Assignment 4

IT851: Information and Systems Security Lab

**ANINDYA KUNDU**

IT, 8th Semester

ID 510817020

Repository:  
[github.com/meganindya/btech-assignments/information-and-systems-security/assg-4](https://github.com/meganindya/btech-assignments/tree/main/information-and-systems-security/assg-4)

10-05-2021

Implement an *Iterated Substitution Permutation cipher* consisting of Nr = 4 rounds, with the following specifications:

1. Each round consists of round-key mixing followed by a substitution and a permutation.
2. Assume the plain text and cipher text, each to be 8-bits long.
3. The key schedule is generated by selecting (4r-3)-th through (4r+4)-th key bits as the round key for round r. (The minimum length of the key is given by 1×8+ Nr×4=24 bits. Select a random string of 24 bits as the key.)
4. The round key mixing is done by a bitwise XOR operation.
5. Perform key whitening at the beginning and end of each round.
6. Assuming l = 4, the substitution function at each round is specified by the following S-box:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Input | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
| Output | E | 4 | D | 1 | 2 | F | B | 8 | 3 | A | 6 | C | 5 | 9 | 0 | 7 |

1. The permutation function for each round is:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Input | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| Output | 1 | 4 | 5 | 7 | 3 | 6 | 2 | 8 |

Implement both the encryption and decryption functions for the above cipher, in the following modes of encipherment:

* Electronic Code Book (ECB) mode
* Cipher Block Chaining (CBC) mode

Source: iterated-substitution-permutation-cipher.c

*#include* <stdio.h>  *// printf, scanf*

*#include* <stdlib.h> *// malloc*

*#include* <string.h> *// strlen*

*/\**

*\* Utility function that converts an integer (base 10) to binary (base 2) string.*

*\* Integer range is [0, 255], which can ver covered by 8 bits*

*\**

*\* n: integer number*

*\* s: character array to fill (binary) bits in (array length assumed to be 8)*

*\*/*

void int\_to\_binary(int n, char \*s)

{

    int mask = 1;

*for* (int i = 0; i < 8; i++)

    {

        s[7 - i] = (n & mask) == 0 ? '0' : '1';

        mask <<= 1;

    }

}

*/\**

*\* Utility function that converts a binary (base 2) string to integer (base 10).*

*\* Integer range is [0, 255], which can ver covered by 8 bits*

*\**

*\* s: character array of bits representing the binary (array length assumed to be 8)*

*\**

*\* returns:*

*\* integer (base 10) equivalent*

*\*/*

int binary\_to\_int(char \*s)

{

    int n = 0, mask = 1;

*for* (int i = 0; i < 8; i++)

    {

        n += (s[7 - i] - '0') \* mask;

        mask <<= 1;

    }

*return* n;

}

*/\**

*\* Utility function that returns the XOR of two bits represented as characters ('0' or '1'),*

*\**

*\* a: operand 1*

*\* b: operand 2*

*\**

*\* returns:*

*\* a ^ b (as character)*

*\*/*

char xor\_bin(char a, char b)

{

*return* a == b ? '0' : '1';

}

int s\_key[16] = {14, 4, 13, 1, 2, 15, 11, 8, 3, 10, 6, 12, 5, 9, 0, 7};

int s\_key\_inv[16];

int p\_key[8] = {1, 4, 5, 7, 3, 6, 2, 8};

int p\_key\_inv[8];

void s\_box(char \*s, int \*key)

{

    char bin\_m[8], bin\_l[8];

*for* (int i = 0; i < 8; i++)

    {

        bin\_m[i] = i < 4 ? '0' : s[i - 4];

        bin\_l[i] = i < 4 ? '0' : s[i];

    }

    int n\_m = binary\_to\_int(bin\_m);

    int n\_l = binary\_to\_int(bin\_l);

    int\_to\_binary(key[n\_m], bin\_m);

    int\_to\_binary(key[n\_l], bin\_l);

*for* (int i = 0; i < 8; i++)

        s[i] = i < 4 ? bin\_m[i + 4] : bin\_l[i];

}

void p\_box(char \*s, int \*key)

{

    char temp[8];

*for* (int i = 0; i < 8; i++)

        temp[i] = s[key[i] - 1];

*for* (int i = 0; i < 8; i++)

        s[i] = temp[i];

}

void mix(char \*s, char \*key)

{

*for* (int i = 0; i < 8; i++)

        s[i] = xor\_bin(s[i], key[i]);

}

void print\_byte(char \*s)

{

*for* (int i = 0; i < 8; i++)

        printf("%c", s[i]);

}

void encrypt\_block(char \*block, char \*key, int no\_p)

{

*for* (int i = 0; i < 8; i++)

        block[i] = xor\_bin(block[i], key[i]);

    s\_box(block, s\_key);

*if* (no\_p == 0)

        p\_box(block, p\_key);

}

void decrypt\_block(char \*block, char \*key, int no\_p)

{

*if* (no\_p == 0)

        p\_box(block, p\_key\_inv);

    s\_box(block, s\_key\_inv);

*for* (int i = 0; i < 8; i++)

        block[i] = xor\_bin(block[i], key[i]);

}

*/\**

*\* Iterative Substitution Permutation Cipher encrypts (in place) blockwise w.r.t a round key.*

*\**

*\* blocks: plaintext blocks (8-bit each)*

*\* blocks\_n: number of blocks*

*\* keys: 5 round keys (8-bit each)*

*\* mode: 'ECB', 'CBC', 'OFB'*

*\*/*

void encrypt(char \*\*blocks, int blocks\_n, char \*\*keys, char \*mode)

{

*if* (strcmp(mode, "ECB") == 0)

    {

*for* (int k = 0; k < 4; k++)

        {

            printf("  Round %d:\n  ----\n", k + 1);

            char key[8];

            printf("    Key:    ");

*for* (int j = 0; j < 8; j++)

            {

                key[j] = keys[k][j];

                printf("%c", key[j]);

            }

            printf("\n");

*for* (int i = 0; i < blocks\_n; i++)

            {

                char block[8];

*for* (int j = 0; j < 8; j++)

                    block[j] = blocks[i][j];

                encrypt\_block(block, key, k == 3 ? 1 : 0);

*for* (int j = 0; j < 8; j++)

                    blocks[i][j] = block[j];

            }

            printf("    Blocks: ");

*for* (int i = 0; i < blocks\_n; i++)

            {

*for* (int j = 0; j < 8; j++)

                    printf("%c", blocks[i][j]);

                printf(" ");

            }

            printf("\n\n");

        }

        printf("  Whitening:\n  ----\n");

        char key[8];

        printf("    Key:    ");

*for* (int j = 0; j < 8; j++)

        {

            key[j] = keys[4][j];

            printf("%c", key[j]);

        }

        printf("\n");

*for* (int i = 0; i < blocks\_n; i++)

        {

            char block[8];

*for* (int j = 0; j < 8; j++)

                block[j] = blocks[i][j];

            char key[8];

*for* (int j = 0; j < 8; j++)

                key[j] = keys[4][j];

            mix(block, key);

*for* (int j = 0; j < 8; j++)

                blocks[i][j] = block[j];

        }

        printf("    Blocks: ");

*for* (int i = 0; i < blocks\_n; i++)

        {

*for* (int j = 0; j < 8; j++)

                printf("%c", blocks[i][j]);

            printf(" ");

        }

        printf("\n");

    }

*else* *if* (strcmp(mode, "CBC") == 0)

    {

*for* (int k = 0; k < 4; k++)

        {

            printf("  Round %d:\n  ----\n", k + 1);

            char key[8];

            printf("    Key:    ");

*for* (int j = 0; j < 8; j++)

            {

                key[j] = keys[k][j];

                printf("%c", key[j]);

            }

            printf("\n");

            char c\_bin[8] = "01101001";

*for* (int i = 0; i < blocks\_n; i++)

            {

                char block[8];

*for* (int j = 0; j < 8; j++)

                    block[j] = blocks[i][j];

                mix(block, c\_bin);

                encrypt\_block(block, key, k == 3 ? 1 : 0);

*for* (int j = 0; j < 8; j++)

                    c\_bin[j] = block[j];

*for* (int j = 0; j < 8; j++)

                    blocks[i][j] = block[j];

            }

            printf("    Blocks: ");

*for* (int i = 0; i < blocks\_n; i++)

            {

*for* (int j = 0; j < 8; j++)

                    printf("%c", blocks[i][j]);

                printf(" ");

            }

            printf("\n\n");

        }

        printf("  Whitening:\n  ----\n");

        char key[8];

        printf("    Key:    ");

*for* (int j = 0; j < 8; j++)

        {

            key[j] = keys[4][j];

            printf("%c", key[j]);

        }

        printf("\n");

*for* (int i = 0; i < blocks\_n; i++)

        {

            char block[8];

*for* (int j = 0; j < 8; j++)

                block[j] = blocks[i][j];

            char key[8];

*for* (int j = 0; j < 8; j++)

                key[j] = keys[4][j];

            mix(block, key);

*for* (int j = 0; j < 8; j++)

                blocks[i][j] = block[j];

        }

        printf("    Blocks: ");

*for* (int i = 0; i < blocks\_n; i++)

        {

*for* (int j = 0; j < 8; j++)

                printf("%c", blocks[i][j]);

            printf(" ");

        }

        printf("\n");

    }

*else*

    {

    }

}

*/\**

*\* Iterative Substitution Permutation Cipher decrypts (in place) blockwise w.r.t a round key.*

*\**

*\* blocks: plaintext blocks (8-bit each)*

*\* blocks\_n: number of blocks*

*\* keys: 5 round keys (8-bit each)*

*\* mode: 'ECB', 'CBC', 'OFB'*

*\*/*

void decrypt(char \*\*blocks, int blocks\_n, char \*\*keys, char \*mode)

{

*if* (strcmp(mode, "ECB") == 0)

    {

        printf("  Whitening:\n  ----\n");

        char key[8];

        printf("    Key:    ");

*for* (int j = 0; j < 8; j++)

        {

            key[j] = keys[4][j];

            printf("%c", key[j]);

        }

        printf("\n");

*for* (int i = 0; i < blocks\_n; i++)

        {

            char block[8];

*for* (int j = 0; j < 8; j++)

                block[j] = blocks[i][j];

            char key[8];

*for* (int j = 0; j < 8; j++)

                key[j] = keys[4][j];

            mix(block, key);

*for* (int j = 0; j < 8; j++)

                blocks[i][j] = block[j];

        }

        printf("    Blocks: ");

*for* (int i = 0; i < blocks\_n; i++)

        {

*for* (int j = 0; j < 8; j++)

                printf("%c", blocks[i][j]);

            printf(" ");

        }

        printf("\n\n");

*for* (int k = 3; k >= 0; k--)

        {

            printf("  Round %d:\n  ----\n", k + 1);

            char key[8];

            printf("    Key:    ");

*for* (int j = 0; j < 8; j++)

            {

                key[j] = keys[k][j];

                printf("%c", key[j]);

            }

            printf("\n");

*for* (int i = 0; i < blocks\_n; i++)

            {

                char block[8];

*for* (int j = 0; j < 8; j++)

                    block[j] = blocks[i][j];

                decrypt\_block(block, key, k == 3 ? 1 : 0);

*for* (int j = 0; j < 8; j++)

                    blocks[i][j] = block[j];

            }

            printf("    Blocks: ");

*for* (int i = 0; i < blocks\_n; i++)

            {

*for* (int j = 0; j < 8; j++)

                    printf("%c", blocks[i][j]);

                printf(" ");

            }

            printf("\n");

*if* (k != 3)

                printf("\n");

        }

    }

*else* *if* (strcmp(mode, "CBC") == 0)

    {

        printf("  Whitening:\n  ----\n");

        char key[8];

        printf("    Key:    ");

*for* (int j = 0; j < 8; j++)

        {

            key[j] = keys[4][j];

            printf("%c", key[j]);

        }

        printf("\n");

*for* (int i = 0; i < blocks\_n; i++)

        {

            char block[8];

*for* (int j = 0; j < 8; j++)

                block[j] = blocks[i][j];

            char key[8];

*for* (int j = 0; j < 8; j++)

                key[j] = keys[4][j];

            mix(block, key);

*for* (int j = 0; j < 8; j++)

                blocks[i][j] = block[j];

        }

        printf("    Blocks: ");

*for* (int i = 0; i < blocks\_n; i++)

        {

*for* (int j = 0; j < 8; j++)

                printf("%c", blocks[i][j]);

            printf(" ");

        }

        printf("\n\n");

*for* (int k = 3; k >= 0; k--)

        {

            printf("  Round %d:\n  ----\n", k + 1);

            char key[8];

            printf("    Key:    ");

*for* (int j = 0; j < 8; j++)

            {

                key[j] = keys[k][j];

                printf("%c", key[j]);

            }

            printf("\n");

            char c\_bin[8] = "01101001";

*for* (int i = 0; i < blocks\_n; i++)

            {

                char block[8];

*for* (int j = 0; j < 8; j++)

                    block[j] = blocks[i][j];

                char bin\_temp[8];

*for* (int j = 0; j < 8; j++)

                    bin\_temp[j] = block[j];

                decrypt\_block(block, key, k == 3 ? 1 : 0);

                mix(block, c\_bin);

*for* (int j = 0; j < 8; j++)

                    c\_bin[j] = bin\_temp[j];

*for* (int j = 0; j < 8; j++)

                    blocks[i][j] = block[j];

            }

            printf("    Blocks: ");

*for* (int i = 0; i < blocks\_n; i++)

            {

*for* (int j = 0; j < 8; j++)

                    printf("%c", blocks[i][j]);

                printf(" ");

            }

            printf("\n");

*if* (k != 3)

                printf("\n");

        }

    }

*else*

    {

    }

}

*// -----------------------------------------------------------------------------*

int main(int argc, char \*argv[])

{

    char s[8];

    char k[8];

    printf("\nImplementation of Iterative Substitution Permutation Cipher\n--------\n");

    printf("Enter an ASCII string to encrypt: ");

    scanf("%[^\n]s", s);

    int repeat;

*do*

    {

        repeat = 0;

        printf("Enter 24-bit key string in Hexadecimal (0-9, A-F): ");

        scanf("%s", k);

*if* (strlen(k) != 6)

        {

            repeat = 1;

*continue*;

        }

*for* (int i = 0; i < 6; i++)

        {

*if* (!((k[i] >= '0' && k[i] <= '9') || (k[i] >= 'A' && k[i] <= 'F')))

            {

                printf("  Invalid string, retry\n");

                repeat = 1;

*break*;

            }

        }

    } *while* (repeat);

    printf("\n========\n");

    int blocks\_n = strlen(s);

    char \*\*blocks;

    blocks = malloc((blocks\_n) \* sizeof \*blocks);

*for* (int i = 0; i < blocks\_n; i++)

    {

        blocks[i] = malloc(8 \* sizeof \*blocks[i]);

    }

    printf("\nBit blocks:\n  ");

*for* (int i = 0; i < blocks\_n; i++)

    {

        char binary[8];

        int\_to\_binary(s[i], binary);

*for* (int j = 0; j < 8; j++)

        {

            blocks[i][j] = binary[j];

        }

        printf(" %c (%d)  ", s[i], s[i]);

    }

    printf("\n  ");

*for* (int i = 0; i < blocks\_n; i++)

    {

*for* (int j = 0; j < 8; j++)

        {

            printf("%c", blocks[i][j]);

        }

        printf(" ");

    }

    char key[24];

    char \*\*keys;

    keys = malloc(5 \* sizeof \*keys);

*for* (int i = 0; i < 5; i++)

    {

        keys[i] = malloc(8 \* sizeof \*keys[i]);

    }

    printf("\n\nKey:\n  ");

*for* (int i = 0; i < 6; i++)

    {

        printf("  %c  ", k[i]);

    }

    printf("\n  ");

*for* (int i = 0; i < 6; i++)

    {

        int n = k[i] >= 'A' && k[i] <= 'F' ? k[i] - 'A' + 10 : k[i] - '0';

        char binary[8];

        int\_to\_binary(n, binary);

*for* (int j = 4; j < 8; j++)

        {

            key[i \* 4 + (j - 4)] = binary[j];

        }

*for* (int j = 0; j < 4; j++)

        {

            printf("%c", key[i \* 4 + j]);

        }

        printf(" ");

    }

    printf("\n\nRound keys:\n");

*for* (int i = 1; i <= 5; i++)

    {

        printf("  ");

*for* (int j = 4 \* i - 4; j < 4 \* i + 4; j++)

        {

*if* (j == 4 \* i)

                printf(" ");

            keys[i - 1][j - (4 \* i - 4)] = key[j];

            printf("%c", key[j]);

        }

        printf("\n");

    }

    char op\_s[blocks\_n];

    printf("\n========================================\n");

    printf("Electronic Code Book mode\n");

    printf("========================================\n");

*for* (int i = 0; i < 8; i++)

        p\_key\_inv[p\_key[i] - 1] = i + 1;

*for* (int i = 0; i < 16; i++)

        s\_key\_inv[s\_key[i]] = i;

    printf("\nEncryption:\n");

    encrypt(blocks, blocks\_n, keys, "ECB");

    printf("\nEncrypted blocks:\n  ");

*for* (int i = 0; i < blocks\_n; i++)

    {

*for* (int j = 0; j < 8; j++)

            printf("%c", blocks[i][j]);

        printf(" ");

    }

    printf("\n  ");

*for* (int i = 0; i < blocks\_n; i++)

    {

        char binary[8];

*for* (int j = 0; j < 8; j++)

            binary[j] = blocks[i][j];

        int n = binary\_to\_int(binary);

        printf("  (%3d)  ", n);

    }

    printf("\n");

    printf("\n========================\n");

    printf("\nDecryption:\n");

    decrypt(blocks, blocks\_n, keys, "ECB");

    printf("\nDecrypted blocks:\n  ");

*for* (int i = 0; i < blocks\_n; i++)

    {

*for* (int j = 0; j < 8; j++)

            printf("%c", blocks[i][j]);

        printf(" ");

    }

    printf("\n  ");

*for* (int i = 0; i < blocks\_n; i++)

    {

        char binary[8];

*for* (int j = 0; j < 8; j++)

            binary[j] = blocks[i][j];

        int n = binary\_to\_int(binary);

        printf(" %c (%3d) ", n, n);

        op\_s[i] = n;

    }

    printf("\n");

    printf("\n--------\nDecrypted string: ");

*for* (int i = 0; i < blocks\_n; i++)

        printf("%c", op\_s[i]);

    printf("\n--------\n");

    printf("\n========================================\n");

    printf("Cipher Block Chaining mode\n");

    printf("========================================\n");

*for* (int i = 0; i < 8; i++)

        p\_key\_inv[p\_key[i] - 1] = i + 1;

*for* (int i = 0; i < 16; i++)

        s\_key\_inv[s\_key[i]] = i;

    printf("\nEncryption:\n");

    encrypt(blocks, blocks\_n, keys, "CBC");

    printf("\nEncrypted blocks:\n  ");

*for* (int i = 0; i < blocks\_n; i++)

    {

*for* (int j = 0; j < 8; j++)

            printf("%c", blocks[i][j]);

        printf(" ");

    }

    printf("\n  ");

*for* (int i = 0; i < blocks\_n; i++)

    {

        char binary[8];

*for* (int j = 0; j < 8; j++)

            binary[j] = blocks[i][j];

        int n = binary\_to\_int(binary);

        printf("  (%3d)  ", n);

    }

    printf("\n");

    printf("\n========================\n");

    printf("\nDecryption:\n");

    decrypt(blocks, blocks\_n, keys, "CBC");

    printf("\nDecrypted blocks:\n  ");

*for* (int i = 0; i < blocks\_n; i++)

    {

*for* (int j = 0; j < 8; j++)

            printf("%c", blocks[i][j]);

        printf(" ");

    }

    printf("\n  ");

*for* (int i = 0; i < blocks\_n; i++)

    {

        char binary[8];

*for* (int j = 0; j < 8; j++)

            binary[j] = blocks[i][j];

        int n = binary\_to\_int(binary);

        printf(" %c (%3d) ", n, n);

        op\_s[i] = n;

    }

    printf("\n");

    printf("\n--------\nDecrypted string: ");

*for* (int i = 0; i < blocks\_n; i++)

        printf("%c", op\_s[i]);

    printf("\n--------\n");

    printf("\n");

*for* (int i = 0; i < blocks\_n; i++)

        free(blocks[i]);

    free(blocks);

*for* (int i = 0; i < 5; i++)

        free(keys[i]);

    free(keys);

}

(sample run on next page)

Sample run



