EMU – Department of Computer Engineering

CMSE 455

Security of Computer Systems

Lab2 Report

Instructor

Prof. Dr. Alexander Chefranov

Assistant

Nada Kollah

Team Members:

Seyit Ahmet Inci 19331143

Salih Katırcıoğlu 19331008

Mustafa Mengütay 20331143

Course Group No: 01

Report Due Date: 27th Apr 2024

Semester Term: Spring 2024-2025

Outline

Introduction

ECB, or Electronic Codebook, is the abbreviation for the Data Encryption Standard (DES). It is DES's most basic and weakest variant. Using the same key, each plaintext block is separately encrypted in the ECB mode of operation. To put it another way, each block of data is treated separately and encrypted with the same key, resulting in a block of ciphertext that matches.
One major problem is that blocks of plaintext will always result in blocks of ciphertext that are identical. This suggests that the ciphertext contains information that may be used to identify the plaintext, opening it up to potential attacks.

Problem Definition

The DES algorithm's software application implementation is demonstrated in this study. After a comprehensive analysis, we present below how it evolved during the writing and modification of the programme using the Python programming language.

Tools Used for Implementation

To implement the code, we utilised Visual Studio, a free integrated development environment (IDE) with extensive functionality that includes code editing, debugging, testing, and more.
JS is the programming language utilised in the creation of DES.

Project Management

Work Distribution

We split up the work evenly, with each person writing a portion of the code, understanding it, and providing an explanation.
Seyit oversaw the report, to which each team member contributed their assigned sections.

Meetings

We had 3 meetings in total.

First Meeting:

Date: 17 April 2024

Objective: Division of task and choose programming language

Result: Programming language determined and tasks divided for each group member

Second Meeting:

Date: 24 April 2024

Objective: Merging each part, fixing bugs, and review report following code.

Result: All parts merged and checked running successfully. Also, the lab report reviewed.

Third Meeting:

Date: 27 April 2024

Objective: Improvements in the code for the extended deadline after assistant approval

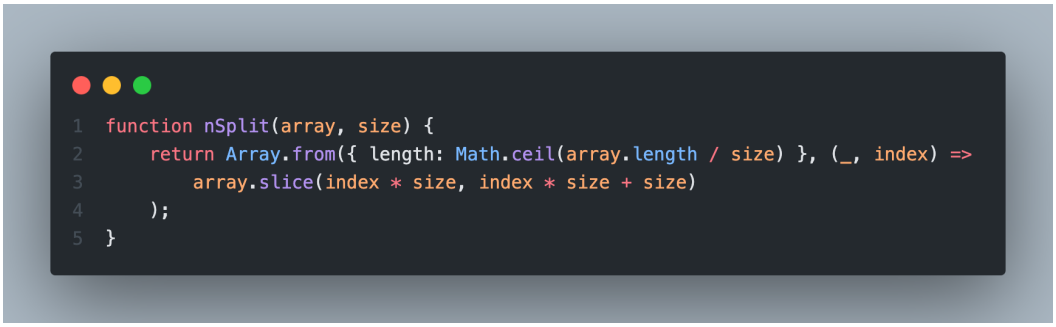
Result: New project done and checked running successfully. Also, the lab report reviewed.

Description of the Developed Program

This laboratory assignment introduces several essential JavaScript functions:

Code Segment 1.1:

nSplit Function : This function efficiently splits an array into smaller chunks of the specified size and returns an array containing these chunks.



```
function nSplit(array, size) {
    return Array.from({ length: Math.ceil(array.length / size) }, (_, index) =>
        array.slice(index * size, index * size + size)
    );
}
```

Code Segment 1.2:

convertToString Function: This function, convertToString, serves to transform an array of binary bits into a string of characters.

```
const convertToString = array => {
    // Chunking array of bits to 8-sized bytes
    const byteChunks = nSplit(array, 8);

    // Converting each byte to char and then concatenating
    const result = byteChunks.map(byte => String.fromCharCode(parseInt(byte.join(''), 2))).join('');

    // Returning result
    return result;
};
```

Code Segment 1.3:

convertToBinary Function: This function, convertToBinary, converts a given text string into an array of binary bits.

```
const convertToBinary = text => {
    // Initializing variable required
    let bitArray = [];
    for (const letter of text) {
        // Getting binary (8-bit) value of letter
        const binVal = binValue(letter.charCodeAt(0), 8);
        // Making list of the bits
        const binValArr = Array.from(binVal, Number);
        // Appending the bits to array
        bitArray.push(...binValArr);
    }

    // Returning answer
    return bitArray;
};
```

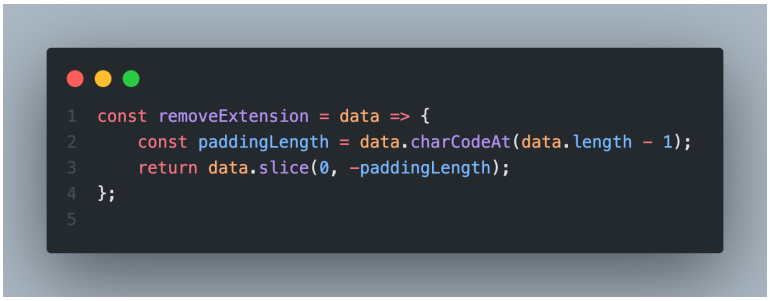
Code Segment 1.4:

binValue Function: This function binValue takes two parameters: val, the value to be converted to binary, and bitSize, the size of the resulting binary string.

```
const binValue = (val, bitSize) => {
    // Function to return the binary value as a string of given size.
    let binVal = val.toString(2);

    // Appending with the required number of zeros in front
    while (binVal.length < bitSize) {
        binVal = "0" + binVal;
    }

    // Returning binary value
    return binVal;
};
```

Code Segment 2.1:

removeExtension Function: The function removeExtension is designed to remove padding added to data during encryption.



```
const removeExtension = data => {
    const paddingLength = data.charCodeAt(data.length - 1);
    return data.slice(0, -paddingLength);
};

```

Code Segment 2.2:

DESDecryption Function: The DESDecryption function decrypts ciphertext using the Data Encryption Standard (DES) algorithm.

```
const DESDecryption = (key, text, extension) => {
    const plainText = DES(text, key, false);

    if (extension === true) {
        return removeExtension(plainText);
    }

    return plainText;
};
```

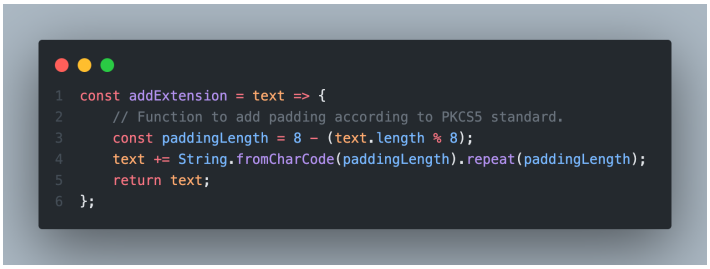
Code Segment 3.1:

divide_to_bytes.nSplit Function: The nSplit function divides an array of elements into smaller chunks of a specified size.

```
const nSplit = (text_list, size) => {
    const result = [];
    for (let i = 0; i < text_list.length; i += size) {
        result.push(text_list.slice(i, i + size));
    }
    return result;
};
```

Code Segment 4.1:

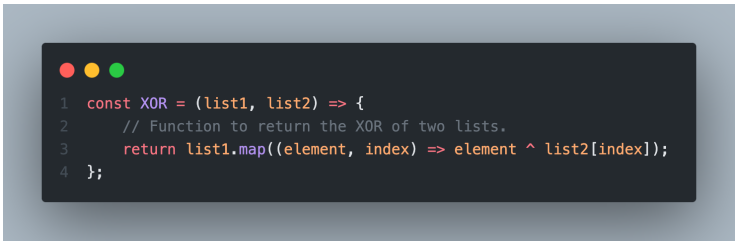
addExtention Function: The addExtension function adds padding to the input text according to the PKCS5 standard.



```
const addExtension = text => {
    // Function to add padding according to PKCS5 standard.
    const paddingLength = 8 - (text.length % 8);
    text += String.fromCharCode(paddingLength).repeat(paddingLength);
    return text;
};
```

Code Segment 4.2:

XOR Function: The XOR function takes two arrays of equal length as input and performs the XOR operation element-wise between corresponding elements of the arrays.



```
const XOR = (list1, list2) => {
    // Function to return the XOR of two lists.
    return list1.map((element, index) => element ^ list2[index]);
};
```

Code Segment 4.3:

DES: This DES function is the core of the Data Encryption Standard (DES) algorithm. Let's break down what it does:

 # Input Parameters

- text: The input text to be encrypted or decrypted.

- key: The encryption/decryption key.

- isEncrypt: A boolean flag indicating whether to encrypt (true) or decrypt (false) the text.

 # Key Generation

The function generates the required round keys based on the provided encryption key. This step is crucial for both encryption and decryption.

 # Text Processing

The input text is divided into 8-byte blocks, as required by the DES algorithm.

Each block is converted into a binary representation.

 # Encryption or Decryption

For each block:

- The initial permutation is applied to the block.

- The block is divided into two halves: left and right.

- The main encryption or decryption process is repeated for 16 rounds.

- In each round, the right half of the block is expanded, XOR-ed with the round key, passed through substitution boxes (S-boxes), and permuted.

- Finally, the left and right halves are swapped, and the process is repeated for the next round.

- After the last round, the two halves are concatenated and subjected to the final permutation.

 # Output

- The function returns the final encrypted or decrypted result as a string.

This function encapsulates the entire DES algorithm, providing both encryption and decryption functionalities based on the specified key and input text.

```
const DES = (text, key, isEncrypt) => {
    const isDecrypt = !isEncrypt;
    const keys = generateKeys(key);
    const plain_text_to_8byte_blocks = nSplit(text, 8);
    let result = [];

    for (const block of plain_text_to_8byte_blocks) {
        let blockArray = convertToBinary(block);
        blockArray = permutation(blockArray, initialPermutationMatrix);
        let [left_block, right_block] = nSplit(blockArray, 32);
        let temp;

        for (let i = 0; i < 16; i++) {
            const expanded_right_block = expand(right_block, expandMatrix);
            temp = isEncrypt ? XOR(keys[i], expanded_right_block) : XOR(keys[15 - i], expanded_right_block);
            temp = Sbox_substitution(temp);
            temp = permutation(temp, eachRoundPermutationMatrix);
            temp = XOR(left_block, temp);
            left_block = right_block;
            right_block = temp;
        }

        result = result.concat(permutation(right_block.concat(left_block), finalPermutationMatrix));
    }

    const finalResult = convertToString(result);
    return finalResult;
};
```

Code Segment 4.4:

ExpandMatrix Function : The expandMatrix is a table used in the DES algorithm for expanding a 32-bit array to a 48-bit array. How it works:

- The matrix is a list of integers representing the positions in the 32-bit array that need to be copied or expanded.

- Each row in the matrix corresponds to a 6-bit portion of the expanded array.

- The values in the matrix indicate which bits from the 32-bit array are copied to generate the expanded 48-bit array.

- This process repeats for each row in the matrix, resulting in an expanded 48-bit array.

```
const expandMatrix = [
    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
];
```

Code Segment 4.5:

DesEncription Function: The DESEncryption function performs the encryption of plaintext using the DES (Data Encryption Standard) algorithm. Here's how it works:

- Extension Check: If the extension parameter is true, indicating that padding is required, the function calls the addExtension function to add padding to the plaintext according to the PKCS5 standard.

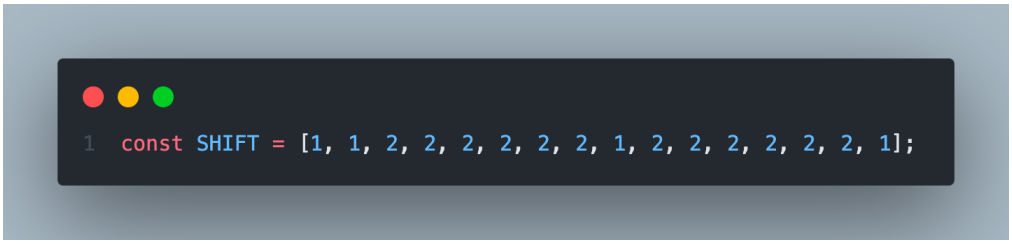
- Encryption: The function then calls the DES function with the plaintext, encryption key (key), and a boolean flag true to indicate encryption.

- Return: The encrypted ciphertext is returned as the result.

```
const DESEncryption = (text, key, extension) => {
    if (extension) {
        text = addExtension(text);
    }
    return DES(text, key, true);
};
```

Code Segment 5.1:

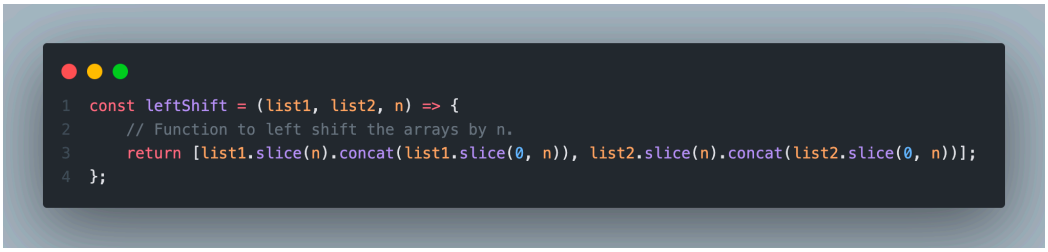
SHIFT array: The SHIFT array defines the number of bit positions to shift each half of the 64-bit block during each round of the DES algorithm. Here's a breakdown: During each round of DES, the 64-bit block is split into two 32-bit halves, the left half and the right half. The SHIFT array specifies the number of bit positions to shift each half leftwards. The number of shifts varies for each round. After each round, both halves are shifted left by the specified number of bits. The leftmost bits that are shifted out are reinserted at the rightmost positions. This shifting operation is a key component of the permutation and substitution processes in DES, contributing to its diffusion and confusion properties.



```
const SHIFT = [1, 1, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 1];
```

Code Segment 5.2:

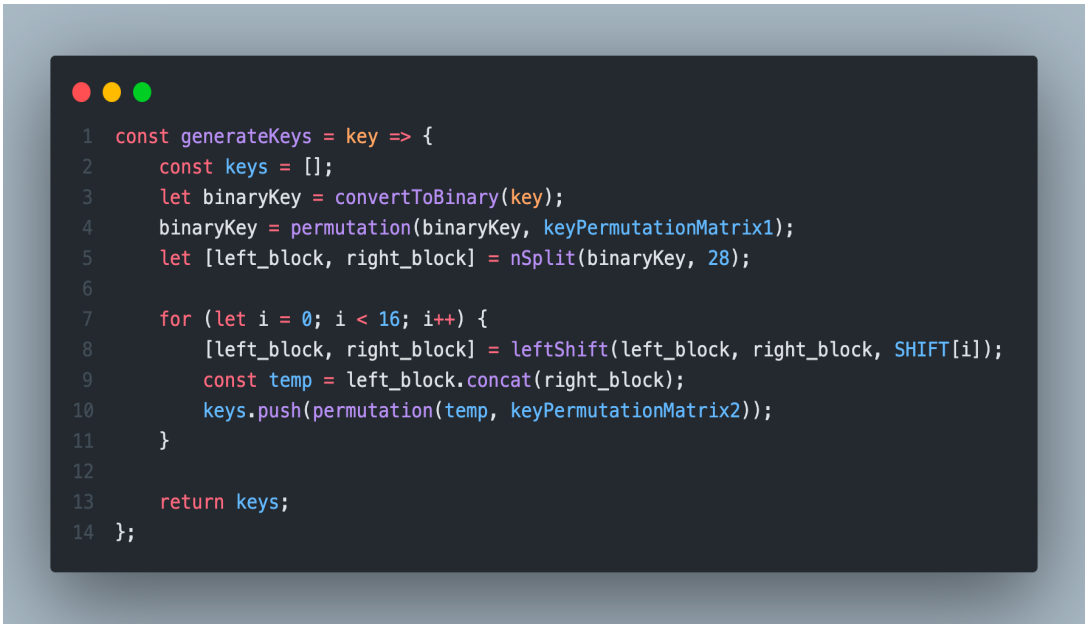
const LEFTSHIFT: The leftShift function performs a left shift operation on two arrays by a specified number of positions n. Here's how it works: It takes two arrays list1 and list2 as input, representing two halves of a block (typically 32 bits each). The function performs a left shift operation on each array independently by n positions. For each array, it slices the array into two parts: The first part consists of elements starting from index n to the end of the array. The second part consists of elements from the beginning of the array up to index n. It then concatenates these two parts in the shifted order. Finally, it returns an array containing the shifted versions of list1 and list2.



```
const leftShift = (list1, list2, n) => {
    // Function to left shift the arrays by n.
    return [list1.slice(n).concat(list1.slice(0, n)), list2.slice(n).concat(list2.slice(0, n))];
};
```

Code Segment 5.3:

keys/generateKeys Function: The generateKeys function is a crucial part of the DES algorithm. It's responsible for generating a set of 16 subkeys from the initial key provided. Here's a breakdown of how it works: Input: It takes a key as input, which is the initial encryption or decryption key provided by the user. Output: It returns an array of 16 subkeys, each of which is used in a specific round of the DES algorithm. Key Generation: The initial key is converted into binary format using the convertToBinary function. The binary key undergoes an initial permutation using keyPermutationMatrix1. The permuted key is then split into two halves, left_block and right_block, each containing 28 bits. For each of the 16 rounds of the DES algorithm: Both halves undergo a left circular shift by a variable amount determined by the SHIFT array. The shifted halves are concatenated and undergo a final permutation using keyPermutationMatrix2. The resulting permutation becomes one of the 16 subkeys, which is added to the keys array. Final Output: The function returns the array keys, containing 16 subkeys, each represented as an array of bits.



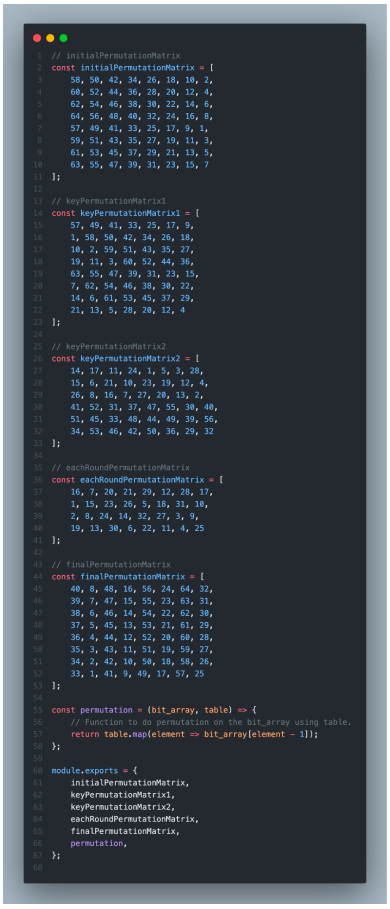
```
const generateKeys = key => {
    const keys = [];
    let binaryKey = convertToBinary(key);
    binaryKey = permutation(binaryKey, keyPermutationMatrix1);
    let [left_block, right_block] = nSplit(binaryKey, 28);

    for (let i = 0; i < 16; i++) {
        [left_block, right_block] = leftShift(left_block, right_block, SHIFT[i]);
        const temp = left_block.concat(right_block);
        keys.push(permutation(temp, keyPermutationMatrix2));
    }

    return keys;
};
```

Code Segment 6:

permutation/perm: The code defines several permutation matrices used in the DES (Data Encryption Standard) algorithm along with a function to perform permutation on a given bit array using a specified permutation table. Here's an explanation of each part:

1. Permutation Matrices: - initialPermutationMatrix: Defines the initial permutation matrix used in the DES algorithm to permute the input data before encryption and after decryption. - keyPermutationMatrix1: Specifies the permutation matrix used to permute the initial encryption key to generate subkeys. - keyPermutationMatrix2: Specifies another permutation matrix used in key generation. - eachRoundPermutationMatrix: Defines the permutation matrix used in each round of the DES algorithm. - finalPermutationMatrix: Specifies the final permutation matrix applied to the output of the last round before producing the final ciphertext.

2. Permutation Function: permutation: This function takes a bit array (bit_array) and a permutation table (table). It performs permutation on the bit array using the specified table. Each element in the table represents the position of the corresponding bit in the output array. These permutation matrices and the permutation function are essential components of the DES algorithm, allowing for the manipulation and transformation of data and keys during encryption and decryption processes.



```
// initialPermutationMatrix
const initialPermutationMatrix = [
    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
];

// keyPermutationMatrix1
const keyPermutationMatrix1 = [
    57, 49, 41, 33, 25, 17, 9,
    1, 58, 50, 42, 34, 26, 18,
    10, 2, 59, 51, 43, 35, 27,
    19, 11, 3, 60, 52, 44, 36,
    63, 55, 47, 39, 31, 23, 15,
    7, 62, 54, 46, 38, 30, 22,
    14, 6, 61, 53, 45, 37, 29,
    21, 13, 5, 28, 20, 12, 4
];

// keyPermutationMatrix2
const keyPermutationMatrix2 = [
    14, 17, 11, 24, 1, 5, 3, 28,
    15, 6, 21, 10, 23, 19, 12, 4,
    26, 8, 16, 7, 27, 20, 13, 2,
    41, 52, 31, 37, 47, 55, 30, 40,
    51, 45, 33, 48, 44, 49, 39, 56,
    34, 53, 46, 42, 50, 36, 29, 32
];

// eachRoundPermutationMatrix
const eachRoundPermutationMatrix = [
    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
];

// finalPermutationMatrix
const finalPermutationMatrix = [
    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
];

const permutation = (bit_array, table) => {
    // function to do permutation on the bit_array using table.
    return table.map(element => bit_array[element - 1]);
};

module.exports = {
    initialPermutationMatrix,
    keyPermutationMatrix1,
    keyPermutationMatrix2,
    eachRoundPermutationMatrix,
    finalPermutationMatrix,
    permutation,
};
```

Code Segment 7:

sbox/SboxesArray Array: The SboxesArray is an array containing eight S-boxes, each represented as a nested array. Each S-box consists of four rows and sixteen columns, where each cell contains a decimal value from 0 to 15.

Here's a brief explanation of the structure:

The outer array contains eight elements, representing eight different S-boxes used in the DES algorithm.

Each inner array represents a single S-box.

Within each S-box array, there are four rows and sixteen columns.

Each row corresponds to a pair of bits from the input, and each column corresponds to a pair of bits from the output.

The value in each cell of the S-box represents a substitution value based on the input bits. These values are used during the encryption and decryption processes to perform substitution-permutation operations.

```
const SboxesArray = [
    [
        [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],
    ],

    [
        [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],
    ],

    [
        [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],
    ],

    [
        [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],
    ],

    [
        [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],
    ],

    [
        [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],
    ],

    [
        [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],
    ],

    [
        [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],
    ]
];
```

Code Segment 8:

sbox/Sbox_substitution Function: The Sbox_substitution function performs the substitution step using S-boxes in the DES algorithm. Here's a breakdown of how it works:

It first splits the input bit array into blocks of 6 bits each using the nSplit function.
Then, for each block:
It extracts the first and last bits (block[0] and block[5]) to determine the row index.
It extracts the middle four bits (block.slice(1, 5)) and joins them into a string to determine the column index.
It uses these row and column indices to access the appropriate value from the S-box array (SboxesArray) corresponding to the current block.
It converts this value into a 4-bit binary string using the binValue function.
Finally, it flattens this binary string into individual bits and appends them to the result array.

```
function Sbox_substitution(bit_array) {
    const blocks = nSplit(bit_array, 6);
    const result = [];

    for (let i = 0; i < blocks.length; i++) {
        const block = blocks[i];
        const row = parseInt(`${block[0]}${block[5]}`, 2);
        const column = parseInt(block.slice(1, 5).join(''), 2);
        const sbox_value = SboxesArray[i][row][column];
        const bit_value = binValue(sbox_value, 4);

        result.push(...bit_value.split('').map(Number));
    }

    return result;
}
```

Client & Server Segments :

Overall, these codes establishes a bidirectional communication channel between the client and the server, allowing them to exchange encrypted messages using the DES algorithm. The user can input messages from the console, which are encrypted and sent to the server, and vice versa.

Server.js

```
const net = require('net');
const readline = require('readline');

const { DESDecryption } = require("./des/decryption");
const { DESEncryption } = require("./des/encryption");

const key = 'cmpe455labProject!';

const server = net.createServer((socket) => {
    console.log('Server: Client connected.');

    // Receive data from Client
    socket.on('data', (data) => {
        // const decryptedMessage = decrypt(data.toString(), key);
        const decryptedMessage = DESDecryption(key, data.toString(), key);
        console.log('Server: Received Encrypted value from Client:', data.toString());
        console.log('Server: Received message from Client:', decryptedMessage);
    });

    // Handle client disconnection
    socket.on('end', () => {
        console.log('Server: Client disconnected.');
    });

    // Set up readline interface for server input
    const rl = readline.createInterface({
        input: process.stdin,
        output: process.stdout
    });

    // Listen for server input
    rl.on('line', (input) => {
        // Send the input message to the client
        // const encryptedInput = encrypt(input, key);
        const encryptedInput = DESEncryption(input, key, key);
        socket.write(encryptedInput);
    });
});

const PORT_A = 3000;
server.listen(PORT_A, () => {
    console.log(`Server: Client listening on port ${PORT_A}`);
});
```

Client.js

```
const net = require('net');
const readline = require('readline');

const { DESDecryption } = require("./des/decryption");
const { DESEncryption } = require("./des/encryption");

const key = 'cmpe455labProject!';

const prompt = readline.createInterface({
    input: process.stdin,
    output: process.stdout
});

// Create a client to connect to Server
const client = new net.Socket();

// Connect to Server A
const PORT_A = 3000;
const HOST_A = '127.0.0.1'; // localhost
client.connect(PORT_A, HOST_A, () => {
    console.log('Client connected to Server.');

    // Start listening for user input
    prompt.prompt();
});

// Handle user input
prompt.on('line', (input) => {
    // Send user input to Server
    const encryptedInput = DESEncryption(input, key, key);
    client.write(encryptedInput);

    // Continue listening for user input
    prompt.prompt();
});

// Handle data received from Server
client.on('data', (data) => {
    const decryptedMessage = DESDecryption(key, data.toString(), key);
    console.log('Client: Received Decrypted message from Server:', data.toString());
    console.log('Client: Received message from Server:', decryptedMessage);
});

// Handle disconnection from Server
client.on('close', () => {
    console.log('Client: connection to server closed.');
});
```

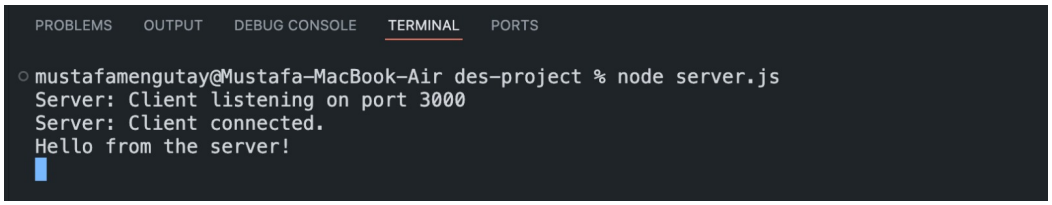
Tests

In our code, we have shown the images of the tests we have done on the server and client side in this section after the code explanations.

How to run the project?

First thing first, Node.js should be installed on the computer. Secondly, Run the server.js -> `node server.js` and lastly, run the client.js -> `node client.js`

1. Server side: message sending output from server to client



2. Client side: message received from server side and decrypted by client. The output shown

3. Client side: message sent by client to server

4. Client side: message received from client and outputed on server side

Conclusion

In summary, this report offers a thorough examination of the implemented Data Encryption Standard (DES) program along with its fundamental components. Through detailed implementation insights and explanations of key functionalities, the report elucidates the program's design and operational aspects. Furthermore, rigorous testing across multiple devices has been conducted to validate the program's functionality, ensuring seamless operation and reliability across varied environments.

References

https://github.com/shashwatkathuria/Cryptography

https://github.com/Baticsute/Handle-DES-Encryption-Decryption

https://github.com/saishmirajkar/File-Encryption-and-Decryption-with-DES-Algorithm

Source Codes

You may find our whole code on our Github account at

https://github.com/Kedu88/CMPE-DES-Algorithm-Project.git

Additionally, the zip file that is included with the report contains the whole code.