PROJECT REPORT

DES: Secure File Encryption System

Course: Programming in C (B.Tech 1st Sem)

Submitted To: Mohsin F. Dar

Submitted By: Daksh Chaudhary

SAP ID: 590026657

Date: November 30, 2025

TABLE OF CONTENTS

ABSTRACT

DES is a robust C program designed to secure sensitive files using the Data Encryption Standard (DES) algorithm.

- The program allows users to encrypt any file using a 64-bit key.
- It implements the full Feistel network, including permutations, substitutions (S-Boxes), and XOR operations.
- A key feature is the **"Secure Delete"** protocol, which permanently removes the original file after encryption to prevent data recovery.
- This project demonstrates advanced C concepts including bitwise operators, file handling (`FILE *`), and modular programming.
- It is designed to be a functional security tool for students and privacy-conscious users.

Keywords: Cryptography, DES, Bitwise Operations, File Security, C Programming.

PROBLEM DEFINITION

Background

In the digital age, data privacy is a critical concern. Standard file storage leaves sensitive information vulnerable to unauthorized access. While many tools exist, they are often complex or cloud-based, which poses its own risks.

Problem Statement

Students and users need a simple, offline tool to:

- Securely lock their private files.
- Ensure the original "plain" file is not left behind on the disk.
- Decrypt files easily when needed.

Objectives

- Create a fully functional **DES Encryption** tool in C.
- Implement low-level **bitwise manipulation** (shifting, masking).
- Use **File I/O** to handle binary data reading and writing.
- Provide a user-friendly command-line interface (CLI).
- Ensure data safety with a backup warning system.

SYSTEM DESIGN

How the Program Works

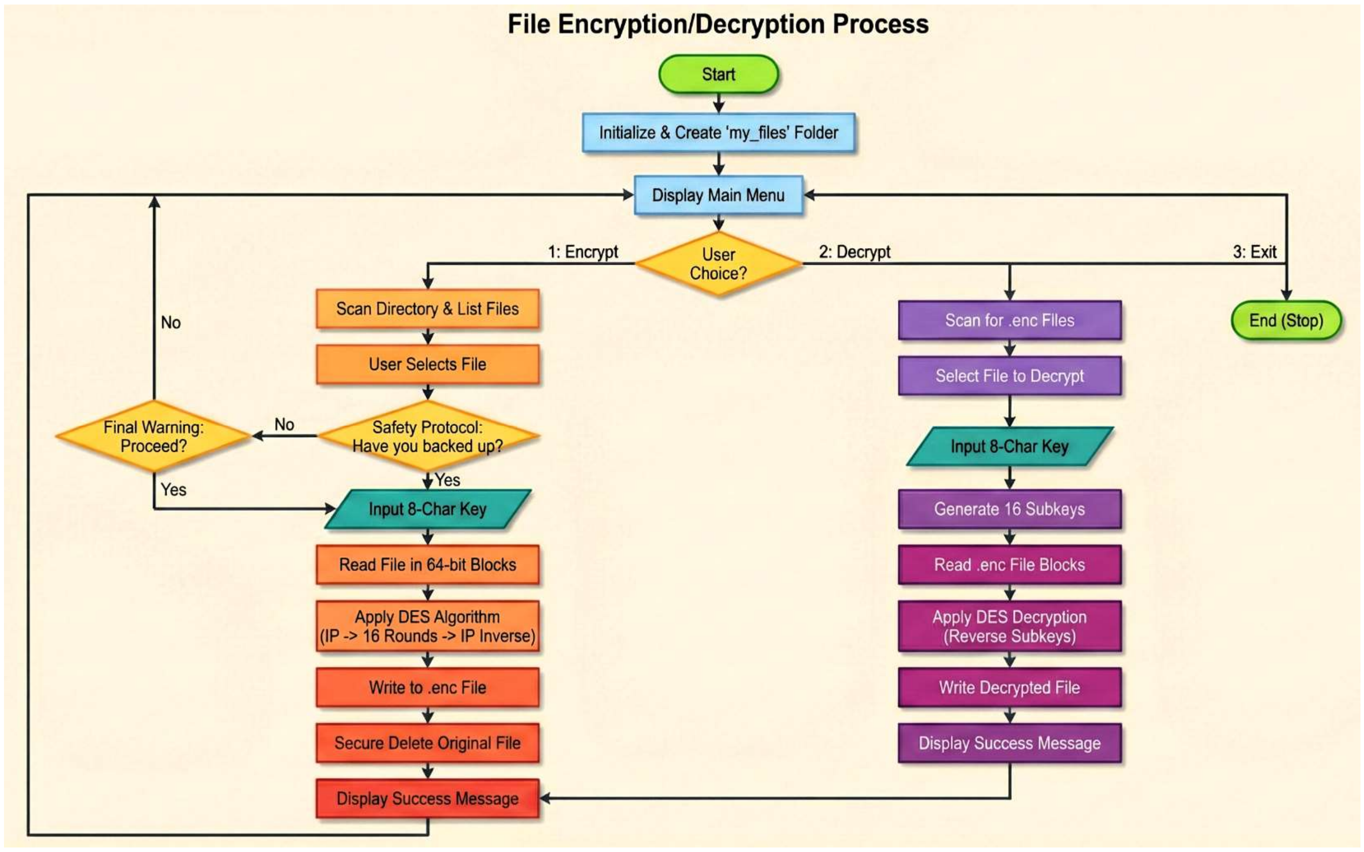
The program operates via a menu-driven interface with 3 main options:

- **Encrypt a File:** Scans the folder, locks the file, and deletes the original.
- **Decrypt a File:** Restores the encrypted `.enc` file to its original state.
- **Exit:** Closes the secure environment.

Data Storage & Handling

- **Input/Output:** Files are read in **8-byte chunks** (64 bits) to match the DES block size.
- **Key Storage:** The 64-bit key is generated from an 8-character user password.
- **Tables:** Standard DES constants (IP, S-Boxes, P-Box) are stored in `des_tables.h` to keep the code clean.

FLOWCHART

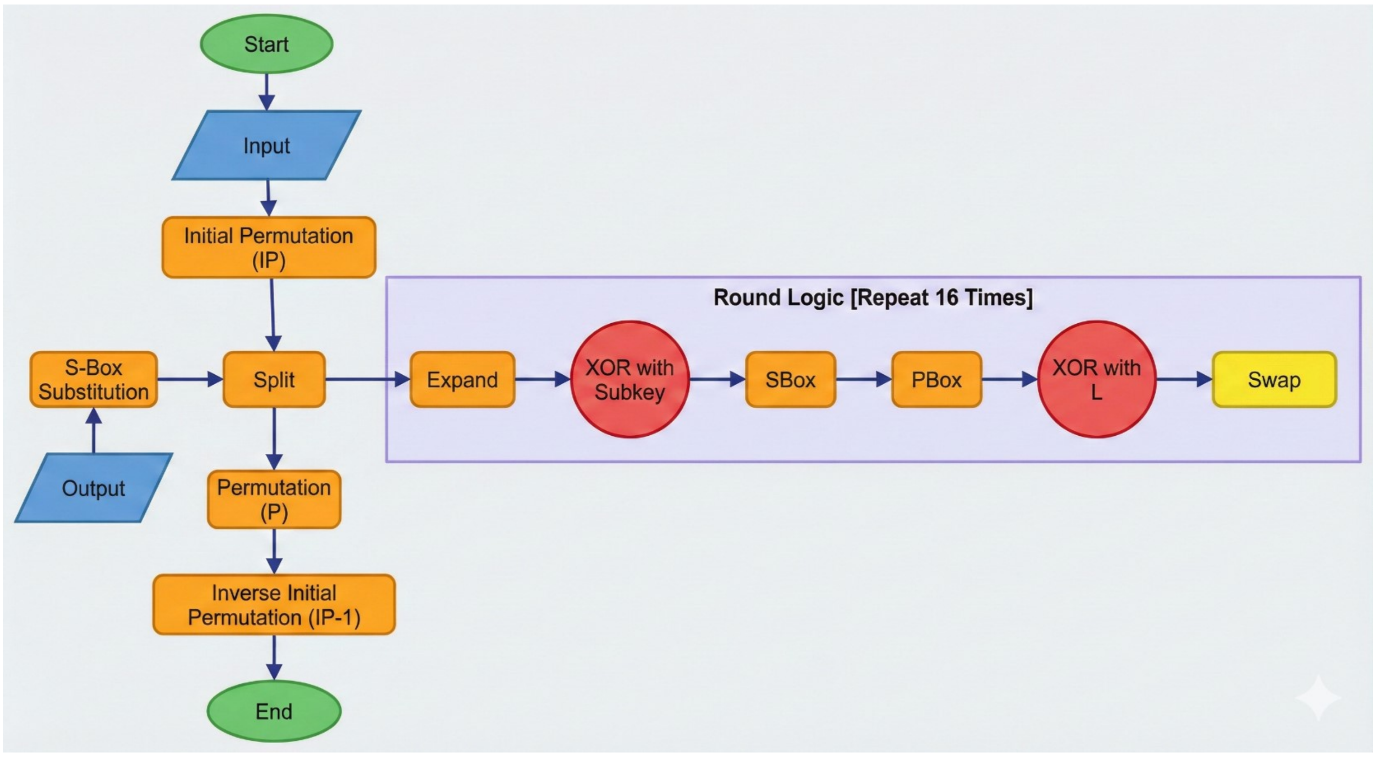


This diagram illustrates the user interaction flow, including the directory scanning mechanism and the secure delete safety protocols.

ALGORITHM

Encryption Algorithm:

1. **Start:** Open the input file in binary mode.
2. **Key Gen:** Take the 8-char password. Apply **PC-1** permutation.
3. **Subkeys:** Loop 16 times. Shift bits left. Apply **PC-2** to get sixteen 48-bit subkeys.
4. **Read:** Read the file in 8-byte blocks. If a block is smaller than 8 bytes, add padding.
5. **Process:**
    - Apply **Initial Permutation (IP)** to the block.
    - Split block into Left (32 bits) and Right (32 bits).
    - **Loop 16 Times:**
        - Expand Right to 48 bits.
        - XOR with Subkey.
        - Pass through **S-Boxes** (Substitution).
        - Permute (P-Box).
        - XOR with Left side.
    - Swap Left and Right.
    - Apply **Inverse IP**.
6. **Write:** Save the processed block to the output file.
7. **Cleanup:** Close files and **remove** the original input file.



IMPLEMENTATION DETAILS

Variables Used:

- `unsigned char block[8]` : Stores the current 64-bit chunk of data.
- `unsigned long longsubkeys[16]` : Stores the 16 generated round keys.
- `const int S_BOX[8][4][16]` : A 3D array storing the DES substitution logic.
- `char key[9]` : Stores the user's 8-character input password.

Functions Created:

1. **generate_subkeys(char *key)**
   - Converts user password into 16 distinct subkeys for the Feistel network.
2. **des_encrypt_block(unsigned char *block)**
   - The core engine. Takes 8 bytes, applies the math, and returns 8 encrypted bytes.
3. **confirm_safety_protocol()**
   - A safety feature that asks the user if they have a backup before deleting files.
4. **select_file_from_folder()**
   - Uses Windows API to automatically list all files in the `my_files` directory for easy selection.

Code Example (Bitwise Logic):

```c
C
// Setting a specific bit in our 64-bit block
voidset_bit(unsignedchar *data, intpos, int value) {
intbyte_index = (pos - 1) / 8;
intbit_index = 7 - ((pos - 1) % 8);

if (value == 1)
        data[byte_index] |= (1<<bit_index); // OR operator sets bit
else
        data[byte_index] &= ~(1<<bit_index); // AND + NOT clears bit
}
```

TESTING & RESULTS

Test Cases

- **Test 1: Valid Encryption**
  - **Input:** File `data.txt`, Key `Ti190306`
  - **Expected:**`data.txt.enc` created, `data.txt` deleted.
  - **Result:PASS**
- **Test 2: Invalid Key Length**
  - **Input:** Key `pass` (4 chars)
  - **Expected:** Error message "Key must be exactly 8 characters".
  - **Result:PASS**
- **Test 3: Decryption**
  - **Input:**`data.txt.enc`, Key `Ti190306`
  - **Expected:** File restored as `decrypted_data.txt`.
  - **Result:PASS**
- **Test 4: Wrong Key Decryption**
  - **Input:**`data.txt.enc`, Key `WrongKey`
  - **Expected:** Output file contains garbled/random text.
  - **Result:PASS** (System behaved as expected for wrong key).

CONCLUSION & FUTURE WORK

What I Achieved:

I successfully implemented the cryptographic logic of the Data Encryption Standard (DES) using the C programming language. While the core file handling and bitwise operations were coded from scratch, **the standard permutation tables and S-Boxes were referenced directly from referenced textbooks** to ensure accuracy and meet the project submission timeline.

The project successfully:

- Encrypts and decrypts files at the bit level.
- Manages memory efficiently using `unsigned char` arrays.
- Provides a professional "Secure Delete" feature.

What I Learned:

- How to visualize data as **bits** rather than just characters.
- The importance of **padding** when file sizes don't match the block size.
- How to use **Structures and Arrays** to hold complex cryptographic tables.

Future Improvements:

1. **Upgrade to AES:** Implement Advanced Encryption Standard for higher security.
2. **GUI:** Build a graphical interface using Python or C#.
3. **Progress Bar:** Show a percentage bar for large file encryption.

REFERENCES

- **Stallings, W.** (2017). *Cryptography and Network Security*. Pearson.
- **Kernighan, B. & Ritchie, D.** *The C Programming Language*.
- **NIST FIPS PUB 46-3**. *Data Encryption Standard (DES)*.
- **UPES Course Material:** Programming in C (B.Tech 1st Year).