Data Encryption and Decryption: Learning C Development through the Implementation of a Simplified Data Encryption Standard (S-DES)

By\ Khalid M. Al-Hawaj

February 10th, 2026

1 Introduction

As computers are becoming more intrusive and prevalent in different facets of life, digital privacy and security is evermore becoming important. To protect the integrity and privacy of data, encryption aims at obscuring the data by performing multiple operations given a secret key. In its essence, data encryption is meant to be computational expensive to prohibit exhaustive search of encryption keys. When designing a compute system, data encryption is a very important operation, as it is used very often and incurs heavy load on the compute system. In non-homomorphic encryption, an encrypted data is decrypted first to perform various operations, and the result must be encrypted before it is stored again. As cryptographic schemes perform intense and elaborate bit-manipulation operations, implementing such schemes pose a great exercise for learning proper development in any language.

The goal of this workshop is to develop a library for simplified data encryption standard (S-DES), which is a simplified encryption scheme of data encryption standard (DES) [7]. The driving program will take as an input two strings: a password and a message. The program will create a key from the password. The program will ask the user for the requested operation: either encryption or decryption. Then, the program will use the implemented library of (S-DES) to either encrypt or decrypt the given message. If the requested cryptographic action is encryption, then the provided message from the user is treated as a string of 8-bit extended ASCII characters. On the other hand, if the requested cryptographic action is decryption, then the provided message is treated as a hexadecimal number (the digits of the hexadecimal number is represented as 8-bit extended ASCII characters), which needs to be parsed correctly. The result of either encryption (a string of hexadecimal digits represented through 8-bit extended ASCII characters) or decryption (a string of 8-bit extended ASCII characters) is finally printed to the user.

The rest of this document is structured as follows: Section 2 discusses the simplified data encryption standard (S-DES), and Section 3 provides an extended discussion on the implementation of the simplified data encryption standard (S-DES) library.

2 Simplified Data Encryption Standard (S-DES)

Simplified data encryption standard (S-DES) is based on the National Bearu of Standards' (NBS) data encryption standard (DES) [7]; DES is a block-cipher symmetric-key encryption scheme proposed by NBS in 1977. Based on the work by Horst Feistel[5, 6], IBM developed DES and submitted it to the NBS as part of a initiative to protect data security and privacy. Before being published in 1976, the NBS modified DES slightly to address some weaknesses. The modifications has sparked a lot of debate since it made DES susceptible to brute-force attacks[3]. However, it has been revealed retrospectively that NBS was making DES more resilient against differential cryptanalysis[1, 2].

Note: S-DES has been developed specifically for this workshop and bears no relation to S-DES proposed by Edward Schaefer[4].

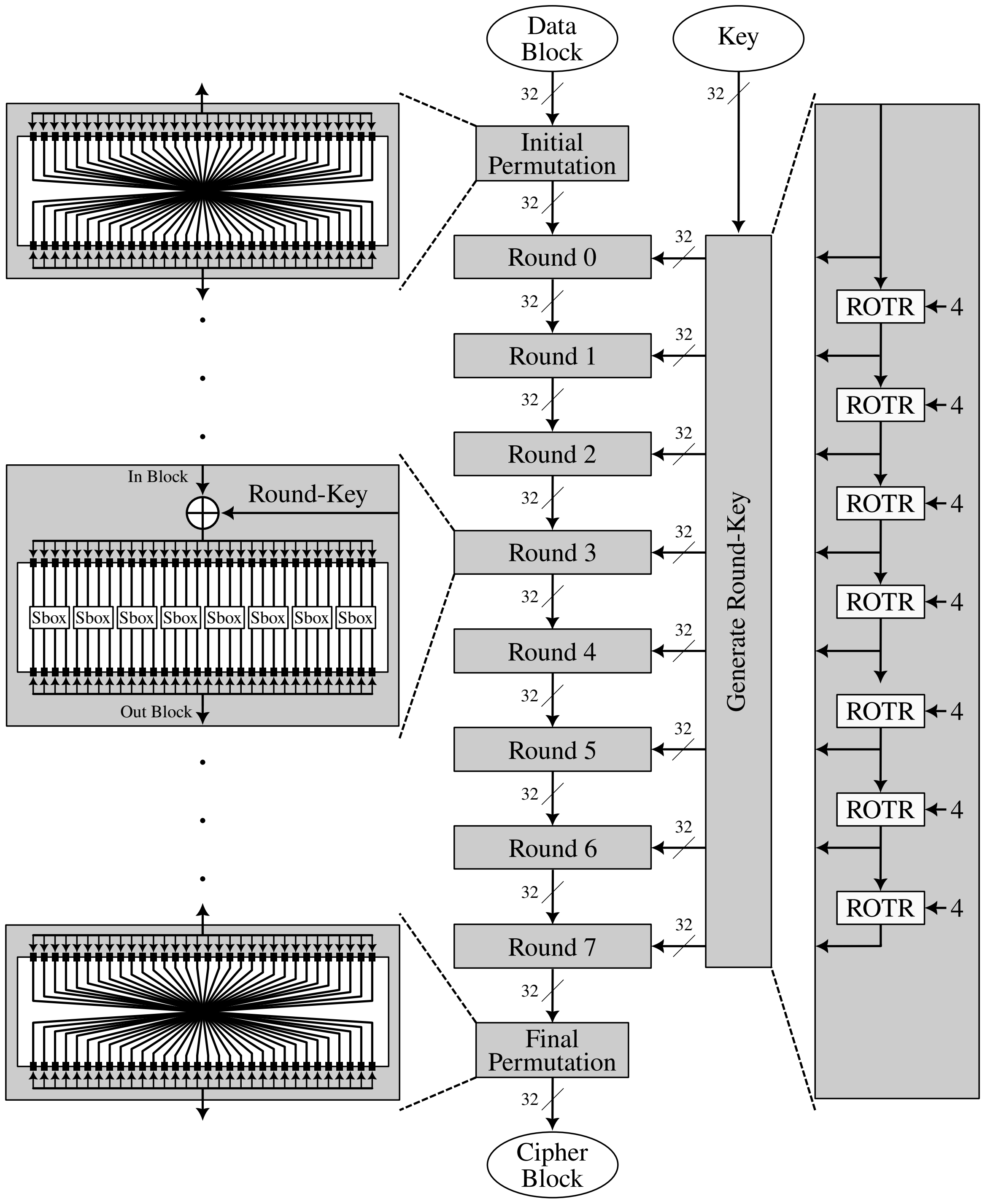

Figure 1: **S-DES Overview**–General overview of the S-DES encryption scheme; to decrypt, the flow goes from the bottom to the top; while to encrypt, the flow goes from the top to the bottom.

The S-DES is composed of four major functionalities: generating key from plain-text password, generating round-key, permutation, and S-DES rounds. Figure 1 shows the overview of the S-DES scheme. Input data to encrypt is divided into blocks of 32-bit, hence the block-cipher nature of S-DES. If the data length is not divisible by 32, the data is padded with zeros. Each block is encrypted with the S-DES scheme shown in Figure 1. To decrypt a cipher data, the S-DES scheme is applied in exact reverse (i.e., starting from the bottom of the scheme towards the top).

	Hexadecimal Digit															
`round_num % 4`	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	2	12	4	1	7	10	11	6	8	5	3	15	13	0	14	9
1	14	11	2	12	4	7	13	1	5	0	15	10	3	9	8	6
2	4	2	1	11	10	13	7	8	15	9	12	5	6	3	0	14
3	11	8	12	7	1	14	2	13	6	15	0	9	10	4	5	3

Table 1: **S-Box Substitution**–This table shows the substitution for each hexadecimal digit (i.e., nibble) of the block based on the round number modulo four.

2.1 Key Generation

S-DES generates a 32-bit key from a given plain-text password. The algorithm for key generation is very simple; for every character of the plain-text password:

1. The 8-bit ASCII of the character is XOR'ed with the 32-bit key (which is initialized to zero).

2. The 32-bit key is rotated rotated-right by 8-bit.

3. The 8-bit ASCII of the character is added to the 32-bit key.

2.2 Generating Round-Key

Each S-DES round will take a 32-bit round-key that is derived from the main 32-bit key. To generate a round-key round x, the main 32-bit key is rotated right by $x \times 4$ times.

2.3 Permutation

Each block of data is first permutated (i.e., the bits of the block are shuffled) before being operated and processed. At the end of the S-DES scheme, a reverse permutation is applied to restore the original order of the bits. In S-DES, the initial permutation is a simple bit-reversal of the block; therefore, the reverse of the initial permutation is the same (i.e., a simple bit-reversal).

2.4 Round

In S-DES, the block will go through consecutive eight rounds of cipher; the input of each round is the output of the previous round. First round takes its input from the initial permutation, while the last round gives its output to the final permutation. In each round, the incoming block is XOR'ed with the corresponding round-key (i.e., for round x, the incoming block is XOR'ed with round-key x). Then, the XOR'ed block is passed through S-boxes (substitution boxes). S-boxes simply replace each 4-bit hexadecimal digit (i.e., a nibble) of the block with a different value according to fixed dictionary. The dictionary is shown in Table 1. There are four dictionaries–first four rounds use different dictionaries, then the subsequent four rounds re-use the four dictionaries. The S-boxes replacement is what makes the S-DES non-linear making it resilient against differential cryptanalysis.

2.5 DES vs. S-DES

The S-DES is a scheme that has been developed to reflect similar functionality and mirror the strength of the original DES[7] algorithm, while simplifying the implementation. Comparing S-DES and DES, couple of differences stand out:

1. block size is 64 bits.

2. The initial permutation is more elaborate than a simple bit reversal.

3. The key is 56 bit rather than 32 bit. The 48-bit round-key is generated by dividing the 56 bits into two 28-bit numbers; the halves are rotated left by either one bit or two bits, respectively. Then a 48-bit round-key is selected from these 28-bit halves—24 bits from one half and 24 bits from the other half. Each round uses different and unique selection of the bits.

4. The round divides the 64-bit block to 32-bit halves. The round-key is XOR'ed with a one half. The half is expanded into 48 bits by duplicating some bits (known as expansion permutation). The S-boxes replace each 6 bits into 4 bits. Instead of using a modulo of the round number to determine the second dimension, the two outer bits (i.e., the most-significant and least-significant bits) are used to address the second dimension. Finally, the outcome of the S-boxes is XOR'ed with the other half of the block.

5. instead of 8 rounds, DES employs 16 rounds.

3 Implementation of S-DES Library

This section will show-case an example implementation of the S-DES scheme in the C programming language. You may use this implementation as a guidance for your MIPS32 implementation.

3.1 Padding and Sanitizing Input

To properly handle inputs, you should read user input as a string into an intermediate buffer. Then, you can process the raw input from the intermediate buffer into a final buffer. Part of the processing is to sanitize (i.e., remove unwanted characters) and pad (i.e., add extra characters). In our case, to sanitize the message string, we replace newline characters with null-characters, thus terminating the string whenever we encounter a newline character. Padding should happen before the encryption routine to avoid any dependencies and assumptions about endianness. Here is the padding implementation.

3.2 Generating 32-bit Main Key

The plain-text password is read from the user. The plain-text password is cleaned and sanitized. Then, the buffer containing the plain-text password is passed as a parameter to a function to generate the 32-bit main key in the following logic:

Algorithm 1 Generating 32-bit Password Hash as Main Key

```
Require: pwd: null-terminated string
  $k \leftarrow 0$
  for each character c in pwd do
    $k \leftarrow k \oplus c$
    $k \leftarrow k$ rotate right by 8
    $k \leftarrow k + c$
  end for
  return $k$
```

3.3 Permutation

Permutation is implemented as a bit-reversal. In the S-DES scheme, both initial and final permutation are implemented the same (since a reverse of a bit-reversal is a bit-reversal!!).

3.4 Generating 32-bit Round-Key

To generating a 32-bit round-key, we require to implement a simple rotate right function. In C, there is no operation to perform rotation; as a result, we have to verbosely implement the rotation using shifts and bit-wise OR operations.

Algorithm 2 Generating 32-bit Round Key

```
Require: key: 32-bit main key
Require: i: round index
  $rot\_amnt \leftarrow (i \% 8) \times 4$
  $k \leftarrow key$ rotate right by $rot\_amnt$
  return $k$
```

3.5 S-DES Encryption

First, we need a top function that would take input padded plain-text and break it into blocks. Each block then can be encrypted independently.

Algorithm 3 Top-Level SDES Encrypt

```
Require: pt_msg: data to be encrypted
Require: pt_sz: number of bytes to encrypt
  for each 32-bit block b in pt_msg do
      $b \leftarrow$ padded(b)
      $c \leftarrow$ sdes_encrypt_block(b)
      append $c$ to ct_msg
  end for
  return ct_msg
```

The encryption routine for S-DES is just the application of initial permutation, followed by eight rounds (with generating their respective round-keys), and then concluding with final permutation:

Algorithm 4 Single block S-DES Encrypt

```
Require: pt_block: block to be encrypted
Require: key: main key for encryption
  block $\leftarrow$ sdes_perm(pt_block)
  for each round in the 8 rounds do
      roundkey $\leftarrow$ sdes_gen_roundkey(round_num, key)
      block $\leftarrow$ sdes_encrypt_round(round_num, block, roundkey)
  end for
  return block
```

The S-DES encryption rounds are simply an XOR followed by S-boxes:

Algorithm 5 Single S-DES Encrypt Round

```
Require: block: block to be encrypted
Require: roundkey: 32-bit roundkey key
Require: i: round index
  block $\leftarrow$ block $\oplus$ roundkey
  block $\leftarrow$ sdes_encrypt_sbox(i, block)
  return $k$
```

The S-boxes are simple look-up tables, where each hexadecimal digit is replaced with another according to a 4 look-up tables. The choice of the look-up table is the modulo of the round number, see Table 1.

3.6 S-DES Decryption

S-DES decryption is the opposite of the encryption, where we preform the scheme in exactly the same opposite direction of encryption.

First, we start with the a top function to break incoming hexadecimal data into blocks:

Algorithm 6 Top-Level SDES Decrypt

```
Require: ct_msg: data to be decrypted
Require: ct_sz: number of bytes to decrypted
  for each 32-bit block b in ct_msg do
    $b \leftarrow$ padded(b)
    $p \leftarrow$ sdes_decrypt_block(b)
    append $p$ to pt_msg
  end for
  return pt_msg
```

Then, we implement a function to implement S-DES decryption of one block:

Algorithm 7 Single block S-DES Decrypt

```
Require: ct_block: block to be decrypted
Require: key: main key for decrypted
  block $\leftarrow$ sdes_perm(ct_block)
  for each round in the 8 rounds decreasingly (i.e., starting from 7 down to 0) do
    block $\leftarrow$ sdes_decrypt_round(round_num, block, roundkey)
    roundkey $\leftarrow$ sdes_gen_roundkey(round_num, key)
  end for
  return block
```

Please, note that the rounds are sequenced from 7 down to 0 (i.e., in opposite order of the encryption)

Then, we implement a decryption round:

Algorithm 8 Single S-DES Decrypt Round

```
Require: block: block to be encrypted
Require: roundkey: 32-bit roundkey key
Require: i: round index
  block $\leftarrow$ sdes_decrypt_xobs(i, block)
  block $\leftarrow$ block $\oplus$ roundkey
  return $k$
```

Finally, the S-boxes are simple reverse substitution, where each hexadecimal digit is replaced with its column index according to a four look-up tables (i.e., opposite of S-box substitution). The choice of the look-up table is the modulo of the round number (consult Table 1).

References

[1] Eli Biham and Adi Shamir. “Differential Cryptanalysis of DES-Like Cryptosystem”. In: *Journal of Cryptology* 4.1 (1991).

[2] Eli Biham and Adi Shamir. “Differential Cryptanalysis of the Full 16-Round DES”. In: *Proc. of Crypto* (1992).

[3] Whitfield Diffie and Martin E. Hellman. “Special Feature: Exhaustive Cryptanalysis of the NBS Data Encryption Standard”. In: *Computer* 10.6 (1977), pp. 74–84.

[4] Schaefer Edward F. “A Simplified Data Encryption Standard Algorithm”. In: *Cryptologia* 20.1 (1996), pp. 77–84.

[5] Horst Feistel. “Cryptography and Computer Privacy”. In: *Scientific American* 228.5 (1973), pp. 15–23.

[6] Horst Feistel. “US3798359A: Block Cipher Cryptographic System”. In: *US Patent* (1971).

[7] National Bureau of Standard (NBS). “Data Encryption Standard”. In: *Federal Information Processing Standards Publication (FIPS PUB) 46* (1977).