University College Dublin

COMP3025J Security & Privacy
Project 1 Report

Supervisor: xxx

Student:
xxx Qixxx
UCD Number: 22xxx
BDIC Number: 22xxx

Contents

1 Introduction

DES is a symmetric-key block cipher standardized in FIPS PUB 46-3 with block size 64 bits and 16 rounds of Feistel structure. DES remains a canonical algorithm for teaching cryptography, despite its short key size (56 effective bits) and practical obsolescence for modern security applications [1, 2]. This project implements both encryption and decryption parts of DES according to the FIPS specification and builds a PyQt5-based GUI demonstrating text and file modes, as required by the project description. The goal of this project is to:

1. Implement both DES encryption and decryption for arbitrary-length plaintext/ciphertext (i.e., by processing in 64-bit blocks and padding as required).
2. Provide a user-friendly interface allowing users to supply plaintext/ciphertext via text input or file system selection.
3. Ensure encryption and decryption are independent operations (each invoked with an explicit key).
4. Include robustness measures (error handling, informative messages, and file-mode workflows including compression/decompression where applicable).

2 Background and Standards

The Data Encryption Standard (DES) is specified in the FIPS PUB 46-3 (reaffirmed Oct 25, 1999). The standard describes the initial and final permutations (IP, FP), the expansion permutation (E), the S-boxes, the P permutation, and the key schedule producing 16 subkeys. The Feistel structure makes encryption and decryption symmetric: decryption uses the same round function but applies the 16 subkeys in reverse order [1, 2].

3 Design and Architecture

3.1 High-level architecture

The project is organized into three logical layers:

- **Algorithm layer**: DES algorithm implementation (bit-level operations, permutations, S-boxes, key schedule). This part is implemented in `des_alg.py`.
- **Application layer**: orchestration code that splits long inputs into 64-bit blocks, pads the last block, converts between bytes and bit arrays, and manages file IO (compression for file mode). This part is implemented in `main.py`.
- **UI layer**: PyQt5 GUI providing text input, file selection, copy/clear buttons, Encrypt/Decrypt triggers, status messages and result display. This part is implemented in `main.py`.

3.2 Project files

- **data.py**: Contains the DES constant tables: permutation indices (IP, IP_INV, E, P), S-box definitions (S_BOX), and other fixed arrays. This file keeps large static data separate from algorithm logic.
- **des_alg.py**: Implements the DES primitive functions:
  - low-level helpers: `permutation()`, `xor()`, bit conversion helpers.
  - key schedule: `calculate_keys(key_bits)` producing 16 subkeys.

    - core functions: `encryption(input_bits, key_bits)` and `decryption(cipher_bits, key_bits)`.

  The design keeps algorithmic correctness isolated from UI and IO.

- **main.py**: The application and GUI layer built on PyQt5:
    - implements the PyQt5 window, layout, buttons and events,
    - implements `mainDES()` which orchestrates conversions between bytes and bit arrays, calls into `des_alg.py` per block, and handles file-mode compression/decompression,
    - handles clipboard actions, status messages, error dialogs and delayed-close behavior.

The runtime combination is straightforward: the UI (`main.py`) imports des_alg and data tables and calls the encryption/decryption functions for each 8-byte block, then displays or writes the resulting bytes.

4 Algorithm Implementation

The DES algorithm is a classical symmetric-key algorithm, where the same key is used for both encryption and decryption. In this project, the implementation of the DES algorithm is encapsulated within the `des_algorithm.py` file, which includes both the encryption and decryption processes, as well as shared utility functions. The `data.py` file contains all the data tables used in the DES algorithm. In the snippets below, we retain the original function names so readers can easily map them back to the corresponding files.

4.1 Utility Functions

Several utility functions are employed in the DES algorithm to facilitate various operations:

- permutation and permutation_reverse: These functions handle the data blocks' initial and final permutations (IP and IP reverse).
- calculate_keys: This function generates the subkeys used in each round of the DES process. More explicitly, you can understand the process by reading the code:

```
def calculate_keys(key: list) -> list:
    keys = []
    CD = permutation(key, PC_1)  # Use PC_1 substitution to output
        56 bits.
    C = CD[:28]
    D = CD[28:]
    for i in range(16):
        # Perform a left circular shift on the C and D halves of
            the key
        C = left_shift(C, SHIFTS[i])
        D = left_shift(D, SHIFTS[i])
        # Concatenate C and D, then permute using PC_2 to generate
            the subkey
        K = permutation(C + D, PC_2)
        # Append the generated subkey to the keys list
        keys.append(K)
    return keys
```

Listing 1: `calculate_keys()` from `des_alg.py`

- xor: Performs bitwise XOR operations, crucial for mixing subkeys and data blocks.

- s_box_compression: Executes the S-box compression, transforming expanded blocks into smaller, permuted blocks.

4.2 Encryption Logic

Encryption takes a 64-bit key and plaintext and produces a 64-bit ciphertext. The steps involved are as follows:

1. Apply the initial permutation (IP) to the plaintext block.
2. Split the result of the IP into two halves: the left 32 bits (L) and the right 32 bits (R).
3. Compute the subkeys for each round using the provided key.
4. For each of the 16 rounds, calculate the new values of L and R using the function f, which includes expansion, substitution, and permutation operations.
5. After completing all rounds, concatenate the final L and R.
6. Apply the final permutation (IP reverse) to the concatenated result to obtain the ciphertext.

```
def encryption(plaintext: list, key: list) -> list:
    # Initial permutation
    combined_text = permutation(plaintext, IP)
    L = combined_text[:32]
    R = combined_text[32:]
    # Generate 16 subkeys
    keys = calculate_keys(key)
    # 16 rounds of encryption
    for i in range(16):
        # Expansion permutation to 48 bits
        expanded_R = permutation(R, E)
        # XOR with the current key
        xor_R = xor(expanded_R, keys[i])
        # S-box compression to 32 bits
        compressed_R = s_box_compression(xor_R, S_BOX)
        # P-box permutation
        permuted_R = permutation(compressed_R, P)
        # XOR with L to get the new R
        final_R = xor(permuted_R, L)
        # Prepare for the next round
        L = R
        R = final_R
    # Combine R and L
    final_permuted = permutation(R + L, IP_INV)
    return final_permuted
```

Listing 2: `encryption()` from `des_alg.py`

4.3 Decryption Logic

The decryption process inputs a 64-bit key and ciphertext and outputs the 64-bit plaintext. Basically the logic is symmetric to the encryption's logic, so I'll not state again here.

```
def decryption(ciphertext: list, key: list) -> list:
    # Initial permutation (inverse)
    combined_text = permutation_reverse(ciphertext, IP_INV)
    R = combined_text[:32]
    L = combined_text[32:]
    # Generate 16 subkeys and reverse the order for decryption
    keys = calculate_keys(key)[::-1]
    # 16 rounds of the DES algorithm
    for i in range(16):
        # Swap L and R for decryption
        temp = L
        final_R = R
        R = temp
        # Expansion permutation to 48 bits
        expanded_R = permutation(R, E)
        # XOR with the current key
        xor_R = xor(expanded_R, keys[i])
        # S-box compression to 32 bits
        compressed_R = s_box_compression(xor_R, S_BOX)
        # P-box permutation
        permuted_R = permutation(compressed_R, P)
        # XOR with L to get the new L
        L = xor(permuted_R, final_R)
    # Final permutation after recombining L and R
    final_permuted = permutation_reverse(L + R, IP)
    return final_permuted
```

Listing 3: `decryption()` from `des_alg.py`

The above two functions illustrate the key idea: the same round function is used for encryption and decryption; decryption merely applies the subkeys in reverse order. The full code includes the exact permutation tables (IP, IP_INV, E, P) and the S-BOX definitions (in `data.py`), as well as helper functions like `permutation()`, `xor()`, `s_box_compression()` and `calculate_keys()`.

4.4 File-mode I/O and GUI glue code

The UI orchestrates whether input should be treated as text or file bytes; here are key snippets from `main.py` that show how the application handles modes and file workflows.

```
def mainDES(self):
    key = self.key_edit.text()
    text = self.input_edit.toPlainText()
    self.result = bytes()

    # validate input
    if not key or not text:
        QMessageBox.warning(self, 'Warning', 'You␣have␣not␣entered␣key␣
            or␣text', QMessageBox.Ok)
        return

    # file-mode: read file bytes (either for encrypt or decrypt)
    if self.use_file:
        if self.mode == 'encrypt':
            self.compress_file()        # create a zip from selected file
            file_name = self.zip_file_name
        else: # decrypt
            file_name = self.file_path
        with open(file_name, 'rb') as f:
            text = f.read()
    else:
        # text-mode: for encrypt -> encode, for decrypt -> interpret raw
            bytes
        if self.mode == 'encrypt':
            text = text.encode()
        else:
            # convert characters to their byte values
            t = bytes()
            for ch in text: t += ord(ch).to_bytes(1, 'little')
            text = t

    # pad key to 8 bytes, pad text to multiple of 8 bytes...
    # convert to bit arrays, call encryption/decryption per 8-byte block
    # accumulate self.result
    # file-mode: for decrypt, write decrypted bytes to temp zip and
      uncompress
    # text-mode: set output_edit to decoded
```

Listing 4: `mainDES()` from `main.py`

And for file generating and processing, the logic goes like this:

```
def save_file(self):
    encrypted_file_path = self.file_path.replace('.', '(encrypted).')
    with open(encrypted_file_path, 'wb') as f:
        f.write(self.result)
    self.output_edit.setText('File␣saved:␣' + encrypted_file_path)

def compress_file(self):
    base_file_name = self.file_path.split('/')[-1]
    zip_file_name = f"{base_file_name.split('.')[0]}.zip"
    internal_file_name = f"encrypted_{base_file_name}"
    self.zip_file_name = zip_file_name
    self.file_name = internal_file_name
    with zipfile.ZipFile(zip_file_name, 'w', zipfile.ZIP_BZIP2) as zf:
        zf.write(self.file_path, internal_file_name)

def uncompress_file(self):
    file_path = '/'.join(self.zip_file_name.split("/")[:-1]) + '/'
    try:
        with zipfile.ZipFile(self.zip_file_name, 'r') as zf:
            zf.extractall(file_path)
        self.output_edit.setText(f"File␣decryption␣results␣have␣been␣
           saved␣to␣{file_path␣+␣self.file_name}")
    except zipfile.BadZipFile:
        self.output_edit.setText("Bad␣zip␣file␣or␣corrupted␣archive")
```

Listing 5: File write/decompress helpers (simplified)

4.5 Data tables and constants

`data.py` module stores all the necessary tables required by the DES algorithm, including IP and IP reverse permutation tables, Key permutation table, Expansion permutation table, eight S-boxes, and P permutation table.

5 User guide

Below is a step-by-step walkthrough of the user interactions to demonstrate the tool.

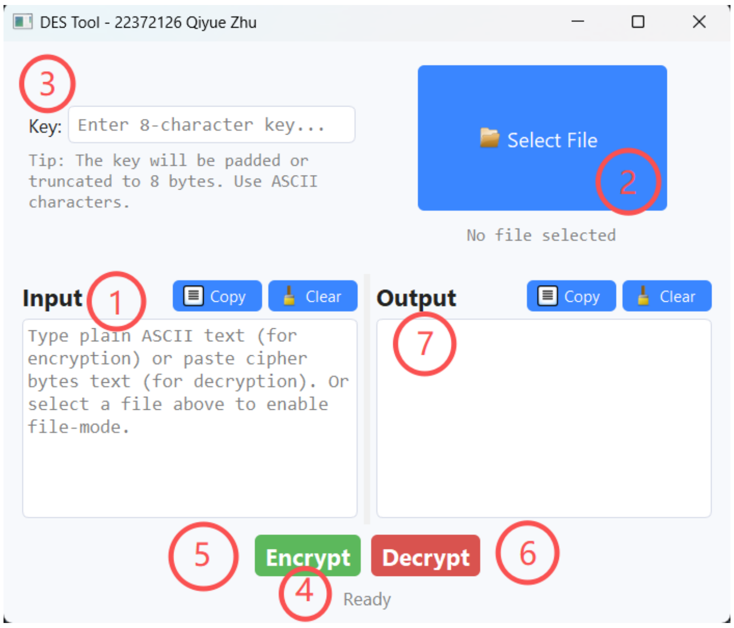
5.1 Startup

1. Run the program:

```
python main.py
```

   As shown in Figure 1, the main window opens with:

   - Top-left (3): **key** input box and a short hint underneath.
   - Top-right (2): large **Select File** button and filename display.
   - Middle-left (1): Input text area with **Copy** and **Clear** buttons.
   - Middle-right (7): Output area (read-only) with **Copy** and **Clear** buttons.
   - Bottom (4), (5) & (6): **Encrypt** (5) and **Decrypt** (6) buttons, and a **status label** (4).


DES Tool - 22372126 Qiyue Zhu

Key: Enter 8-character key...

Tip: The key will be padded or truncated to 8 bytes. Use ASCII characters.

No file selected

Type plain ASCII text (for encryption) or paste cipher bytes text (for decryption). Or select a file above to enable file-mode.

Ready

Figure 1: Main application window

5.2 Text mode encryption/decryption

1. Type (or paste) plaintext into the **Input** box (1).

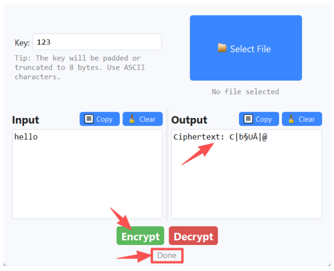
2. Enter an 8-character key (or any string and the program will pad/truncate to 8 bytes) in the Key input area (3). The tip is the format hint for users.

3. The status (4) is ready, meaning the program is ready to compile.

4. Click **Encrypt** (5):

   - The app will perform block-by-block DES encryption and show the ciphertext in the **Output** area (7). Note that for text-mode encryption, output may contain non-printable characters and the app prints them as-is, as shown in Figure 2.

5. To reverse, copy the ciphertext back to the **Input** area (1). You can use the "copy" button to copy content or use the "clear" button to clear the area. Ensure the same key is entered, and click **Decrypt** (6). The **Output** (7) will show plaintext, as shown in Figure 3.



Key: 123

Tip: The key will be padded or truncated to 8 bytes. Use ASCII characters.

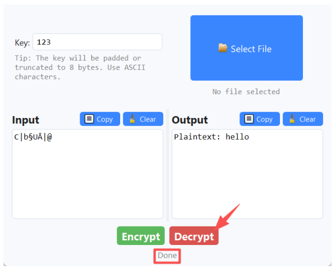
No file selected

hello

Ciphertext: C|b§UÂ|@

Done

Figure 2: Encrypt



Key: 123

Tip: The key will be padded or truncated to 8 bytes. Use ASCII characters.

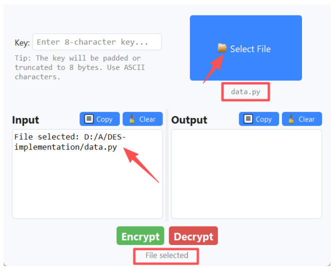
No file selected

C|b§UÂ|@

Plaintext: hello

Done

Figure 3: Decrypt



Key: Enter 8-character key...

Tip: The key will be padded or truncated to 8 bytes. Use ASCII characters.

data.py

File selected: D:/A/DES-implementation/data.py

File selected

Figure 4: Upload File

5.3 File mode encryption/decryption

1. Click the large **Select File** button (2) and choose a file from disk. The filename will display under the button and in the **Input** area (1), as shown in Figure 4.

2. Enter a key (3).

3. Click **Encrypt** (5):

    - The tool will compress the selected file into a ZIP archive and then encrypt the ZIP bytes block-by-block. The encrypted bytes are written to disk as `encrypted_<basename>.zip`.
    - A non-modal dialog informs the user that saving has completed and the app may auto-close after a short delay.

4. For decryption:

    - Click **Select File** (2) and pick the encrypted file (the one produced by the program).
    - Enter the same key used to encrypt in the key input area (3).
    - Click **Decrypt** (6): the app will decrypt bytes, write them to a temporary zip file `decrypted_<basename>.zip`, and then extract its contents to the directory.

5.4 Tips

- Use the **Copy** button on the input area to copy input or the one on the output area to copy output. The status will be reflected on the status label as shown in Figure 5 and 6.

- Use the **Clear** button on the input area to clear input and reset the selected file state (this returns the UI to initial 'Ready' state).

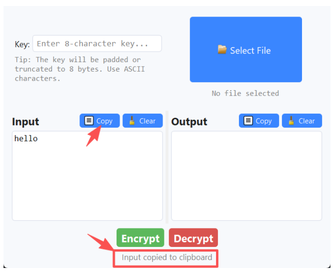
- The separation bar can be moved if one area need to use more space, as shown in Figure 7.

- Always use the same key for encryption and decryption.

6 Problems Encountered and Solutions

6.1 Empty Key Entered

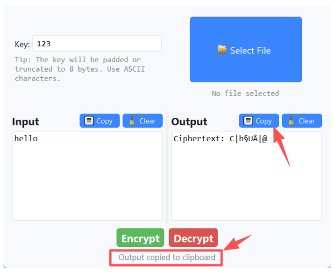
Problem: User user clicks the**Encrypt** button (5) or **Decrypt** button (6) without entering the key in the key area (3).
Solution: The system will pop up an alert to user to inform them as shown in Figure 8. The user can continue after clicking "OK".



hello

Input copied to clipboard

Figure 5: Copy Input

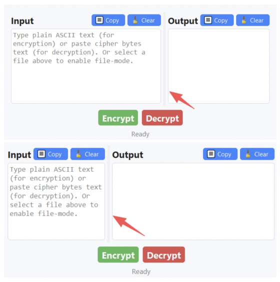

Key:
Tip: The key will be padded or truncated to 8 bytes. Use ASCII characters.
Select File
No file selected
Input
Copy
Clear
Output
Copy
Clear
Encrypt
Decrypt
123

hello

Ciphertext: C|b§UA|@

Output copied to clipboard

Figure 6: Copy Output

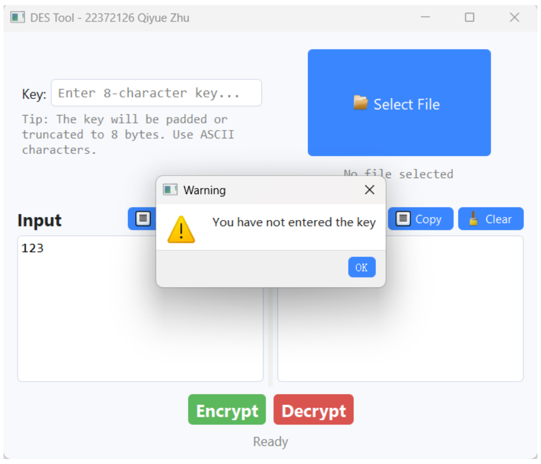


Type plain ASCII text (for encryption) or paste cipher bytes text (for decryption). Or select a file above to enable file-mode.

Figure 7: Movable Separate Bar

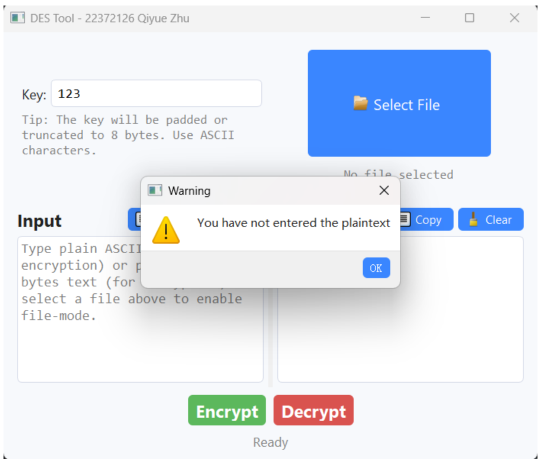
6.2 Empty Text Entered

Problem: User user clicks the**Encrypt** button (5) or **Decrypt** button (6) without entering the plain text in the input area (1) or select a file (2).
Solution: The system will pop up an alert to user to inform them as shown in Figure 9. The user can continue after clicking "OK".


DES Tool - 22372126 Qiyue Zhu
Key:
Enter 8-character key...
Tip: The key will be padded or truncated to 8 bytes. Use ASCII characters.
Select File
No file selected
Input
Copy
Clear
Warning
OK
Encrypt
Decrypt
Ready
123

You have not entered the key

Figure 8: Empty Key


DES Tool - 22372126 Qiyue Zhu
Key:
Tip: The key will be padded or truncated to 8 bytes. Use ASCII characters.
Select File
No file selected
Input
Copy
Clear
Warning
OK
Encrypt
Decrypt
Ready
123

You have not entered the plaintext

Type plain ASCII encryption) or p bytes text (for select a file above to enable file-mode.

Figure 9: Empty Text

7 Conclusion and Future Work

This project implements DES encryption and decryption according to FIPS PUB 46-3 and provides a practical GUI to demonstrate the algorithm on both text and files. The implementation emphasizes correctness, usability and robustness.

During the completion of this project on the implementation of the DES algorithm, several challenges were encountered. These challenges provided significant learning opportunities and highlighted areas for potential improvement.

- Character Encoding: Initially, input and output handling required effective encoding to correctly process all types of characters, including Chinese characters. Ensuring accurate and consistent character encoding across different systems and languages is crucial for the integrity of data processing.
- EncryptionKey Length: The DES algorithm specifies a fixed key length; however, the project requirements allowed users to enter keys of any length. This flexibility can lead to security vulnerabilities and operational errors. Implementing strict validation to enforce the correct key length as per DES standards will be necessary in the future.

- Exception Handling and Testing of Boundary Cases: Currently, the system only prompts the user when necessary fields are not entered. However, there are many other boundary conditions that require robust exception handling to maintain the application's reliability and security.

For future work, the primary goal is to refine the existing application and integrate more secure algorithms, such as AES or Triple DES. Extensive testing and enhanced handling of boundary cases are planned, which will improve security and ensure that the application remains in sync with current cryptographic standards.

Additionally, optimizing the performance of the encryption and decryption processes and improving the user interface will significantly enhance the overall user experience.

References

[1] National Institute of Standards and Technology (NIST), *FIPS PUB 46-3: Data Encryption Standard (DES)*, Reaffirmed October 25, 1999. Available: https://csrc.nist.gov/pubs/fips/46-3/final and archived PDF at https://csrc.nist.gov/CSRC/media/Publications/fips/46/3/archive/1999-10-25/documents/fips46-3.pdf.

[2] Data Encryption Standard (DES), Wikipedia, consulted 2025. https://en.wikipedia.org/wiki/Data_Encryption_Standard.

[3] NIST CSRC: FIPS 46-3 page. https://csrc.nist.gov/pubs/fips/46-3/final.