DES Encryption/Decryption System

1. Introduction

This code details the design, implementation, and testing of a DES (Data Encryption Standard) encryption/decryption system. The system was implemented in Python and is built on a series of bitwise operations and permutation tables. The design emphasizes clarity and modularity, with a clear separation between key generation, encryption rounds, and decryption routines. In addition, intermediate round outputs are compared to verify the correctness of the encryption and decryption processes.

2. System Overview

2.1. Objectives

The primary goal of the system is to encrypt and decrypt messages using the DES algorithm. Key features of the system include:

- **Block-based Encryption:** The system processes plaintext in blocks of 64 bits (or 8 characters) at a time.
- **Generating Key:** It generates 16 round keys from an 8-character ASCII key using a series of permutations and bit shifts.
- **Round Functions:** The encryption process includes 16 rounds of complex bit manipulation, including expansion, substitution via S-boxes, and permutation.
- **Reversibility:** Decryption is performed by reversing the order of the round keys, thereby restoring the original plaintext.
- **Intermediate Verification:** The design provides intermediate outputs (e.g., after the 1st and 14th rounds) to help debug and verify that the encryption and decryption processes are exact inverses.

2.2. System Components

The system consists of several main components:

- **Permutation Tables:** Hardcoded tables for initial permutation, inverse permutation (final permutation), expansion, and straight (P-box) permutation.
- **S-boxes:** Eight S-boxes that perform non-linear substitution.
- **Key Generation Module:** Includes functions to drop parity bits, split the key, perform left shifts, and compress the key into 48-bit round keys.

- **Round Function (F-box):** Combines expansion, XOR with a subkey, S-box substitution, and straight permutation.
- **Encryption/Decryption Routines:** Methods for processing a single block and handling multiple blocks (text transformation, padding, and recombination).

3. Detailed System Design

3.1. Key Generation

The system accepts an 8-character ASCII key, which is converted into its 64-bit binary representation. The following key operations are performed:

1. **Parity Bit Drop:**
   The 64-bit key is reduced to 56 bits using the `parity_bit_drop_hash` table. This table omits every 8th bit (used as a parity check in the original DES specification).

Splitting and Shifting:
The 56-bit key is divided into two 28-bit halves. Each round requires these halves to be cyclically shifted left by a specified number of positions, as defined in the shifting schedule:
Python
The left-shift function is implemented so that the left and right halves wrap around seamlessly.

2. **Key Compression:**
   After each round's shift, the halves are concatenated to form a 56-bit key. This key is then compressed into a 48-bit subkey using the `key_compression_hash` table. This process is repeated for 16 rounds, generating 16 unique round keys stored for use during both encryption and decryption.

3.2. Encryption Process

The encryption process transforms plaintext into ciphertext in several steps:

1. **Text to Binary Conversion:**
   Plaintext is converted into a binary string. The input text is processed in blocks of 8 characters (64 bits). If the last block is not full, it is padded with zeros.
2. **Initial Permutation:**
   Each 64-bit block is permuted using the `initial_permutation` table. This rearrangement spreads the bits across the block to create a more uniform distribution before the rounds begin.
3. **Sixteen Rounds of Encryption:**
   The permuted block is divided into two halves (left and right, each 32 bits). For each round:

- **Expansion:**
  The right half is expanded from 32 bits to 48 bits using the `expansion_hash` table. This step duplicates certain bits to match the round key size.
- **Key Mixing:**
  The expanded right half is XORed with the current round's 48-bit subkey.
- **Substitution:**
  The resulting 48-bit block is split into eight 6-bit segments. Each segment is substituted with a 4-bit output using its corresponding S-box. The S-box substitution introduces non-linearity into the system.
- **Permutation (P-box):**
  The 32-bit output from the S-boxes is permuted using the `p_box` table. This further shuffles the bits.
- **XOR and Swap:**
  The output is then XORed with the left half. The halves are swapped (except during the final round) to prepare for the next round.

4. During the rounds, the system saves intermediate outputs (e.g., after the 1st and 14th rounds) for later comparison in the decryption process.
5. **Final Permutation:**
   After all rounds are complete, the two halves are concatenated (with a final swap applied) and processed using the `final_permutation` table. This step produces the final 64-bit ciphertext, which is then converted back to a textual format.

3.3. Decryption Process

Decryption reverses the encryption process by applying the round keys in reverse order. The same functions for initial permutation, round processing, and final permutation are used:

- The ciphertext is first converted from text back into binary blocks.
- The initial permutation is applied, followed by 16 rounds where the round keys are used in reverse.
- Intermediate outputs are again compared to verify that the first round of encryption matches the corresponding round in decryption.
- Finally, the final permutation yields the original plaintext block.

4. Testing and Verification

The system was tested with several plaintexts, such as:

- `"AnArDES!"`
- `"20712091"`
- `"cse350A2"`

For each test case, the process includes:

- **Encryption:**
  Converting plaintext to binary, processing through 16 rounds, and applying the final permutation.
- **Decryption:**
  Converting the ciphertext back into binary and processing through the decryption rounds.
- **Intermediate Round Comparisons:**
  The output after the 1st encryption round is compared with the output after the 15th decryption round; similarly, outputs from later rounds are compared. These checks ensure the encryption and decryption processes are true inverses.
- **Result Verification:**
  The recovered plaintext matches the original input, and intermediate outputs (presented in both binary and ASCII formats) are printed for debugging.

The printed output shows both bit-level representations and ASCII conversions, allowing for in-depth verification of each stage of the algorithm.

5. Conclusion

This DES-based system demonstrates a complete implementation of the classic DES algorithm. The code is modular and includes detailed key generation, encryption, and decryption functions. By comparing intermediate states during encryption and decryption, the system validates the reversibility of DES operations. Despite DES's limitations by modern cryptographic standards, this implementation serves as an educational tool for understanding block cipher design and bitwise operations. The code is well-documented, and extensive logging of intermediate rounds provides valuable insight into the inner workings of DES.

6. Reference

-https://www.nku.edu/~christensen/DESschneier.pdf
-lecture slides
-neso academy's tutorials

By- Angadjeet Singh(2022071)

Arav Amawate(2022091)