Data Encryption Standard

Implementation in Python

2017MCS2089 - Nichit Bodhak Goel
2017MCS2076 - Shadab Zafar

DES Introduction

- DES is a block cipher, meaning that it operates on plaintext blocks of 64 bits and returns ciphertext blocks of the same size.

- It is based on Feistel Cipher, and has 16 rounds.

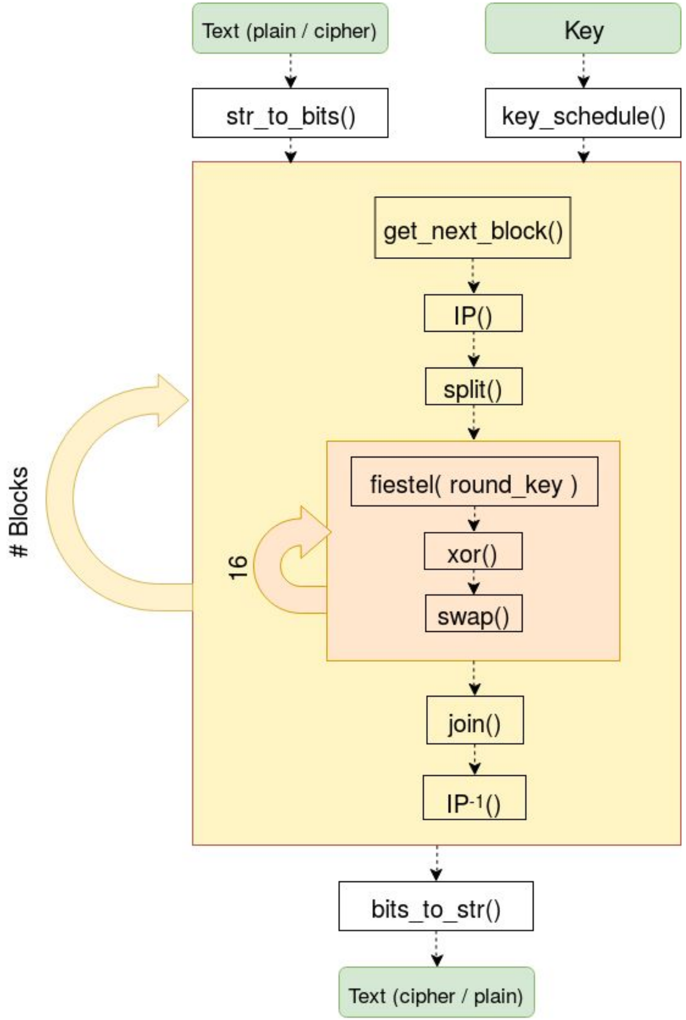
- Input block of 64 bits is divided into two blocks of 32 bits each, a left half block L and a right half R.

- Before the Feistel rounds, the input block is permuted using a special initial permutation, which is inverted after the Feistel rounds.

Project Overview

- We implemented DES using Python.
- For implementation, we followed official NIST document FIPS 46-3.
- There are 2 basic functions:
- **encrypt(plain_text, key)** and **decrypt(cipher_text, key)** which both call internally same function **des(text,key,typ)** where typ = **“encrypt” or “decrypt”** representing type of calling function.
- Key schedule is implemented by **key_schedule(key)**.
- Feistel box is implemented by **feistel(R,K)**
- To assist these functions, we used two self-created files
  - **constants.py** :- Contains DES constants like permutation matrices and S-Boxes.
  - **utils.py** :- Contains various utility functions like xor(), bits_to_hex() etc.

DES



Text (plain / cipher)
Key
str_to_bits()
key_schedule()
get_next_block()
IP()
split()
# Blocks
16
fiestel(round_key)
xor()
swap()
join()
IP-1()
bits_to_str()
Text (cipher / plain)

DES Encryption

- Here we take plain-text to be “4c6f7665676f6f64” and key to be “4e6576696c6c6c65” in hexadecimal format.
- Both plaintext and the key is converted to binary format and we get a 64 bit block of plain text and key.
- Out of 64 bits of key, 8 bits of key are redundant parity bits.
- Initial permutation is applied to plaintext.
- Permuted plaintext is split into L and R; key is used to initiate key schedule.
- Iterate over 16 rounds by invoking feistel function, XOR’ing and swapping appropriately.
- Perform final swap and inverse initial permutation.
- Ciphertext = “8c8fb94d1bac5358”

DES Encryption Output

```
Plain Text (Hexadecimal Format) = 4c6f7665676f6f64
Key (Hexadecimal Format) = 4e6576696c6c6c65


 0 - L 0: ff04ff7a    R 0: 00fe6376
 1 - L 1: 00fe6376    R 1: 2764bf12
 2 - L 2: 2764bf12    R 2: 18dc334e
 3 - L 3: 18dc334e    R 3: 672ebf32
 4 - L 4: 672ebf32    R 4: 40dc730e
 5 - L 5: 40dc730e    R 5: e706876a
 6 - L 6: e706876a    R 6: 90d42b2e
 7 - L 7: 90d42b2e    R 7: ff6eef62
 8 - L 8: ff6eef62    R 8: 18b64b5e
 9 - L 9: 18b64b5e    R 9: ff66b75a
10 - L10: ff66b75a    R10: 18d4336e
11 - L11: 18d4336e    R11: 3f46c70a
12 - L12: 3f46c70a    R12: d0de2326
13 - L13: d0de2326    R13: ff6e9f02
14 - L14: ff6e9f02    R14: c0f47b4e
15 - L15: c0f47b4e    R15: 2724bf52
16 - L16: 2724bf52    R16: c8d42b5e


Cipher Text (Hexadecimal Format) = 8c8fb94d1bac5358
```

DES Decryption

- Take ciphertext to be “8c8fb94d1bac5358” and key to be “4e6576696c6c65”.
- Apply initial permutation, split into two 32-bit blocks and swap.
- Iterate over 16-rounds in the reverse direction to invert the encryption.
- As XOR is its own inverse, this is trivial.
- For key schedule, reverse the shift schedule and right-shift instead of left.
- Finally we obtain plain-text by combining L and R and applying inverse initial permutation, Plain text = “4c6f7665676f6f64”.

DES Decryption Output

```
Cipher Text (Hexadecimal Format) = 8c8fb94d1bac5358
Key (Hexadecimal Format) = 4e6576696c6c6c65
```

```
 0 - L 0: c8d42b5e   R 0: 2724bf52
 1 - L 1: 2724bf52   R 1: c0f47b4e
 2 - L 2: c0f47b4e   R 2: ff6e9f02
 3 - L 3: ff6e9f02   R 3: d0de2326
 4 - L 4: d0de2326   R 4: 3f46c70a
 5 - L 5: 3f46c70a   R 5: 18d4336e
 6 - L 6: 18d4336e   R 6: ff66b75a
 7 - L 7: ff66b75a   R 7: 18b64b5e
 8 - L 8: 18b64b5e   R 8: ff6eef62
 9 - L 9: ff6eef62   R 9: 90d42b2e
10 - L10: 90d42b2e   R10: e706876a
11 - L11: e706876a   R11: 40dc730e
12 - L12: 40dc730e   R12: 672ebf32
13 - L13: 672ebf32   R13: 18dc334e
14 - L14: 18dc334e   R14: 2764bf12
15 - L15: 2764bf12   R15: 00fe6376
16 - L16: 00fe6376   R16: ff04ff7a
```

```
De-ciphered Plain Text (Hexadecimal Format) = 4c6f7665676f6f64
```

DES Validation

- To validate DES implementation: Output of the J^{th} encryption round should be identical to the output of the $(16-J)^{th}$ decryption round.
- In the encrypt() and decrypt() functions, store the intermediate outputs in an array.
- Finally assert that J^{th} and $(16-J)^{th}$ outputs of encrypt and decrypt respectively are equal.
- The program executes successfully with no assertion failures, thus validating our implementation.

TRIPLE - DES

We used three different keys (K1, K2, K3) to implement Triple-DES.

- Encryption :- Cipher Text = $E_{K3}(D_{K2}\ (E_{K1}(\text{Plain Text})\ )\ )$
- Decryption :- Plain Text = $D_{K1}(E_{K2}\ (D_{K3}(\text{Cipher Text})\ )\ )$

```
Plain Text (Hexadecimal Format) = 4c6f7665676f6f64
Key 1 (Hexadecimal Format) = 626564617a7a6c65
Key 2 (Hexadecimal Format) = 4d697261636c6573
Key 3 (Hexadecimal Format) = 4c6f67696369616e

Cipher Text (Hexadecimal Format) = 8cbfb9479ba01b18
```

```
Cipher Text (Hexadecimal Format) = 8cbfb9479ba01b18
Key 3 (Hexadecimal Format) = 4c6f67696369616e
Key 2 (Hexadecimal Format) = 4d697261636c6573
Key 1 (Hexadecimal Format) = 626564617a7a6c65

Deciphered Plain Text (Hexadecimal Format) = 4c6f7665676f6f64
```

Validation done by assertions.

References

1. FIPS 46-3 - The official NIST document describing DES.
2. DES - Wikipedia

THANK YOU