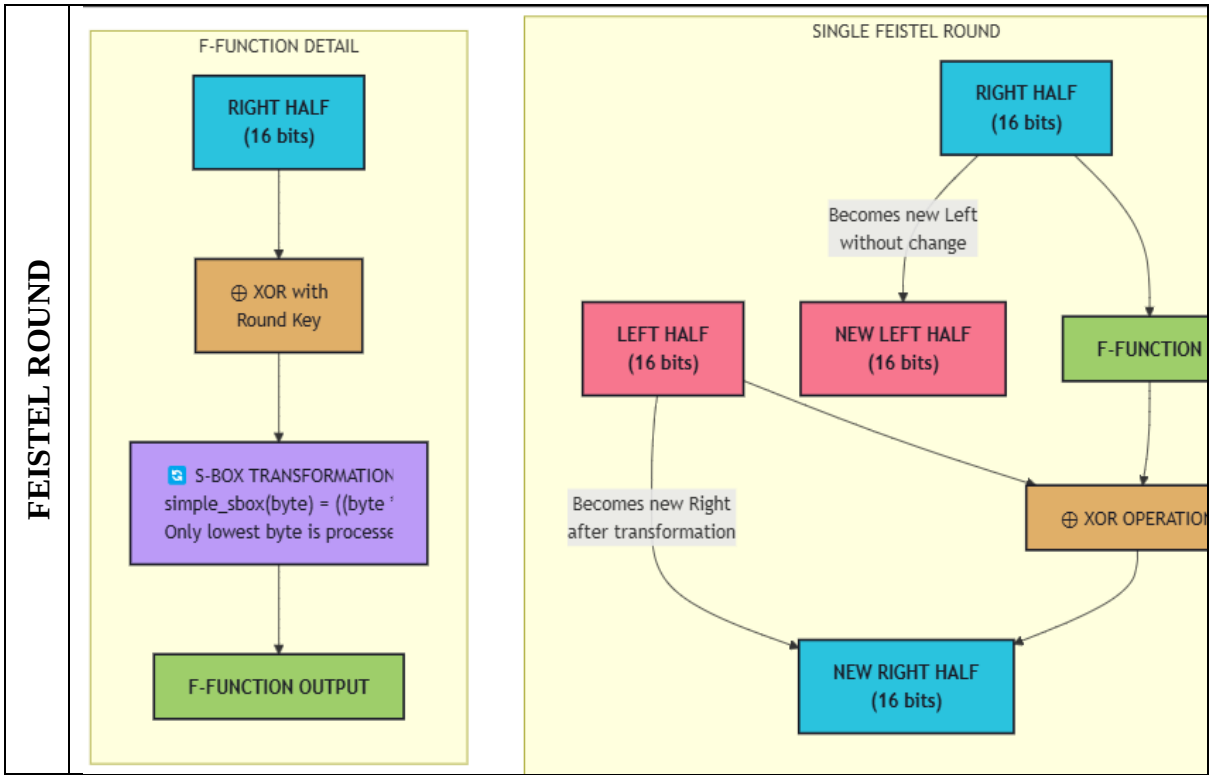
Detailed Decryption Process :

1. **Ciphertext Preparation:**
    - **Block Splitting:**
      The ciphertext is divided into 32-bit (4-byte) blocks.
2. **Block Processing:**
    - **Splitting Each Block:**
      Each 32-bit block is split into two 16-bit halves (L and R).
3. **Feistel Rounds (4 Rounds, in Reverse Order):**
    - **Subkey Extraction in Reverse:**
      The same 64-bit key is used, but the four 16-bit subkeys are applied in reverse order.
    - **For Each Round:**
        - **Reversed Operation:**
          The decryption round function (feistel_round_inv) is applied. It uses the left half and the current subkey (from the reversed key order):
            - The left half is XORed with the subkey, masked to 8 bits, and passed through the simple_sbox function.
            - The result is then XORed with the right half to recover the original left half.
        - **Swap Halves:**
          The halves are swapped back to undo the effect of the encryption round.
4. **Finalizing the Block:**
    - After all rounds have been processed, the two halves are combined to reconstruct the original 32-bit plaintext block.
5. **Removing Padding:**
    - Once all blocks have been decrypted and concatenated, the PKCS#7 padding is removed to retrieve the original plaintext message.

DIAGRAM OVERVIEW



Encryption	Decryption	S – BOX FUNCTION
		



2. SECURITY ANALYSIS

Potential Attacks & Mitigations :

- **Brute Force:**
  With a 64-bit key, there are 2^64 possible keys. Although this key space is smaller than that used in modern ciphers (which often employ 128 bits or more), it can be acceptable for many lightweight applications where extreme security is not the primary concern.
- **Differential & Linear Cryptanalysis:**
  - **Differential:** The nonlinearity introduced by the S-box, combined with the mixing properties of the Feistel structure, provides a degree of resistance to differential attacks.
  - **Linear:** Similarly, while the simple S-box may not be as robust as more complex alternatives, the overall structure still complicates the derivation of linear approximations.
- **Mitigation:**
  The multi-round diffusion and the simplicity of the operations ensure that the cipher remains computationally light while still offering a basic level of security. It is important to note that the use of only four rounds may reduce the margin against these attacks when compared to well-established ciphers.

Comparison with Existing Lightweight Ciphers :

Cipher	Block Size	Rounds	Key Size	Characteristics
PRESENT	64 bits	31	80 or 128 bits	Provides higher security but increases complexity and computational cost.
SIMON	32, 48, 64, 96, 128 bits	32–72 (varies by block/key size)	64, 72, 96, 128, 144, 192, 256 bits	Designed for efficiency in hardware; employs simple bitwise operations for lightweight encryption.
SPECK	32, 48, 64, 96, 128 bits	22–34 (varies by block/key size)	64, 72, 96, 128, 144, 192, 256 bits	Optimized for software efficiency; uses modular addition, XOR, and rotation for fast execution.
This Implementation	32 bits	4	64 bits	A simpler, educational design focusing on efficiency. Suitable for low-cost applications where extreme security is not required.

3. PERFORMANCE CONSIDERATIONS

Computational Cost :

- **Per Round Operations:**
    - **XOR for Key Mixing:**
      In the feistel_round function, the expression (right ^ key) performs an XOR to mix the right half with the subkey.
    - **S-box Lookup:**
      The result of the XOR is then passed to the simple_sbox function via simple_sbox((right ^ key) & 0xFF). This function performs a multiplication, an addition, and a modulo operation to introduce nonlinearity.
    - **Final XOR:**
      The output of the S-box is XORed with the left half using left ^ ... to produce the new right half.
- **Overall:**
  For each 32-bit block, four rounds are executed. This results in a small, fixed number of

operations per block, which is highly advantageous for devices with limited processing power.

Memory & Latency :

- **Memory Usage:**
  The design processes 32-bit blocks and only requires storage for a few 16-bit subkeys and temporary variables. This minimal memory footprint is ideal for 8-bit microcontrollers or similar devices with limited RAM.

- **Power Consumption & Latency:**
  The operations—comprising simple arithmetic and bitwise operations—are both low-power and fast. The low number of rounds contributes to reduced latency, making this cipher well-suited for real-time, resource-constrained environments.

4. IMPLEMENTATION & SIMULATION

Implementation Details :

- **Language:**
  The cipher is implemented in Python (see enigma_block.py), with additional modules (client.py and server.py) demonstrating its use in a networked application.

- **Functionality:**

  - **Encryption:**
    The implementation uses PKCS#7 padding to ensure that the plaintext fits the 32-bit block size. Each block is then processed through the Feistel rounds.

  - **Decryption:**
    The decryption process applies the rounds in reverse order (with reversed subkeys) to reconstruct the original plaintext.

- **Correctness Testing:**
  The code has been tested by entering plaintext, encrypting it, and then decrypting it on the server side. Print statements (showing the hexadecimal representation of the ciphertext) were added to help verify that both encryption and decryption processes are functioning as expected.

Performance Evaluation :

- **Encryption Speed:**
  Although Python may not be as optimized as C for embedded systems, the simplicity of the operations ensures that the encryption and decryption speeds are acceptable for academic purposes. For real-world applications on embedded systems, the algorithm could be ported to C.

- **Memory Footprint:**
  The design requires only a few bytes of memory per block, along with minimal additional storage for keys. This makes it ideal for low-memory environments such as 8-bit microcontrollers.

References :

- 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

- 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

- 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

- 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

- 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

Codes Link :

https://drive.google.com/file/d/1HtwseRAbpDcMgOvmOx90i2WuUFtNXbom/view?usp=sharing