Introduction

This report provides an in-depth exploration of the Data Encryption Standard (DES) algorithm, with a focus on software optimization techniques applied to text ciphering. The report will cover the following aspects of the DES algorithm and its implementation:

1. Initial Permutation (IP) and Final Permutation (FP)

The initial_permutation_table and final_permutation_table specify the initial and final permutations, respectively, applied to a 64-bit plaintext block before and after encryption.

2. Expansion Table

The expansion_table expands a 32-bit half-block to 48 bits, facilitating mixing bits and introducing non-linearity in DES encryption.

3. S-Boxes

S-boxes (s_boxes) are substitution boxes used in the DES function to perform substitution, reducing the input from 48 bits to 32 bits.

4. P-Box (Permutation Box)

The p_box_table defines a permutation pattern applied to a 32-bit binary string during each DES round.

5. Helper Functions

Several helper functions are defined:

initial_permutation: Performs initial permutation of input text.

- **expansion:** Expands a 32-bit block to 48 bits.
- **xor:** Performs bitwise XOR operation.
- **substitute:** Substitutes values using S-boxes.
- **p_box_permutation:** Performs permutation according to P-box table.

6. DES Function

The des_function applies DES encryption to a 32-bit right half-block using a 48-bit subkey.

7. Process Block

The process_block function processes a 64-bit block through 16 rounds of DES encryption/decryption.

8. DES Encryption and Decryption

The des_encrypt and des_decrypt functions perform encryption and decryption, respectively, using DES algorithm.

9. Source Code: Finally, readers will have access to the complete source code utilized in this project. This provides a transparent and practical example of the DES algorithm's implementation, facilitating further exploration and experimentation.

Conclusion

In conclusion, this report provides a detailed insight into the DES algorithm and its implementation in Python. By optimizing the software for text ciphering, the DES algorithm ensures secure data transmission and storage, making it a crucial tool in modern cryptography.

Data Encryption Standard (DES) Algorithm

1. Initial Permutation (IP) and Final Permutation (FP)

```

initial_permutation_table = [
    58, 50, 42, 34, 26, 18, 10, 2,
    60, 52, 44, 36, 28, 20, 12, 4,
    62, 54, 46, 38, 30, 22, 14, 6,
    64, 56, 48, 40, 32, 24, 16, 8,
    57, 49, 41, 33, 25, 17, 9, 1,
    59, 51, 43, 35, 27, 19, 11, 3,
    61, 53, 45, 37, 29, 21, 13, 5,
    63, 55, 47, 39, 31, 23, 15, 7
]

final_permutation_table = [
    40, 8, 48, 16, 56, 24, 64, 32,
    39, 7, 47, 15, 55, 23, 63, 31,
    38, 6, 46, 14, 54, 22, 62, 30,
    37, 5, 45, 13, 53, 21, 61, 29,
    36, 4, 44, 12, 52, 20, 60, 28,
    35, 3, 43, 11, 51, 19, 59, 27,
    34, 2, 42, 10, 50, 18, 58, 26,
    33, 1, 41, 9, 49, 17, 57, 25
]

```

Initial_permutation_table;

1. It's a list of 64 integers, each representing a position in a 64-bit binary string.
2. **Purpose:**
    - The initial_permutation_table specifies the initial permutation to be applied to the 64-bit plaintext block before the main encryption rounds.

- This permutation rearranges the bits of the plaintext to enhance the cryptographic properties of the encryption process.

3. **Indices Explanation:**
   - Each number in the initial_permutation_table represents a position in the input 64-bit block.
   - The value at each index in initial_permutation_table indicates the position from which to take the bit in the input block.
   - For example, the first element of initial_permutation_table is 58, meaning the first bit of the output will be the 58th bit of the input.

Final_permutation_table

1. It's a list of 64 integers, representing positions in a 64-bit binary string.

2. **Purpose:**
   - The **final_permutation_table** specifies the final permutation to be applied to the 64-bit block before producing the ciphertext.
   - It rearranges the bits of the 64-bit block according to the specified positions.
   - This step ensures the final output is in the correct order for encryption or decryption.

3. **Indices Explanation:**
   - Each number in the **final_permutation_table** represents a position in the input 64-bit block.
   - The value at each index in **final_permutation_table** indicates the position from which to take the bit in the input block.
   - For example, the first element of **final_permutation_table** is **40**, meaning the first bit of the output will be the 40th bit of the input.

2. Expansion Table

```
expansion_table = [
    32, 1, 2, 3, 4, 5,
    4, 5, 6, 7, 8, 9,
    8, 9, 10, 11, 12, 13,
    12, 13, 14, 15, 16, 17,
    16, 17, 18, 19, 20, 21,
    20, 21, 22, 23, 24, 25,
    24, 25, 26, 27, 28, 29,
    28, 29, 30, 31, 32, 1
]
```

1. The expansion table (**expansion_table**) is used to expand a 32-bit half-block to 48 bits. This expansion facilitates mixing bits from different parts of the block and introduces non-linearity. This is a list of 48 integers, each representing a position in a 32-bit binary string.

2. **Purpose:**
    - The **expansion_table** defines how to expand a 32-bit half-block to a 48-bit block.
    - This expansion is necessary to match the 48-bit subkeys used in each DES round.
    - The table specifies which bits from the 32-bit input are duplicated and reordered to form the 48-bit output.

3. **Indices Explanation:**
    - Each number in the **expansion_table** represents a position in the input 32-bit block.
    - The value at each index in **expansion_table** indicates the position from which to take the bit in the input block.
    - For example, the first element of **expansion_table** is **32**, which means that the first bit of the output will be the 32nd bit of the input.

3. S-Boxes

```
s_boxes = {
    0: [
        [14, 4, 13, 1, 2, 15, 11, 8, 3, 10, 6, 12, 5, 9, 0, 7],
        [0, 15, 7, 4, 14, 2, 13, 1, 10, 6, 12, 11, 9, 5, 3, 8],
        [4, 1, 14, 8, 13, 6, 2, 11, 15, 12, 9, 7, 3, 10, 5, 0],
        [15, 12, 8, 2, 4, 9, 1, 7, 5, 11, 3, 14, 10, 0, 6, 13]
    ],
    1: [
        [15, 1, 8, 14, 6, 11, 3, 4, 9, 7, 2, 13, 12, 0, 5, 10],
        [3, 13, 4, 7, 15, 2, 8, 14, 12, 0, 1, 10, 6, 9, 11, 5],
        [0, 14, 7, 11, 10, 4, 13, 1, 5, 8, 12, 6, 9, 3, 2, 15],
        [13, 8, 10, 1, 3, 15, 4, 2, 11, 6, 7, 12, 0, 5, 14, 9]
    ],
    2: [
        [10, 0, 9, 14, 6, 3, 15, 5, 1, 13, 12, 7, 11, 4, 2, 8],
        [13, 7, 0, 9, 3, 4, 6, 10, 2, 8, 5, 14, 12, 11, 15, 1],
        [13, 6, 4, 9, 8, 15, 3, 0, 11, 1, 2, 12, 5, 10, 14, 7],
        [1, 10, 13, 0, 6, 9, 8, 7, 4, 15, 14, 3, 11, 5, 2, 12]
    ],
    3: [
        [7, 13, 14, 3, 0, 6, 9, 10, 1, 2, 8, 5, 11, 12, 4, 15],
        [13, 8, 11, 5, 6, 15, 0, 3, 4, 7, 2, 12, 1, 10, 14, 9],
        [10, 6, 9, 0, 12, 11, 7, 13, 15, 1, 3, 14, 5, 2, 8, 4],
        [3, 15, 0, 6, 10, 1, 13, 8, 9, 4, 5, 11, 12, 7, 2, 14]
```

```
    4: [
        [2, 12, 4, 1, 7, 10, 11, 6, 8, 5, 3, 15, 13, 0, 14, 9],
        [14, 11, 2, 12, 4, 7, 13, 1, 5, 0, 15, 10, 3, 9, 8, 6],
        [4, 2, 1, 11, 10, 13, 7, 8, 15, 9, 12, 5, 6, 3, 0, 14],
        [11, 8, 12, 7, 1, 14, 2, 13, 6, 15, 0, 9, 10, 4, 5, 3]
    ],
    5: [
        [12, 1, 10, 15, 9, 2, 6, 8, 0, 13, 3, 4, 14, 7, 5, 11],
        [10, 15, 4, 2, 7, 12, 9, 5, 6, 1, 13, 14, 0, 11, 3, 8],
        [9, 14, 15, 5, 2, 8, 12, 3, 7, 0, 4, 10, 1, 13, 11, 6],
        [4, 3, 2, 12, 9, 5, 15, 10, 11, 14, 1, 7, 6, 0, 8, 13]
    ],
    6: [
        [4, 11, 2, 14, 15, 0, 8, 13, 3, 12, 9, 7, 5, 10, 6, 1],
        [13, 0, 11, 7, 4, 9, 1, 10, 14, 3, 5, 12, 2, 15, 8, 6],
        [1, 4, 11, 13, 12, 3, 7, 14, 10, 15, 6, 8, 0, 5, 9, 2],
        [6, 11, 13, 8, 1, 4, 10, 7, 9, 5, 0, 15, 14, 2, 3, 12]
    ],
    7: [
        [13, 2, 8, 4, 6, 15, 11, 1, 10, 9, 3, 14, 5, 0, 12, 7],
        [1, 15, 13, 8, 10, 3, 7, 4, 12, 5, 6, 11, 0, 14, 9, 2],
        [7, 11, 4, 1, 9, 12, 14, 2, 0, 6, 10, 13, 15, 3, 5, 8],
        [2, 1, 14, 7, 4, 10, 8, 13, 15, 12, 9, 0, 3, 5, 6, 11]
    ]
```

S-boxes (**s_boxes**) are substitution boxes used in the DES function to perform the substitution step. Each S-box takes a 6-bit input and produces a 4-bit output, introducing non-linearity. This dictionary has 8 keys (0 through 7), each corresponding to an S-box. Each S-box is a 4x16 matrix, where each row contains 16 integers ranging from 0 to 15.

1. **Purpose:**
    - S-boxes are used in the substitution step of the DES round function.
    - They take a 6-bit input and produce a 4-bit output, effectively reducing the size of the data while introducing non-linearity.
2. **Structure of an S-box:**
    - Each S-box is a list of four lists (rows).
    - Each row contains 16 integers, representing the possible substitution values.
3. **Using S-boxes in Substitution:** The substitution process uses each 6-bit segment of the expanded half-block to look up a value in one of the S-boxes. The 6-bit segment is split into:
    - The first and last bits form a 2-bit row index.
    - The middle four bits form a 4-bit column index.

4. P-Box (Permutation Box)

```
p_box_table = [
    16, 7, 20, 21,
    29, 12, 28, 17,
    1, 15, 23, 26,
    5, 18, 31, 10,
    2, 8, 24, 14,
    32, 27, 3, 9,
    19, 13, 30, 6,
    22, 11, 4, 25
]
```

1. This is a list of 32 integers, each representing a position in a 32-bit binary string.
2. **Purpose:**
    - The **p_box_table** defines a permutation pattern that is applied to a 32-bit binary string during each DES round.
    - This permutation ensures that bits are diffused throughout the block, enhancing security by spreading the influence of each bit.
3. **Indices Explanation:**
    - Each number in the **p_box_table** represents a position in the input 32-bit block.
    - The value at each index in **p_box_table** indicates the position from which to take the bit in the input block.
    - For example, the first element of **p_box_table** is **16**, which means that the first bit of the output will be the 16th bit of the input.

5. Helper Functions

```
def initial_permutation(text, table):
    return ''.join(text[i - 1] for i in table)

1 usage
def expansion(text, table):
    return ''.join(text[i - 1] for i in table)

2 usages
def xor(bits1, bits2):
    return ''.join('0' if bit1 == bit2 else '1' for bit1, bit2 in zip(bits1, bits2))

1 usage
def substitute(expanded_half_block, s_boxes):
    output = ''
    for i in range(8):
        block = expanded_half_block[i*6:(i+1)*6]
        row = int(block[0] + block[5], 2)
        col = int(block[1:5], 2)
        s_box_value = s_boxes[i][row][col]
        output += f'{s_box_value:04b}'
    return output

1 usage
def p_box_permutation(text, table):
    return ''.join(text[i - 1] for i in table)
```

1 - initial_permutation

Parameters:

- **text:** The input text (a binary string) to be permuted.
- **table:** The permutation table, a list of indices specifying the new order of bits.

Operation:

- This function permutes the text according to the indices specified in table.
- text[i - 1] accesses the bit at the position i-1 (since indices in table are 1-based and Python uses 0-based indexing).
- ''.join(...) concatenates all the permuted bits into a single binary string.

Purpose:

- This function is used for both the initial and final permutations in DES.

2- Expansion

- **text:** The input text (a binary string) to be expanded.
- **table:** The expansion table, a list of indices specifying how to duplicate and reorder bits.

Operation:

Similar to initial_permutation, this function rearranges and duplicates the bits of text according to table.

This is typically used to expand a 32-bit half-block to 48 bits.

Purpose:

This function is used in the expansion step of the DES round function.

3- xor

- **Parameters:**
  - **bits1:** The first binary string.
  - **bits2:** The second binary string**.**
- **Operation:**
  - This function performs a bitwise XOR operation between corresponding bits of bits1 and bits2.
  - **'0' if bit1 == bit2 else '1' returns '0' if the bits are the same and '1' if they are different.**
  - **zip(bits1, bits2)** pairs corresponding bits from **bits1 and bits2.**
  - **''.join(...)** concatenates the **XOR** results into a single binary string**.**
- **Purpose:**
  - This function is used to combine the expanded right half-block with the subkey in each DES round.

4- substitute

- **Parameters:**
  - **expanded_half_block**: The 48-bit expanded half-block.
  - **s_boxes**: A list of 8 S-boxes, each defining a substitution rule.
- **Operation:**

- The function processes the **expanded_half_block** in 6-bit segments.
- For each segment:
    - **block[0] + block[5]** forms the row index (2 bits).
    - **block[1:5]** forms the column index (4 bits).
    - **int(..., 2)** converts these binary strings to integers.
    - **s_boxes[i][row][col]** retrieves the 4-bit substitution value from the appropriate S-box.
    - **f'{s_box_value:04b}'** converts the substitution value to a 4-bit binary string.
    - The 4-bit binary strings are concatenated to form the 32-bit output.
- **Purpose:**
    - This function implements the substitution step in each DES round, providing non-linearity in the transformation.

5 - p_box_permutation

- **Parameters:**
    - **text:** The input text (a binary string) to be permuted.
    - **table:** The permutation table, a list of indices specifying the new order of bits.
- **Operation:**
    - Like initial_permutation, this function rearranges the bits of text according to the table.
    - **text[i - 1]** accesses the bit at the position **i-1.**
    - **''.join(...)** concatenates all the permuted bits into a single binary string.
- **Purpose:**
    - This function is used for the permutation step in each DES round, ensuring diffusion of the bits.

6. DES Function

```python
def des_function(right, key):
    expanded_right = expansion(right, expansion_table)
    xor_result = xor(expanded_right, key)
    substituted = substitute(xor_result, s_boxes)
    return p_box_permutation(substituted, p_box_table)
```

The function **des_function** is defined with two parameters:

- **right**: The 32-bit right half of the current block.
- **key**: The 48-bit subkey for the current DES round.

The 32-bit **right** is expanded to 48 bits using the **expansion** function and the **expansion_table**. This expansion process rearranges and duplicates certain bits to increase the bit count from 32 to 48.

The expanded 48-bit **right** is XORed with the 48-bit **key** using the **xor** function. This produces a 48-bit result called **xor_result**.

The 48-bit **xor_result** is then passed through the **substitute** function, which uses S-boxes (**s_boxes**). S-boxes (Substitution boxes) are used to reduce the 48-bit input back to 32 bits. Each 6-bit segment of the input is replaced with a 4-bit segment according to the S-box rules, resulting in a 32-bit **substituted** output.

The 32-bit **substituted** result undergoes a final permutation using the **p_box_permutation** function and the **p_box_table**. This permutation rearranges the bits according to the P-box (Permutation box) table.

The permuted 32-bit result is returned as the output of the function.

7. Process Block

```python
def process_block(block, keys):
    block = initial_permutation(block, initial_permutation_table)
    left, right = block[:32], block[32:]
    for key in keys:
        new_right = xor(left, des_function(right, key))
        left, right = right, new_right
    final_block = initial_permutation(right + left, final_permutation_table)
    return final_block
```

The function **process_block** is defined with two parameters:

- **block**: A 64-bit segment of the ciphertext in binary format.
- **keys**: A list of subkeys to be used in the DES rounds.

- The 64-bit **block** undergoes an initial permutation using the **initial_permutation** function and the **initial_permutation_table**. This permutation reorders the bits according to the DES initial permutation table.
- The permuted block is divided into two 32-bit halves:
- left: The leftmost 32 bits.
- right: The rightmost 32 bits.
- This loop performs 16 rounds of DES encryption/decryption:
- For each subkey in keys:
- The function des_function is applied to right and the current key, producing a 32-bit output.
- This output is XORed with left using the xor function, resulting in new_right.
- The values of left and right are then swapped: left becomes the old right, and right becomes new_right.
- After completing the 16 rounds, the **right** and **left** halves are concatenated in swapped order (**right + left**), and a final permutation is applied using the **initial_permutation** function and the **final_permutation_table**.

8. DES Encryption and Decryption

```
def des_encrypt(plain_text, keys):

    binary_text = ''.join(format(ord(char), '08b') for char in plain_text)
    while len(binary_text) % 64 != 0:
        binary_text += '0'  # Padding with zeros
    cipher_text = ''
    for i in range(0, len(binary_text), 64):
        block = binary_text[i:i+64]
        cipher_text += process_block(block, keys)
    return cipher_text

2 usages
def des_decrypt(cipher_text, keys):
    plain_text = ''
    for i in range(0, len(cipher_text), 64):
        block = cipher_text[i:i+64]
        plain_text += process_block(block, keys[::-1])
    plain_text = ''.join(chr(int(plain_text[i:i+8], 2)) for i in range(0, len(plain_text), 8))
    return plain_text.rstrip('\x00')
```

The function des_decrypt is defined with two parameters:

- **cipher_text:** The encrypted text in binary format.
- **keys:** A list of keys, containing 16 subkeys each 48 bits long.

- An empty string plain_text is initialized to store the decrypted text.
- The cipher_text is processed in 64-bit blocks.
- For each **64-bit block,** the function process_block is called**.** This function performs the core DES decryption operations on a block. For decryption, the keys are used in reverse order **(keys[::-1]).**
- The decrypted block is appended to **plain_text.**
- Since **plain_text** is still in binary format, each **8-bit** segment is converted to an **ASCII** character.
- **int(plain_text[i:i+8], 2)** converts an 8-bit segment to an integer.
- **chr(int(...))** converts that integer to a character.
- All characters are joined together into a single string **(''.join(...)).**
- Any **null padding** characters **(\x00)** are stripped from the end of **plain_text (rstrip('\x00')).**
- The cleaned plain text is returned as the output of the function.

This report outlines the DES algorithm, its components, and their implementations in Python. The DES algorithm provides encryption and decryption functionalities, ensuring secure data transmission and storage.

brute_force_decrypt

```
1 usage
def brute_force_decrypt(cipher_text):
    for i in range(2 ** 56):
        key = format(i, '056b')
        decrypted_text = des_decrypt(cipher_text, [key]*16)
        if decrypted_text.isprintable():
            print("hacked successful with key:", key)
            print("hacked text:", decrypted_text)
            return
    print("Brute-force decryption failed. No meaningful text found.")

brute_force_decrypt(cipher_text)
```

I did research here, of course it works to crack the password, but it can't. Because I learned that it can be broken on very good computers and that even those working on the internet cannot break it, this process will take a very long time or computers such as quantum must be used. I hope my information is correct. I tried my best.

1. **Looping through Keys:**
    - The function iterates through all possible 56-bit DES keys by using **range(2 ** 56)**.
    - For each iteration **i**, it converts **i** to a 56-bit binary string using **format(i, '056b')**, representing a potential DES key.
2. **Decrypting with Each Key:**
    - For each key, it calls the **des_decrypt** function with the ciphertext and the current key repeated 16 times (since DES uses 16 rounds).
    - The **des_decrypt** function attempts decryption using the provided key.
3. **Checking Decryption:**
    - If the decrypted text is printable (contains only printable ASCII characters), it assumes that decryption was successful.
    - It then prints the hacked key and the decrypted text.
4. **Termination:**
    - If a printable decrypted text is found, the function returns.
    - If no printable decrypted text is found after checking all possible keys, it prints a message indicating that brute-force decryption failed.

OUTPUT

Text before encryption.

```
Plain text:  I remember as a child, and as a young budding naturalist, spending all my time observing and testing
the world around me moving pieces, altering the flow of things, and documenting ways the world
responded to me. Now, as an adult and a professional naturalist, I've approached language in the
same way, not from an academic point of view but as a curious child still building little mud dams
in creeks and chasing after frogs. So this book is an odd thing: it is a naturalist's walk through
the language-making landscape of the English language, and following in the naturalist's tradition
it combines observation, experimentation, speculation, and documentation activities we don't normally
associate with language. This book is about testing, experimenting, and playing with language. It is
a handbook of tools and techniques for taking words apart and putting them back together again in ways
 that I hope are meaningful and legitimate (or even illegitimate). This book is about peeling back
layers in search of the language-making energy of the human spirit. It is about the gaps in meaning
that we urgently need to notice and name the places where our dreams and ideals are no longer fulfilled
 by a society that has become fast-paced and hyper-commercialized. Language is meant to be a playful,
 ever-shifting creation but we have been taught, and most of us continue to believe, that language must
obediently follow precisely prescribed rules that govern clear sentence structures, specific word orders,
 correct spellings, and proper pronunciations. If you make a mistake or step out of bounds there are
countless, self-appointed language experts who will promptly push you back into safe terrain and scold
you for your errors. And in case you need reminding, there are hundreds of dictionaries and grammar books
 to ensure that you remember the "right" way to use English.
```

Ciphered text:

Since it was too long and a screenshot was required, I included it in my codes once and printed it out and took a screenshot. And I fixed it again. If run, the same cipher will appear side by side rather than one under the other.

```
Cipher text:  0110011111110111011010
```

```
Cipher text:
 011001111111011101101001101100001101100110101101001010100000110011001100101000100100000011111100100100110111111111001
0110011111110111011010011011000011011001101011010010101000001100110011001010
00100100000011111100100100110111111111001001101010100001000001110111101110000001100
11110100001111101101000101001001101000000101011111101010110011011001100111101101101110001
11100011110011000111010111001000100101001101101110110101001111100001101110000001000111111110
001011010010001100100010011111111110000001011001111010110001101100110100011001111100100111000
10000001111010000001110101100100100100100011001000110111101011011101101010100101111111010111
0010111100011000000000100001000101111110011101110011101011000101100100000000111100001100101000111010000001100101110110110
1100010010100101000011110111110100110100001100001110101111000111111001111001110010101100001010000001110111110001010111
0111000111011110110101010110011111111110110110110111010101101001101100010110111011010001100001000011110101010010101000
1001010000110101011111111100001010110100111000010010100111011010110011100001011110010110101000101111101101111000011110
110001101010100001011010010111011111010101101011111101111000111101011011101010110101110011001011000011001001001110100
0111111111000110011010000000101000111010001011010000100101011100101010000110111101000000111110000101101111001011011011
00111111100111111100000010011111010100110110110010101010100101110111100100100110110110010001011010011101111011000000000101
10100000001110001101100101011101110001100110101100101001111100011100111000000010101100001011100011010011100011001110001
00111010010111100111100010110000101111111010101101101111011010011001011100110110000001011001000011100000011111010100101
10100111110011101111110110101001100011001110110101111100110011110010001100011000111111101100011101100101011101110110010
01111110101011101111001011111110100111111110001110010000010100000011100001100010111000100000100011110000101101010 00010
00100010011101111110111111111000000001000101001010011010101000010000100110110010111110111110010010100110000101010110011
101110001001110111111011001100101101001000000000000110100110111111101101011110001111101111100010100000110010010101111111
```

Decrypted text:

```
Decrypted text:  I remember as a child, and as a young budding naturalist, spending all my time observing and testing
the world around me moving pieces, altering the flow of things, and documenting ways the world
responded to me. Now, as an adult and a professional naturalist, I eÙ” …ÁÁÉ½… ¡   ±…¹ Õ… ” ¥¸ Ñ¡” )Í…µ” Ý…ä° ¹½Ð ™É½´ …¸ … .
associate with language. This book is about testing, experimenting, and playing with language. It is
a handbook of tools and techniques for taking words apart and putting them back together again in ways
 that I hope are meaningful and legitimate (or even illegitimate). This book is about peeling back
layers in search of the language-making energy of the human spirit. It is about the gaps in meaning
that we urgently need to notice and name the places where our dreams and ideals are no longer fulfilled
 by a society that has become fast-paced and hyper-commercialized. Language is meant to be a playful,
 ever-shifting creation but we have been taught, and most of us continue to believe, that language must
obediently follow precisely prescribed rules that govern clear sentence structures, specific word orders,
 correct spellings, and proper pronunciations. If you make a mistake or step out of bounds there are
countless, self-appointed language experts who will promptly push you back into safe terrain and scold
you for your errors. And in case you need reminding, there are hundreds of dictionaries and grammar books
 to ensure that you remember the  qÉ¥ ¡Ò�Ò�v�’�Fò�W6R�VævÆ-6¸à
```

MY CODES

```
initial_permutation_table = [
    58, 50, 42, 34, 26, 18, 10, 2,
    60, 52, 44, 36, 28, 20, 12, 4,
    62, 54, 46, 38, 30, 22, 14, 6,
    64, 56, 48, 40, 32, 24, 16, 8,
    57, 49, 41, 33, 25, 17, 9, 1,
    59, 51, 43, 35, 27, 19, 11, 3,
    61, 53, 45, 37, 29, 21, 13, 5,
    63, 55, 47, 39, 31, 23, 15, 7
]

final_permutation_table = [
    40, 8, 48, 16, 56, 24, 64, 32,
    39, 7, 47, 15, 55, 23, 63, 31,
    38, 6, 46, 14, 54, 22, 62, 30,
    37, 5, 45, 13, 53, 21, 61, 29,
    36, 4, 44, 12, 52, 20, 60, 28,
    35, 3, 43, 11, 51, 19, 59, 27,
    34, 2, 42, 10, 50, 18, 58, 26,
    33, 1, 41, 9, 49, 17, 57, 25
]

expansion_table = [
    32, 1, 2, 3, 4, 5,
    4, 5, 6, 7, 8, 9,
    8, 9, 10, 11, 12, 13,
    12, 13, 14, 15, 16, 17,
    16, 17, 18, 19, 20, 21,
    20, 21, 22, 23, 24, 25,
    24, 25, 26, 27, 28, 29,
    28, 29, 30, 31, 32, 1
]
```

```
s_boxes = {
    0: [
        [14, 4, 13, 1, 2, 15, 11, 8, 3, 10, 6, 12, 5, 9, 0, 7],
        [0, 15, 7, 4, 14, 2, 13, 1, 10, 6, 12, 11, 9, 5, 3, 8],
        [4, 1, 14, 8, 13, 6, 2, 11, 15, 12, 9, 7, 3, 10, 5, 0],
        [15, 12, 8, 2, 4, 9, 1, 7, 5, 11, 3, 14, 10, 0, 6, 13]
    ],
    1: [
        [15, 1, 8, 14, 6, 11, 3, 4, 9, 7, 2, 13, 12, 0, 5, 10],
        [3, 13, 4, 7, 15, 2, 8, 14, 12, 0, 1, 10, 6, 9, 11, 5],
        [0, 14, 7, 11, 10, 4, 13, 1, 5, 8, 12, 6, 9, 3, 2, 15],
        [13, 8, 10, 1, 3, 15, 4, 2, 11, 6, 7, 12, 0, 5, 14, 9]
    ],
    2: [
        [10, 0, 9, 14, 6, 3, 15, 5, 1, 13, 12, 7, 11, 4, 2, 8],
        [13, 7, 0, 9, 3, 4, 6, 10, 2, 8, 5, 14, 12, 11, 15, 1],
        [13, 6, 4, 9, 8, 15, 3, 0, 11, 1, 2, 12, 5, 10, 14, 7],
        [1, 10, 13, 0, 6, 9, 8, 7, 4, 15, 14, 3, 11, 5, 2, 12]
    ],
    3: [
        [7, 13, 14, 3, 0, 6, 9, 10, 1, 2, 8, 5, 11, 12, 4, 15],
        [13, 8, 11, 5, 6, 15, 0, 3, 4, 7, 2, 12, 1, 10, 14, 9],
        [10, 6, 9, 0, 12, 11, 7, 13, 15, 1, 3, 14, 5, 2, 8, 4],
        [3, 15, 0, 6, 10, 1, 13, 8, 9, 4, 5, 11, 12, 7, 2, 14]
    ],
    4: [
        [2, 12, 4, 1, 7, 10, 11, 6, 8, 5, 3, 15, 13, 0, 14, 9],
        [14, 11, 2, 12, 4, 7, 13, 1, 5, 0, 15, 10, 3, 9, 8, 6],
        [4, 2, 1, 11, 10, 13, 7, 8, 15, 9, 12, 5, 6, 3, 0, 14],
        [11, 8, 12, 7, 1, 14, 2, 13, 6, 15, 0, 9, 10, 4, 5, 3]
    ],
    5: [
        [12, 1, 10, 15, 9, 2, 6, 8, 0, 13, 3, 4, 14, 7, 5, 11],
        [10, 15, 4, 2, 7, 12, 9, 5, 6, 1, 13, 14, 0, 11, 3, 8],
        [9, 14, 15, 5, 2, 8, 12, 3, 7, 0, 4, 10, 1, 13, 11, 6],
        [4, 3, 2, 12, 9, 5, 15, 10, 11, 14, 1, 7, 6, 0, 8, 13]
    ],
    6: [
        [4, 11, 2, 14, 15, 0, 8, 13, 3, 12, 9, 7, 5, 10, 6, 1],
        [13, 0, 11, 7, 4, 9, 1, 10, 14, 3, 5, 12, 2, 15, 8, 6],
        [1, 4, 11, 13, 12, 3, 7, 14, 10, 15, 6, 8, 0, 5, 9, 2],
        [6, 11, 13, 8, 1, 4, 10, 7, 9, 5, 0, 15, 14, 2, 3, 12]
    ],
    7: [
        [13, 2, 8, 4, 6, 15, 11, 1, 10, 9, 3, 14, 5, 0, 12, 7],
        [1, 15, 13, 8, 10, 3, 7, 4, 12, 5, 6, 11, 0, 14, 9, 2],
        [7, 11, 4, 1, 9, 12, 14, 2, 0, 6, 10, 13, 15, 3, 5, 8],
        [2, 1, 14, 7, 4, 10, 8, 13, 15, 12, 9, 0, 3, 5, 6, 11]
    ]
}

p_box_table = [
    16, 7, 20, 21,
    29, 12, 28, 17,
    1, 15, 23, 26,
    5, 18, 31, 10,
    2, 8, 24, 14,
    32, 27, 3, 9,
    19, 13, 30, 6,
    22, 11, 4, 25
```

```
]

def initial_permutation(text, table):
    return ''.join(text[i - 1] for i in table)

def expansion(text, table):
    return ''.join(text[i - 1] for i in table)

def xor(bits1, bits2):
    return ''.join('0' if bit1 == bit2 else '1' for bit1, bit2 in
zip(bits1, bits2))

def substitute(expanded_half_block, s_boxes):
    output = ''
    for i in range(8):
        block = expanded_half_block[i*6:(i+1)*6]
        row = int(block[0] + block[5], 2)
        col = int(block[1:5], 2)
        s_box_value = s_boxes[i][row][col]
        output += f'{s_box_value:04b}'
    return output

def p_box_permutation(text, table):
    return ''.join(text[i - 1] for i in table)

def des_function(right, key):
    expanded_right = expansion(right, expansion_table)
    xor_result = xor(expanded_right, key)
    substituted = substitute(xor_result, s_boxes)
    return p_box_permutation(substituted, p_box_table)

def process_block(block, keys):
    block = initial_permutation(block, initial_permutation_table)
    left, right = block[:32], block[32:]
    for key in keys:
        new_right = xor(left, des_function(right, key))
        left, right = right, new_right
    final_block = initial_permutation(right + left,
final_permutation_table)
    return final_block

def des_encrypt(plain_text, keys):

    binary_text = ''.join(format(ord(char), '08b') for char in plain_text)
    while len(binary_text) % 64 != 0:
        binary_text += '0'  # Padding with zeros
    cipher_text = ''
    for i in range(0, len(binary_text), 64):
        block = binary_text[i:i+64]
        cipher_text += process_block(block, keys)
    return cipher_text

def des_decrypt(cipher_text, keys):
    plain_text = ''
    for i in range(0, len(cipher_text), 64):
        block = cipher_text[i:i+64]
        plain_text += process_block(block, keys[::-1])
    plain_text = ''.join(chr(int(plain_text[i:i+8], 2)) for i in range(0,
len(plain_text), 8))
    return plain_text.rstrip('\x00')
```

```
key = '0001001100110100010101110111100110011011101111001101111111110001'
keys = [key for _ in range(16)]
print()
plain_text = ('I remember as a child, and as a young budding naturalist, spending all my time observing and testing\n'
              'the world around me moving pieces, altering the flow of things, and documenting ways the world '
              'responded to me. Now, as an adult and a professional naturalist, I’ve approached language in the \n'
              'same way, not from an academic point of view but as a curious child still building little mud dams '
              'in creeks and chasing after frogs. So this book is an odd thing: it is a naturalist’s walk through \n'
              'the language-making landscape of the English language, and following in the naturalist’s tradition '
              'it combines observation, experimentation, speculation, and documentation activities we don’t normally \n'
              'associate with language. This book is about testing, experimenting, and playing with language. It is '
              'a handbook of tools and techniques for taking words apart and putting them back together again in ways\n'
              ' that I hope are meaningful and legitimate (or even illegitimate). This book is about peeling back '
              'layers in search of the language-making energy of the human spirit. It is about the gaps in meaning \n'
              'that we urgently need to notice and name the places where our dreams and ideals are no longer fulfilled'
              ' by a society that has become fast-paced and hyper-commercialized. Language is meant to be a playful,\n'
              ' ever-shifting creation but we have been taught, and most of us continue to believe, that language must'
              'obediently follow precisely prescribed rules that govern clear sentence structures, specific word orders,\n'
              ' correct spellings, and proper pronunciations. If you make a mistake or step out of bounds there are '
              'countless, self-appointed language experts who will promptly push you back into safe terrain and scold \n'
              'you for your errors. And in case you need reminding, there are hundreds of dictionaries and grammar books'
              ' to ensure that you remember the “right” way to use English.')
print()
cipher_text = des_encrypt(plain_text, keys)
decrypted_text = des_decrypt(cipher_text, keys)
print()
print("Plain text: ", plain_text)
print()
print()
print("Cipher text: ", cipher_text)
print()
print()
print("Decrypted text: ", decrypted_text)
print()

def brute_force_decrypt(cipher_text):
    for i in range(2 ** 56):
        key = format(i, '056b')
        decrypted_text = des_decrypt(cipher_text, [key]*16)
        if decrypted_text.isprintable():
            print("hacked successful with key:", key)
```

```
            print("hacked text:", decrypted_text)
            return
    print("Brute-force decryption failed. No meaningful text found.")

brute_force_decrypt(cipher_text)
```