**Symmetric Encryption Algorithm Implementation**

**Experiment Objective**: Learn and reinforce the understanding of major encryption methods in symmetric encryption, such as substitution, permutation, XOR, and round encryption.

**Experiment Content**: Choose one of the following three topics based on your own situation:

1. Implement the DES encryption algorithm based on classroom lectures and perform encryption and decryption operations with a given key and plaintext, outputting the results in TXT format.
2. Implement the AES encryption algorithm based on classroom lectures and perform encryption and decryption operations with a given key and plaintext, outputting the results in TXT format.
3. Write your own encryption method using substitution, permutation, XOR, and round encryption based on the major methods of symmetric encryption. Requirements: The key length should be 1024 bits, and the number of encryption rounds should be no less than 10 and no more than 20.

Experiment Report: Choose Experiment 1 or 2 to complete Report Title 1; choose Experiment 3 to complete Report Title 2.

Report Title 1:

a) Describe the main steps and methods of the DES or AES encryption algorithm in your own words. Divide the algorithm into 4-8 steps and describe each step in no more than 100 words, based on your understanding.

1）Convert the input information into binary bits.

2）Group the converted binary bits into blocks of 64 bits each.

3）Rearrange the 64-bit blocks and divide them into left and right halves of 32 bits each.

4）Expand the right half of 32 bits to 48 bits by adding noise, then perform bitwise XOR with a 48-bit key. Replace the resulting 48 bits with 32 bits, and rearrange the 32-bit result. Perform XOR operation with the left half of 32 bits, and use the XOR result as the right half for the next round. The current right half (32 bits) becomes the left half for the next round.

5）Subkey generation algorithm: Replace the 64-bit key with a 56-bit key, then divide it into left and right halves of 28 bits each. Rotate the left and right halves according to the number of rounds, respectively, by a certain number of positions. The rotated results serve as the input subkeys for the next round. The rotated left and right 28-bit keys can also be combined into a 56-bit key, which is replaced with a 48-bit key for XOR with the information in each round.

Repeat steps 4 and 5 for 16 rounds.

6）After 16 rounds, swap the left and right 32 bits, and then perform rearrangement. The rearranged result is the encrypted result.

b) Decompose the encryption and decryption algorithms into multiple step functions. List the declarations of each step function and describe their functionalities, parameter meanings, and return value results one by one.

void charTobit(char input[500], char output[1000])

Convert the input string message into a binary bit representation and store it in an array.

Parameters:

char input[500]: The input string message.

char output[1000]: The binary bit representation of the input string message.

No return value.

int ifIs64(char input[200])

Check if the length of the input binary bit representation is a multiple of 64.

Parameters:

char input[200]: The binary bit representation.

Return value: 0 or -1. Return -1 if the length is a multiple of 64; return 0 if it is not.

int IP\_64(int input[64], int output[64])

Rearrange the input 64-bit binary bit representation according to the IP\_Table.

Parameters:

int input[64]: The input 64-bit binary bit representation.

int output[64]: The rearranged 64-bit binary bit representation.

No return value.

void IPR\_64(int input[64], int output[64])

After 16 rounds of encryption, rearrange the 64-bit ciphertext according to the IPR\_Table.

Parameters:

int input[64]: The input 64-bit binary bit representation.

int output[64]: The rearranged 64-bit binary bit representation.

No return value.

void messageRightTo48(int input[32], int message48[48])

Expand the right half of the input plaintext (32 bits) to 48 bits using E Table (represented by Table\_48 in the program).

Parameters:

int input[32]: The right half of the input plaintext (32 bits).

int message48[48]: The expanded 48-bit result.

No return value.

void Xor(int array1[M], int array2[M], int xorResult[M], int arrayLen)

Perform bitwise XOR operation between the input binary bit arrays array1 and array2, and store the XOR result in xorResult array.

Parameters:

int array1[M]: One of the arrays to be XORed.

int array2[M]: One of the arrays to be XORed.

int xorResult[M]: The array to store the XOR result.

int arrayLen: The actual number of bits involved in the XOR operation.

No return value.

void binaryTodecimal(int array[], int &result, int arrayLen)

Convert the binary bit representation stored in the array into a decimal number.

Parameters:

int array[]: The array storing the binary bit representation.

int &result: The variable to store the decimal result.

int arrayLen: The actual length of the binary bit representation.

No return value.

void decimalTobinary(int result, int array[4])

Convert a decimal number into a binary bit representation.

Parameters:

int result: The decimal number.

int array[4]: The result of the decimal-to-binary conversion.

No return value.

void message48To32(int input[48], int output[32])

Replace the 48-bit binary bit representation with a 32-bit representation using S-box.

Parameters:

int input[48]: The 48-bit binary bit representation to be replaced.

int output[32]: The resulting 32-bit binary bit representation.

No return value.

void message32Rearrange(int input[32], int output[32])

Rearrange the 32-bit binary bit representation according to the P-box.

Parameters:

int input[32]: The 32-bit binary bit representation to be rearranged.

int output[32]: The rearranged 32-bit binary bit representation.

No return value.

void key64To56(int input[64], int output[56])

Replace the 64-bit key with a 56-bit key according to a certain rule (excluding the 8th bit in each row).

Parameters:

int input[64]: The original 64-bit binary bit key.

int output[56]: The resulting 56-bit binary bit key.

No return value.

void key56Rearrange(int input[56], int output[56])

Rearrange the 56-bit key.

Parameters:

int input[56]: The 56-bit key to be rearranged.

int output[56]: The rearranged 56-bit key.

No return value.

void keyLeftSpin(int input[28], int output[28], int loop)

Perform left circular shift on the 28-bit key according to the relationship between the number of rounds and the number of left shift positions.

Parameters:

int input[28]: The 28-bit key to be left-shifted.

int output[28]: The result of the left circular

void key56To48(int input[56], int output[56])

Replace the 56-bit key with a 48-bit key according to certain rules.

Parameters:

int input[56]: The 56-bit key to be replaced.

int output[56]: The resulting 48-bit key.

No return value.

void swap(int input[64], int output[64])

Swap the first 32 bits and the last 32 bits of the binary bit string.

Parameters:

int input[64]: The 64-bit binary bit string to be swapped.

int output[64]: The swapped 64-bit binary bit string.

No return value.

void pretreatment(char message[500], char key1[100], int handleMessage[500], int key56[56])

Preprocessing function: Convert the plaintext into a binary bit string and check if the length of the converted plaintext is a multiple of 64. If not, pad it with zeros to make it a multiple of 64. Convert the key into a binary bit string and use the key64To56 function to replace the key with a 56-bit key.

Parameters:

char message[500]: The plaintext message.

char key1[100]: The input key.

int handleMessage[500]: The preprocessed binary bit representation of the plaintext.

int key56[56]: The preprocessed binary bit representation of the key.

No return value.

void oneroundEncryption(int Message64[64], int key56[56], int loop)

Perform the entire process of encryption for one round of information.

Parameters:

int Message64[64]: The 64-bit information input for one round of encryption.

int key56[56]: The 56-bit key input for one round of encryption.

int loop: The number of encryption rounds.

No return value.

void oneroundDecryption(int Message64[64], int key56[56], int loop)

Perform the entire process of decryption for one round of information.

Parameters:

int Message64[64]: The 64-bit information input for one round of decryption.

int key56[56]: The 56-bit key input for one round of decryption.

int loop: The number of decryption rounds.

No return value.

void encryption(char message[500], char key1[100])

Encryption function that implements the entire DES encryption process by calling other functions and performing additional operations to obtain the encrypted result.

Parameters:

char message[500]: The plaintext to be encrypted.

char key1[100]: The key used for encryption.

No return value.

void decryption(int secretMessage[500], char key2[100])

Decryption function that implements the entire DES decryption process by calling other functions and performing additional operations to obtain the decrypted result.

Parameters:

int secretMessage[500]: The ciphertext to be decrypted.

char key2[100]: The key used for decryption.

No return value.

int main(int argc, char \*\*argv)

Invoke the encryption function based on the input plaintext, perform encryption, and write the ciphertext to a file. Read the ciphertext from the file, input the key, and perform decryption.

Note: Some parts of the translation might vary depending on the specific context and conventions used in the programming language.

c) The source of data for the S-box in the algorithm is the result of XOR operation between the expanded 48-bit plaintext and the 48-bit key.

The S-box is stored as a two-dimensional array, where the horizontal index represents a decimal number represented by 2 bits, and the vertical index represents a decimal number represented by 4 bits. The values stored in the array are the substituted numbers.

The XOR result of the 48 bits is stored in a char array. Each time, 6 bits are taken from this array. The 2nd to 5th bits are stored in an array of length 4 called arr1, and the 1st and 6th bits are stored in an array of length 2 called arr2. The decimal number represented by arr1 is calculated and used as the vertical index, and the decimal number represented by arr2 is used as the horizontal index. Then, the value to be substituted at this coordinate is found in the two-dimensional array, and this decimal number is converted back to binary and stored. This process is repeated for all 48 bits.

d)result:

