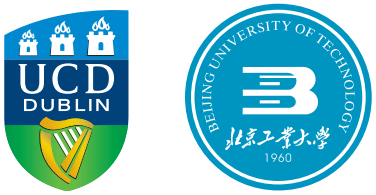

Beijing-Dublin International College

BDIC3025J Project 1 Report

Data Encryption Standard (DES) Implementation

Author

Sichen Li (BJUT ID: 21372309, UCD ID: 21207464)

Date

October 15, 2024

Abstract

This is the report for Project #1 of BDIC3025J Security and Privacy. The project is about implementing the Data Encryption Standard (DES) algorithm and a user interface for easily testing and evaluating the implementation.

1 Introduction

1.1 Data Encryption Standard (DES)

The Data Encryption Standard (DES) is a classic symmetric-key algorithm for encrypting digital data. It is a bit-oriented block cipher that operates on 64-bit blocks of plaintext with a 64-bit key (where 56-bit is used for encryption), and utilizes both transposition and substitution to encrypt the data. DES provides a secure way to protect information by transforming it into an unreadable format and was widely accepted as a secure encryption algorithm for a long time, until it was replaced by the Advanced Encryption Standard (AES) in 2001.

1.2 Project Overview

This project implements both the encryption and decryption parts of the DES algorithm in `Python` programming language. The implementation places no restrictions on the length and character type of input data and the key, which means the user can use any UTF-8 character of any length as the key and input data.

The project also provides a user-friendly UI for demonstrating the capability of the implementation. The UI allows the user to enter the key and plaintext/ciphertext and generate the ciphertext/plaintext accordingly. The user can also upload text files containing plaintext/ciphertext to encrypt/decrypt the content of the file.

1.3 Technologies Used

The project is implemented in `Python` and uses `Gradio` to build the user interface. `Gradio` is a Python library that allows users to quickly create customizable UI components to demonstrate the functionality of their algoirthms.

- Python 3.11
- Gradio 5.0

2 Algorithm Implementation

The implementation of the algorithm employs an object-oriented style and mainly consisted of two classes: `KeyGenerator` for generating round keys based on the key that the user provided, and `DES` for encrypting and decrypting the plaintext/ciphertext using the generated round keys. Apart from these two parts, there are also several utility functions that are used to convert the data from one format to another, performing `XOR` operation and pad/unpad the data to make sure the length of the data is a multiple of 64 bits.

2.1 Key Generation

The `KeyGenerator` class involves five methods:

- `prepare_key`: This method checks if the key is exactly 8-bit. If it is less than 8-bit, it pads the key with 0s to make it 8-bit. If it is more than 8-bit, it truncates the key to 8-bit.

The `prepare_key` method

```
def prepare_key(self, key: bytes) -> bytes:
    if len(key) < 8:
        return key.ljust(8, b'\0')
    elif len(key) > 8:
        return key[:8]
    return key
```

- `PC_1` and `PC_2`: These methods perform the permutation on the key using the `PC-1` and `PC-2` table.
- `left_shift`: This method performs the left shift operation on the key.
- `generate_keys`: This method generates the 16 round keys based on the key that the user provided. It shifts the key based on the following shift schedule:

$$Shift\_Schedule = (1, 1, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 1)$$

The `generate_keys` method

```
def generate_keys(self) -> list:
    left, right = self.PC_1()
    round_keys = []
    SHIFT_SCHEDULE = [1, 1, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 1]

    for shift in SHIFT_SCHEDULE:
        left = self.shift_left(left, shift)
        right = self.shift_left(right, shift)
        combined = left + right
        round_key = self.PC_2(combined)
        round_keys.append(round_key)

return round_keys
```

2.2 Encryption and Decryption using DES

The `DES` class is responsible for the core encryption and decryption functionalities of the DES algorithm. The implementation utilizes the 16 round keys generated by the `KeyGenerator` to perform a sequence of transformations on the plaintext or ciphertext. Below are the key components of the `DES` class:

- `permute`: This is a general-purpose method that performs permutations on a binary string based on a given table. This is used for both the Initial Permutation (IP) and the Final Permutation (FP), as well as during specific steps within each DES round.
- `s_box_substitution`: This method performs the S-box substitution step within each DES round, where the expanded 48-bit data is divided into 8 blocks, and each block is substituted using the corresponding S-box from the lookup table.
- `round`: The DES round function that applies the Expansion (E), XOR, S-box substitution, and Permutation (P) transformations to the data, combining it with a subkey in each round.
- `feistel`: This method represents one Feistel round, where the output of the `round` method is XORed with the left half of the data.
- `process_block`: This method performs the complete encryption or decryption of a single 8-byte block using the Feistel network over 16 rounds.

The `process_block` method

```
def process_block(self, block: bytes, mode: str) -> bytes:
    if len(block) != 8:
        raise ValueError("Block size must be exactly 8 bytes.")

    # Convert block to binary string
    block_int = int.from_bytes(block, byteorder='big')
    block_bin = int_to_bin(block_int, 64)

    # Initial Permutation
    permuted = self.permute(block_bin, lookup_tables['ip'])
    left = permuted[:32]
    right = permuted[32:]

    # 16 DES rounds
    if mode == 'encryption':
        subkeys = self.subkeys
    elif mode == 'decryption':
        subkeys = self.subkeys[::-1]
    else:
        raise ValueError("Mode must be 'encryption' or 'decryption'")

    for subkey in subkeys:
        left, right = self.feistel(left, right, subkey)

    # Preoutput (Right + Left)
    preoutput = right + left

    # Final Permutation
    final_permutation = self.permute(preoutput, lookup_tables['fp'])
    final_int = bin_to_int(final_permutation)
    final_bytes = final_int.to_bytes(8, byteorder='big')

return final_bytes
```

- `DES_algorithm`: The main DES algorithm that encrypts or decrypts the input data in 8-byte blocks. It uses PKCS#7 padding for encryption and removes it after decryption.

The `DES_algorithm` method

```
def DES_algorithm(self, text: bytes, mode: str) -> bytes:
    if mode == 'encryption':
        # Apply PKCS#7 padding
        padded_text = pad(text)
        # Split into 8-byte blocks
        blocks = [padded_text[i:i+8] for i in range(0, len(padded_text), 8)]
    elif mode == 'decryption':
        # For decryption, input should be multiple of 8 bytes
        if len(text) % 8 != 0:
            raise ValueError("Ciphertext length must be a multiple of 8 bytes.")
        blocks = [text[i:i+8] for i in range(0, len(text), 8)]
    else:
        raise ValueError("Mode must be 'encryption' or 'decryption'")

    processed_blocks = []
    for block in blocks:
        processed_block = self.process_block(block, mode)
        processed_blocks.append(processed_block)

    if mode == 'decryption':
        # Concatenate decrypted blocks
        decrypted_data = b''.join(processed_blocks)
        # Remove padding
        try:
            unpadded_data = unpad(decrypted_data)
            return unpadded_data
        except ValueError as e:
            raise ValueError(f"Padding error during decryption: {e}")
    else:
        # For encryption, concatenate encrypted blocks
return b''.join(processed_blocks)
```

- `encrypt`: This method encrypts a plaintext string and returns the ciphertext as a hexadecimal string.
- `decrypt`: This method decrypts a hexadecimal ciphertext string and returns the resulting plaintext string.

The implementation is structured to be easily extended or modified if additional functionality is needed. Each method and class is carefully designed to encapsulate specific tasks within the DES encryption and decryption process, allowing for clear separation of concerns and high readability.

3 User Interface

The user interface is implemented with Gradio, a Python library for quickly building web UI for demonstrating algorithms.

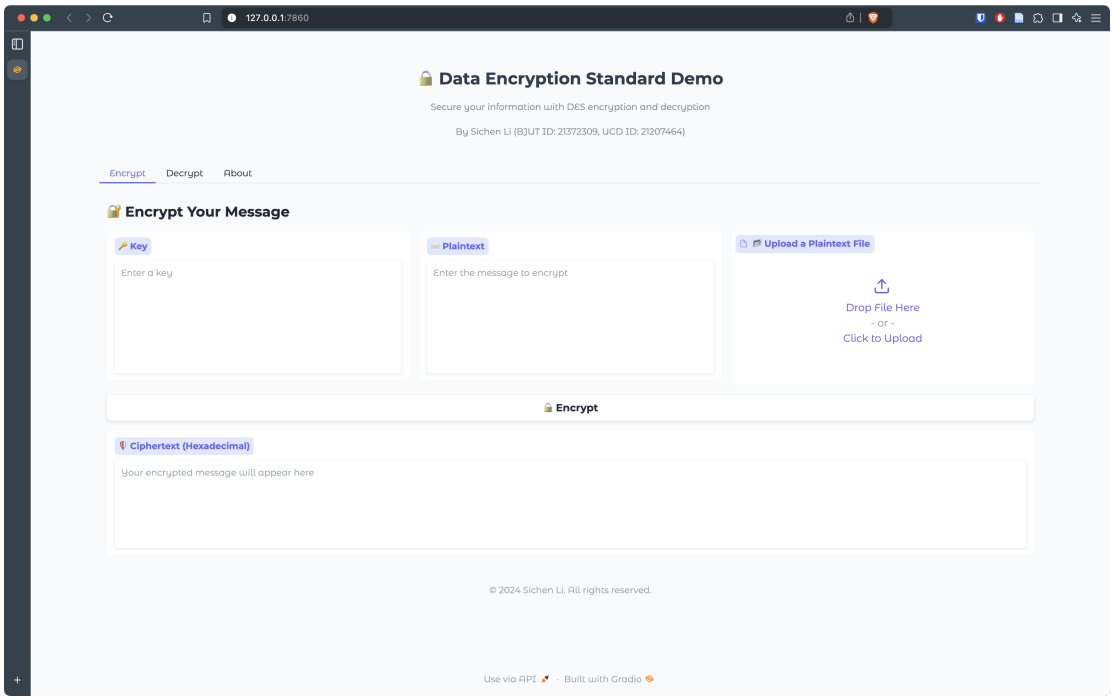

Figure 1: User Interface for the Project

3.1 Prerequisites

Before using the user interface, the user needs to install the required libraries by running the following command under the project directory:

Installing Required Libraries

```
pip install -r requirements.txt
```

The user can then run the user interface by executing the following command:

Running the User Interface

```
python ui.py
```

3.2 Encrypting Data

The user can enter the key and plaintext in the input boxes and click the `Encrypt` button to generate the ciphertext.

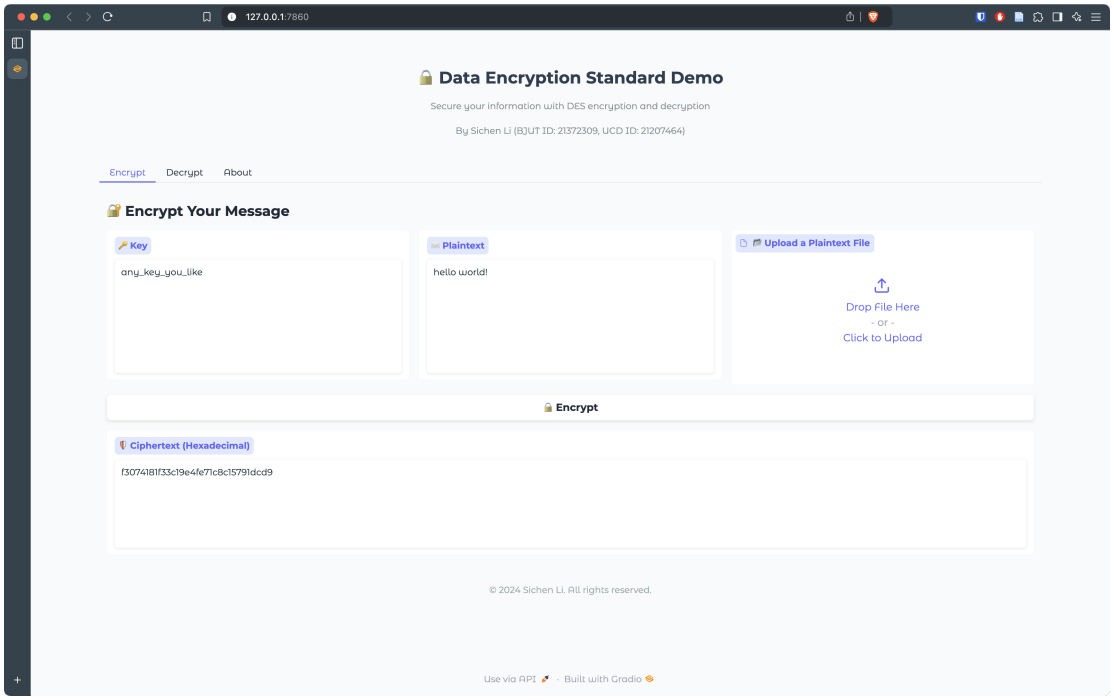

Figure 2: Encrypting Data from Plaintext

In addition to simply entering the key and plaintext, the user can also upload a text file containing the plaintext to encrypt the content of the file.

For example, the user can upload a text file containing the following plaintext:

Figure 3: Text File to Encrypt

After clicking the `Encrypt` button, the user will get the ciphertext of the file.

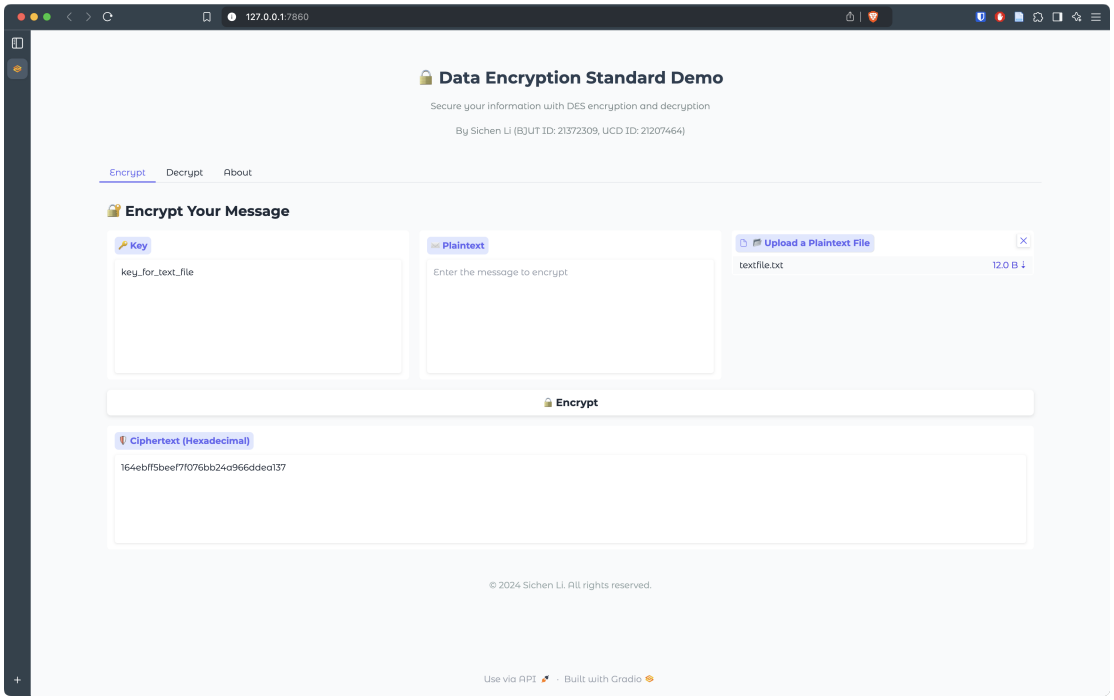

Figure 4: Encrypting Data from a Text File

3.3 Decrypting Data

The user can enter the key and ciphertext in the input boxes and click the `Decrypt` button to generate the plaintext.

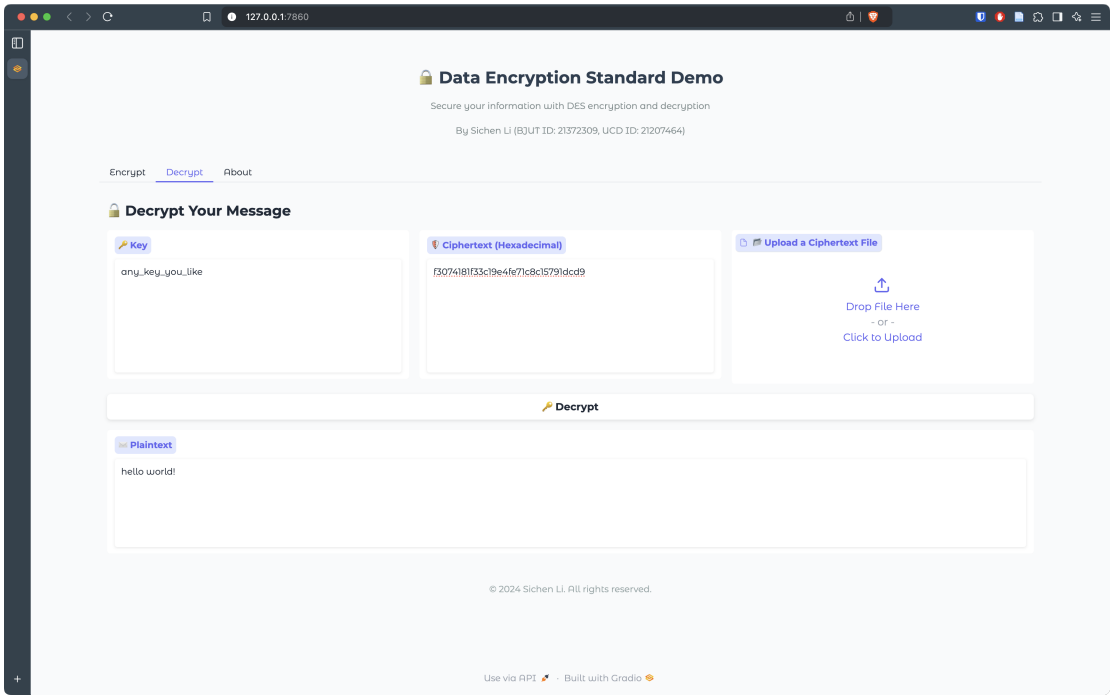

Figure 5: Decrypting Data from Ciphertext

Similar to encryption, the user can also upload a text file containing the ciphertext to decrypt the content of the file.

Figure 6: File to Decrypt

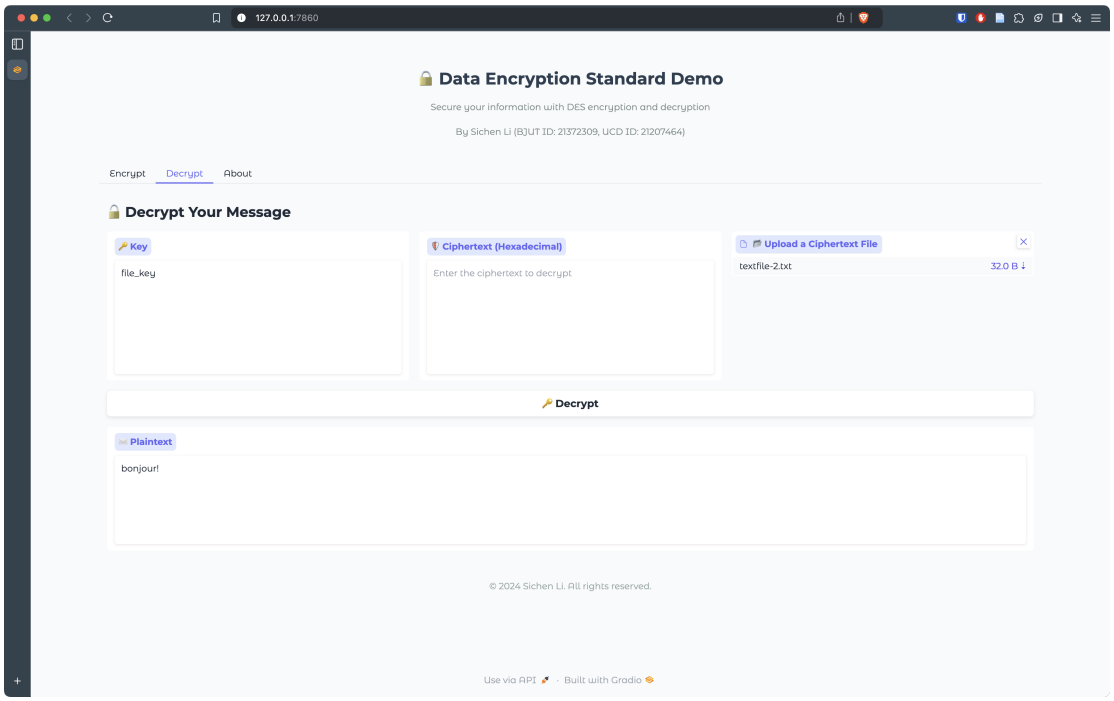

Figure 7: Decrypting Data from a Text File