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

10-05-2021

Implement an *Iterated Substitution Permutation cipher* consisting of $N_r = 4$ 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

7. 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`

```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
 *
```

```c
 * 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 k
ey.
 *
 * 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 k
ey.
 *
 * 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];
```

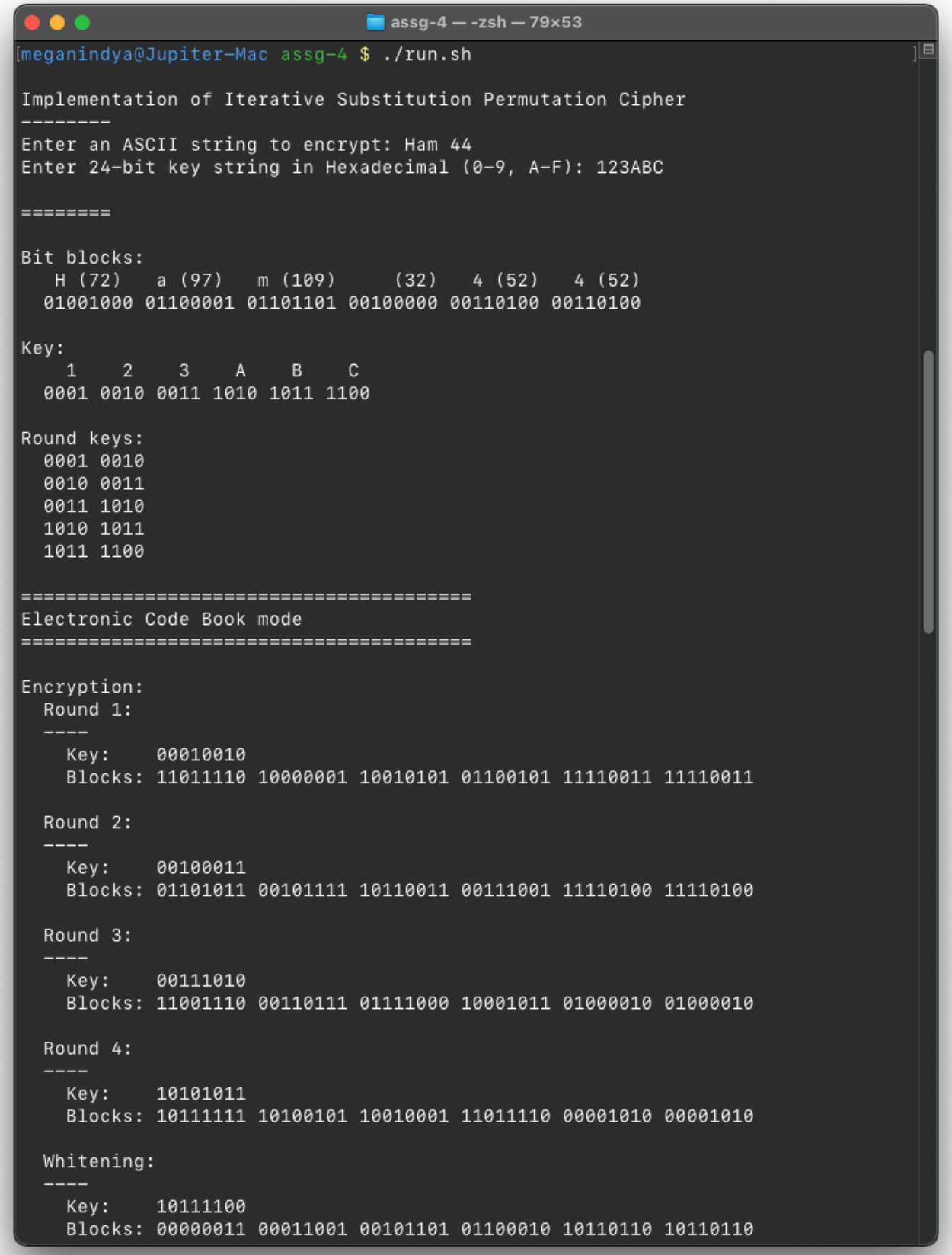
```
    printf("\n=======================================\n");
    printf("Electronic Code Book mode\n");
    printf("=======================================\n");

    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


assg-4 — -zsh — 79×53
meganindya@Jupiter-Mac assg-4 $ ./run.sh
Implementation of Iterative Substitution Permutation Cipher

Enter an ASCII string to encrypt: Ham 44
Enter 24-bit key string in Hexadecimal (0-9, A-F): 123ABC
========
Bit blocks:
H (72) a (97) m (109) (32) 4 (52) 4 (52)
01001000 01100001 01101101 00100000 00110100 00110100
Key:
1 2 3 A B C
0001 0010 0011 1010 1011 1100
Round keys:
0001 0010
0010 0011
0011 1010
1010 1011
1011 1100
==
Electronic Code Book mode
==
Encryption:
Round 1:

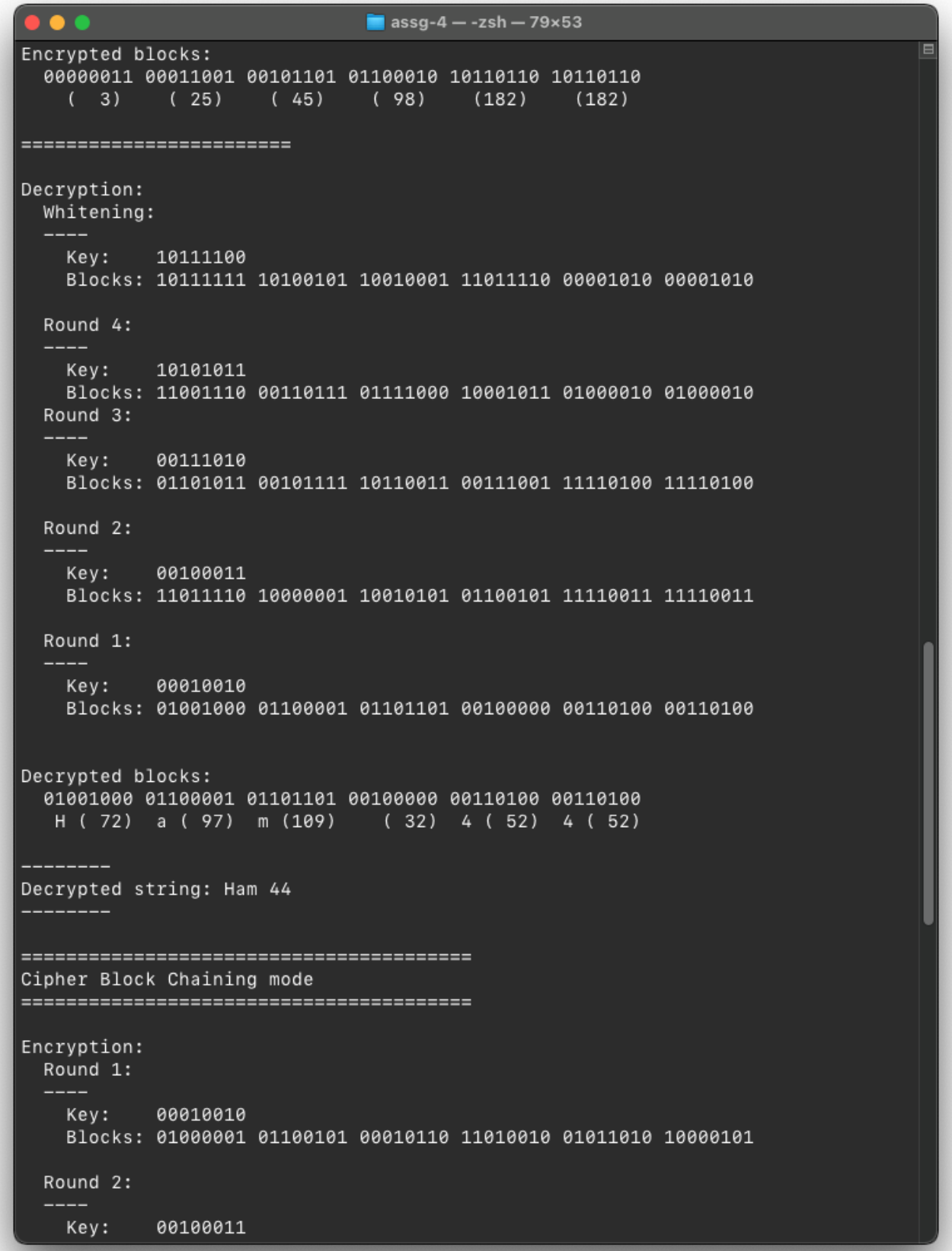
Key: 00010010
Blocks: 11011110 10000001 10010101 01100101 11110011 11110011
Round 2:

Key: 00100011
Blocks: 01101011 00101111 10110011 00111001 11110100 11110100
Round 3:

Key: 00111010
Blocks: 11001110 00110111 01111000 10001011 01000010 01000010
Round 4:

Key: 10101011
Blocks: 10111111 10100101 10010001 11011110 00001010 00001010
Whitening:

Key: 10111100
Blocks: 00000011 00011001 00101101 01100010 10110110 10110110


assg-4 — -zsh — 79×53
Encrypted blocks:
00000011 00011001 00101101 01100010 10110110 10110110
(3) (25) (45) (98) (182) (182)
==========
Decryption:
Whitening:

Key: 10111100
Blocks: 10111111 10100101 10010001 11011110 00001010 00001010
Round 4:

Key: 10101011
Blocks: 11001110 00110111 01111000 10001011 01000010 01000010
Round 3:

Key: 00111010
Blocks: 01101011 00101111 10110011 00111001 11110100 11110100
Round 2:

Key: 00100011
Blocks: 11011110 10000001 10010101 01100101 11110011 11110011
Round 1:

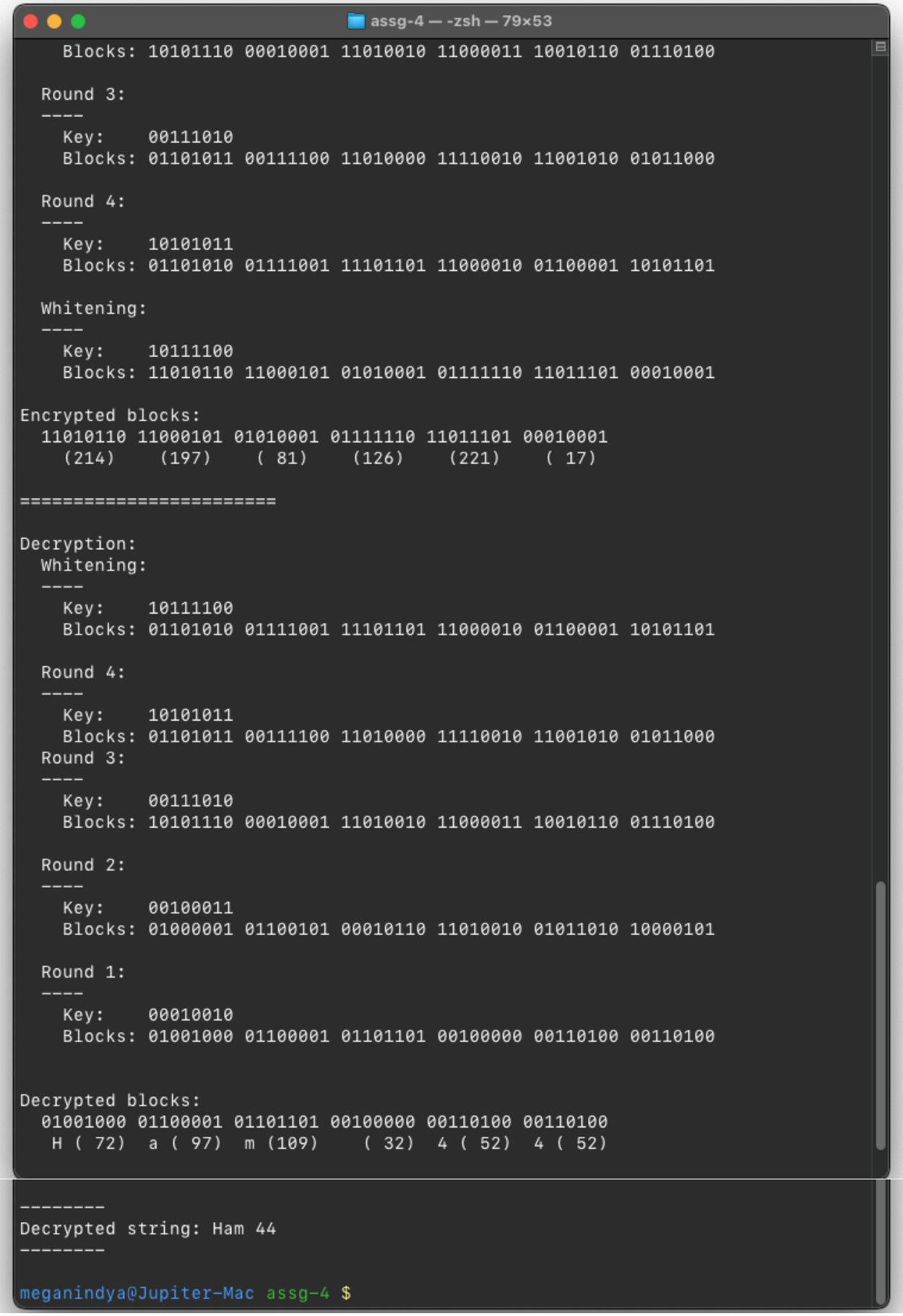
Key: 00010010
Blocks: 01001000 01100001 01101101 00100000 00110100 00110100
Decrypted blocks:
01001000 01100001 01101101 00100000 00110100 00110100
H (72) a (97) m (109) (32) 4 (52) 4 (52)

Decrypted string: Ham 44

==========
Cipher Block Chaining mode
==========
Encryption:
Round 1:

Key: 00010010
Blocks: 01000001 01100101 00010110 11010010 01011010 10000101
Round 2:

Key: 00100011


assg-4 — -zsh — 79×53
Blocks: 10101110 00010001 11010010 11000011 10010110 01110100
Round 3:
Key: 00111010
Blocks: 01101011 00111100 11010000 11110010 11001010 01011000
Round 4:
Key: 10101011
Blocks: 01101010 01111001 11101101 11000010 01100001 10101101
Whitening:
Key: 10111100
Blocks: 11010110 11000101 01010001 01111110 11011101 00010001
Encrypted blocks:
11010110 11000101 01010001 01111110 11011101 00010001
(214) (197) (81) (126) (221) (17)
==========
Decryption:
Whitening:
Key: 10111100
Blocks: 01101010 01111001 11101101 11000010 01100001 10101101
Round 4:
Key: 10101011
Blocks: 01101011 00111100 11010000 11110010 11001010 01011000
Round 3:
Key: 00111010
Blocks: 10101110 00010001 11010010 11000011 10010110 01110100
Round 2:
Key: 00100011
Blocks: 01000001 01100101 00010110 11010010 01011010 10000101
Round 1:
Key: 00010010
Blocks: 01001000 01100001 01101101 00100000 00110100 00110100
Decrypted blocks:
01001000 01100001 01101101 00100000 00110100 00110100
H (72) a (97) m (109) (32) 4 (52) 4 (52)

Decrypted string: Ham 44
----
meganindya@Jupiter-Mac assg-4 $