Assignment 3

IT851: Information and Systems Security Lab

**ANINDYA KUNDU**

IT, 8th Semester

ID 510817020

Repository:  
[github.com/meganindya/btech-assignments/information-and-systems-security/assg-3](https://github.com/meganindya/btech-assignments/tree/main/information-and-systems-security/assg-3)

1. Implement the Auto-key Cipher

Source: 1-cipher-autokey.c

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

*#include* <stdlib.h> *// abs*

*#include* <string.h> *// strlen*

*#include* "../utils.h"

*#define* MOD 26

*/\**

*\* Cipher Auto Key encrypts (in place) all input string and generates a key string.*

*\**

*\* s: the string*

*\* a: auto key*

*\* k: array to fill generated key in*

*\*/*

void encrypt(char \*s, char a, char \*k)

{

    int len = strlen(s);

    k[0] = a;

*for* (int i = 1; i < len; i++)

    {

        k[i] = s[i - 1];

    }

    k[len] = '\0';

    printf("  Key:\n");

    printf("    ");

*for* (int i = 0; i < len; i++)

    {

        printf(" %c  ", k[i]);

    }

    printf("\n    ");

*for* (int i = 0; i < len; i++)

    {

        printf("%2d  ", k[i] - 'A');

    }

    printf("\n\n");

*for* (int i = 0; i < len; i++)

    {

        char c = s[i];

        int c\_n = c - 'A';

        char r = k[i];

        int r\_n = r - 'A';

        int add\_n = c\_n + r\_n;

        int enc\_n = mod\_26(add\_n);

        char enc = enc\_n + 'A';

        printf(

            "    %c (%2d)  ->  [(%2d + %2d) mod %d] = [%2d mod %d]  %c (%2d)\n",

            c,

            c\_n,

            c\_n,

            r\_n,

            MOD,

            add\_n,

            MOD,

            enc,

            enc\_n);

        s[i] = enc;

    }

}

*/\**

*\* Cipher Auto Key decrypts (in place) all characters of a string.*

*\**

*\* s: the string*

*\* k: key string*

*\*/*

void decrypt(char \*s, char \*k)

{

    int len = strlen(k);

*for* (int i = 0; i < len; i++)

    {

        char c = s[i];

        int c\_n = c - 'A';

        char r = k[i];

        int r\_n = r - 'A';

        int sub\_n = c\_n - r\_n;

        int enc\_n = mod\_26(sub\_n);

        char enc = enc\_n + 'A';

        printf(

            "    %c (%2d)  ->  [(%2d - %2d) mod %d] = [%3d mod %d]  %c (%2d)\n",

            c,

            c\_n,

            c\_n,

            r\_n,

            MOD,

            sub\_n,

            MOD,

            enc,

            enc\_n);

        s[i] = enc;

    }

}

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

int main(int argc, char \*argv[])

{

    char s[256];

    char a;

    char k[256];

    printf("\nImplementation of Auto Key Cipher\n--------\n");

    int repeat;

*do*

    {

        printf("Enter a string (A-Z) to encrypt: ");

        scanf("%s", s);

        repeat = 0;

*for* (int i = 0; s[i] != '\0'; i++)

        {

*if* (s[i] < 'A' || s[i] > 'Z')

            {

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

                repeat = 1;

*break*;

            }

        }

    } *while* (repeat);

*do*

    {

        printf("Enter auto key: ");

        scanf(" %c", &a);

        repeat = 0;

*if* (a < 'A' || a > 'Z')

        {

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

            repeat = 1;

*break*;

        }

    } *while* (repeat);

    printf("\nEncryption:\n");

    encrypt(s, a, k);

    printf("\nEncrypted string: %s\n", s);

    printf("\nDecryption:\n");

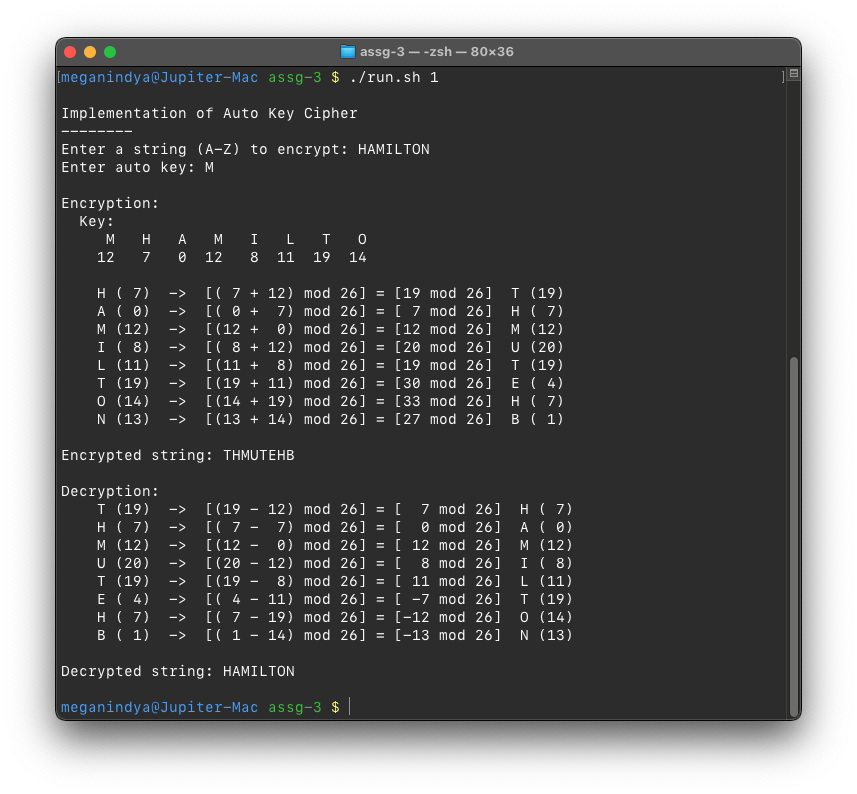
    decrypt(s, k);

    printf("\nDecrypted string: %s\n", s);

    printf("\n");

}

Sample run



2. Implement the following classic polyalphabetic ciphers (Generate the keys pseudo-randomly, check validity of the key, and store it into a key-file):

a. Vigenere Cipher

Source: 2-a-cipher-vigenere.c

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

*#include* <stdlib.h> *// abs*

*#include* <string.h> *// strlen*

*#include* "../utils.h"

*#define* MOD 26

*/\**

*\* Cipher Vigenere encrypts (in place) all input string and generates a key string.*

*\**

*\* s: the string*

*\* key: key string*

*\*/*

void encrypt(char \*s, char \*key)

{

    int len = strlen(s);

    int len\_k = strlen(key);

    char k[256];

*for* (int i = 0; i < len; i++)

    {

        k[i] = key[i % len\_k];

    }

    k[len] = '\0';

    printf("  Key:\n");

    printf("    ");

*for* (int i = 0; i < len; i++)

    {

        printf(" %c  ", k[i]);

    }

    printf("\n    ");

*for* (int i = 0; i < len; i++)

    {

        printf("%2d  ", k[i] - 'A');

    }

    printf("\n\n");

*for* (int i = 0; i < len; i++)

    {

        char c = s[i];

        int c\_n = c - 'A';

        char r = k[i];

        int r\_n = r - 'A';

        int add\_n = c\_n + r\_n;

        int enc\_n = mod\_26(add\_n);

        char enc = enc\_n + 'A';

        printf(

            "    %c (%2d)  ->  [(%2d + %2d) mod %d] = [%2d mod %d]  %c (%2d)\n",

            c,

            c\_n,

            c\_n,

            r\_n,

            MOD,

            add\_n,

            MOD,

            enc,

            enc\_n);

        s[i] = enc;

    }

}

*/\**

*\* Cipher Vigenere decrypts (in place) all characters of a string.*

*\**

*\* s: the string*

*\* key: key string*

*\*/*

void decrypt(char \*s, char \*key)

{

    int len = strlen(s);

    int len\_k = strlen(key);

    char k[256];

*for* (int i = 0; i < len; i++)

    {

        k[i] = key[i % len\_k];

    }

    k[len] = '\0';

*for* (int i = 0; i < len; i++)

    {

        char c = s[i];

        int c\_n = c - 'A';

        char r = k[i];

        int r\_n = r - 'A';

        int sub\_n = c\_n - r\_n;

        int enc\_n = mod\_26(sub\_n);

        char enc = enc\_n + 'A';

        printf(

            "    %c (%2d)  ->  [(%2d - %2d) mod %d] = [%3d mod %d]  %c (%2d)\n",

            c,

            c\_n,

            c\_n,

            r\_n,

            MOD,

            sub\_n,

            MOD,

            enc,

            enc\_n);

        s[i] = enc;

    }

}

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

int main(int argc, char \*argv[])

{

    char s[256];

    char k[256];

    printf("\nImplementation of Vigenere Cipher\n--------\n");

    int repeat;

*do*

    {

        printf("Enter a string (A-Z) to encrypt: ");

        scanf("%s", s);

        repeat = 0;

*for* (int i = 0; s[i] != '\0'; i++)

        {

*if* (s[i] < 'A' || s[i] > 'Z')

            {

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

                repeat = 1;

*break*;

            }

        }

    } *while* (repeat);

*do*

    {

        printf("Enter key: ");

        scanf(" %s", k);

        repeat = 0;

*for* (int i = 0; k[i] != '\0'; i++)

        {

*if* (k[i] < 'A' || k[i] > 'Z')

            {

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

                repeat = 1;

*break*;

            }

        }

    } *while* (repeat);

    printf("\nEncryption:\n");

    encrypt(s, k);

    printf("\nEncrypted string: %s\n", s);

    printf("\nDecryption:\n");

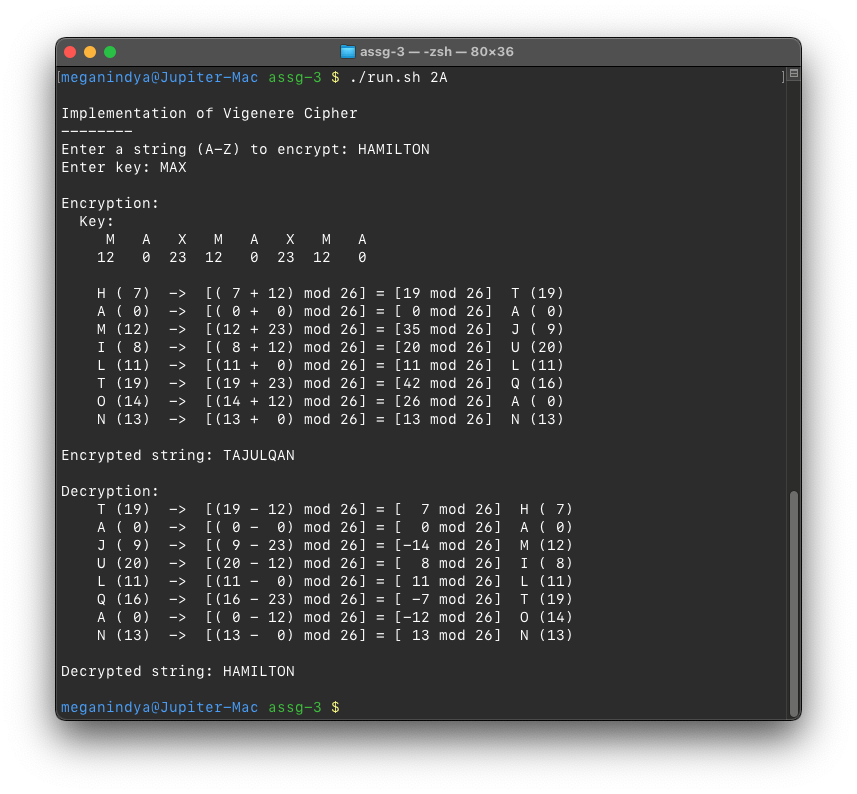
    decrypt(s, k);

    printf("\nDecrypted string: %s\n", s);

    printf("\n");

}

Sample run



b. Keyed Transposition Cipher (assume block size to be 5)

Source: 2-b-cipher-keyed-transposition.c

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

*#include* <stdlib.h> *// abs*

*#include* <string.h> *// strlen*

*#include* "../utils.h"

*#define* MOD 26

*#define* BLK 5

*/\**

*\* Cipher keyed-transposition encrypts (in place) all input string and generates a key string.*

*\**

*\* s: the string*

*\* k: key string*

*\*/*

void encrypt(char \*s, char \*k)

{

    int len = strlen(s);

    int rlen = len / BLK + (len % BLK == 0 ? 0 : 1);

    char mat[rlen][BLK];

*for* (int r = 0; r < rlen; r++)

    {

*for* (int c = 0; c < BLK; c++)

        {

            int pos = r \* BLK + c;

            mat[r][c] = pos < len ? s[pos] : 'Z';

        }

    }

    printf("    ");

*for* (int i = 0; i < BLK; i++)

    {

        printf("%d  ", i + 1);

    }

    printf("\n\n");

    printf("  Initial:\n");

*for* (int r = 0; r < rlen; r++)

    {

        printf("    ");

*for* (int c = 0; c < BLK; c++)

        {

            printf("%c  ", mat[r][c]);

        }

        printf("\n");

    }

    int trx[rlen][BLK];

*for* (int i = 0; i < BLK; i++)

    {

        int c = k[i] - '1';

*for* (int j = 0; j < rlen; j++)

        {

            trx[j][i] = mat[j][c];

        }

    }

    printf("\n  Transposing:\n");

*for* (int r = 0; r < rlen; r++)

    {

        printf("    ");

*for* (int c = 0; c < BLK; c++)

        {

            printf("%c  ", trx[r][c]);

        }

        printf("\n");

    }

*for* (int r = 0; r < rlen; r++)

    {

*for* (int c = 0; c < BLK; c++)

        {

            s[r \* BLK + c] = trx[r][c];

        }

    }

}

*/\**

*\* Cipher keyed-transposition decrypts (in place) all characters of a string.*

*\**

*\* s: the string*

*\* k: key string*

*\*/*

void decrypt(char \*s, char \*k)

{

    int len = strlen(s);

    int rlen = len / BLK + (len % BLK == 0 ? 0 : 1);

    char mat[rlen][BLK];

*for* (int r = 0; r < rlen; r++)

    {

*for* (int c = 0; c < BLK; c++)

        {

            mat[r][c] = s[r \* BLK + c];

        }

    }

    printf("    ");

*for* (int i = 0; i < BLK; i++)

    {

        printf("%d  ", i + 1);

    }

    printf("\n\n");

    printf("  Initial:\n");

*for* (int r = 0; r < rlen; r++)

    {

        printf("    ");

*for* (int c = 0; c < BLK; c++)

        {

            printf("%c  ", mat[r][c]);

        }

        printf("\n");

    }

    char ki[BLK];

*for* (int i = 0; i < BLK; i++)

    {

        ki[k[i] - '1'] = i + '1';

    }

    printf("\n  Key Inverse: %s\n", ki);

    int trx[rlen][BLK];

*for* (int i = 0; i < BLK; i++)

    {

        int c = ki[i] - '1';

*for* (int j = 0; j < rlen; j++)

        {

            trx[j][i] = mat[j][c];

        }

    }

    printf("\n  Inverse Transposing:\n");

*for* (int r = 0; r < rlen; r++)

    {

        printf("    ");

*for* (int c = 0; c < BLK; c++)

        {

            printf("%c  ", trx[r][c]);

        }

        printf("\n");

    }

*for* (int r = 0; r < rlen; r++)

    {

*for* (int c = 0; c < BLK; c++)

        {

            s[r \* BLK + c] = trx[r][c];

        }

    }

}

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

int main(int argc, char \*argv[])

{

    char s[256];

    char k[5];

    printf("\nImplementation of Keyed Transposition Cipher\n--------\n");

    int repeat;

*do*

    {

        printf("Enter a string (A-Z) to encrypt: ");

        scanf("%s", s);

        repeat = 0;

*for* (int i = 0; s[i] != '\0'; i++)

        {

*if* (s[i] < 'A' || s[i] > 'Z')

            {

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

                repeat = 1;

*break*;

            }

        }

    } *while* (repeat);

*do*

    {

        printf("Enter key: ");

        scanf(" %s", k);

*if* (strlen(k) != BLK)

        {

            repeat = 1;

*continue*;

        }

        repeat = 0;

*for* (int i = 0; k[i] != '\0'; i++)

        {

*if* (k[i] < '1' || k[i] > '5')

            {

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

                repeat = 1;

*break*;

            }

        }

    } *while* (repeat);

    printf("\nEncryption:\n");

    encrypt(s, k);

    printf("\nEncrypted string: %s\n", s);

    printf("\nDecryption:\n");

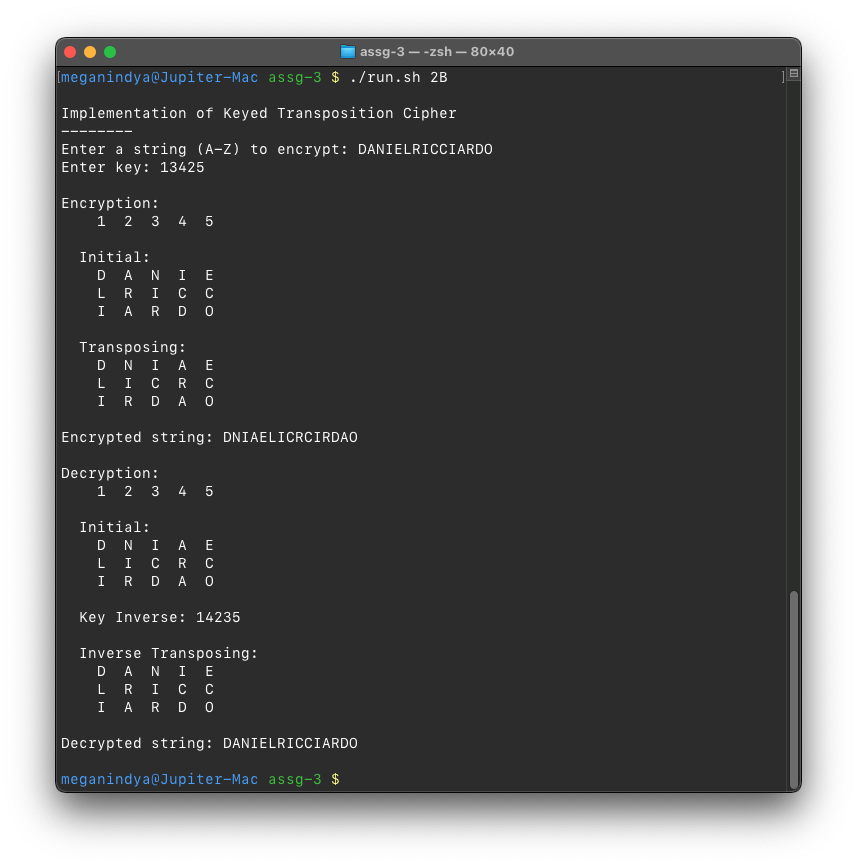
    decrypt(s, k);

    printf("\nDecrypted string: %s\n", s);

    printf("\n");

}

Sample run



1. Modify the Hill Cipher program you wrote for Assignment 1, so as to implement the following Permutation Cipher: (Following π is a permutation of plain text letters positioned at {1, …, 8})

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| x | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| π (x) | 4 | 1 | 6 | 2 | 7 | 3 | 8 | 5 |

Test the operation of your encryption and decryption programs using the above π and its corresponding π. Hence decrypt the following cipher text, which was encrypted using the above π:

TGEEMNELNNTDROEOAAHDOETCSHAEIRLM

HINT: The key of a transposition cipher may be represented as a matrix of zeros and ones.

Source: 3-cipher-permutation.c

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

*#include* <stdlib.h> *// abs, srand, rand*

*#include* <string.h> *// strlen*

*#include* <time.h>   *// time*

*#include* "../utils.h"

*#define* MOD 26

*/\**

*\* Cipher permutation encrypts (in place) all characters of a string w.r.t a key.*

*\**

*\* s: the string*

*\* ord\_key: order of cipher key matrix*

*\* key: cipher key matrix*

*\*/*

void encrypt(char \*s, int ord\_key, int \*\*key)

{

    int len = strlen(s);

    int rlen = len / ord\_key + (len % ord\_key == 0 ? 0 : 1);

    int mat[rlen][ord\_key];

*for* (int r = 0; r < rlen; r++)

    {

*for* (int c = 0; c < ord\_key; c++)

        {

            int pos = r \* ord\_key + c;

            mat[r][c] = pos < len ? s[pos] - 'A' : 25;

        }

    }

    printf("  Initial:\n");

*for* (int r = 0; r < rlen; r++)

    {

        printf("    ");

*for* (int c = 0; c < ord\_key; c++)

        {

            printf("%c (%2d)  ", mat[r][c] + 'A', mat[r][c]);

        }

        printf("\n");

    }

    int temp[rlen][ord\_key];

    printf("\n  Multiplied:\n");

*for* (int r = 0; r < rlen; r++)

    {

*for* (int c = 0; c < ord\_key; c++)

        {

            int sum = 0;

*for* (int m = 0; m < ord\_key; m++)

            {

                sum += mat[r][m] \* key[m][c];

            }

            temp[r][c] = sum;

        }

    }

*for* (int r = 0; r < rlen; r++)

    {

        printf("    ");

*for* (int c = 0; c < ord\_key; c++)

        {

            printf("%c (%2d)  ", temp[r][c] + 'A', temp[r][c]);

        }

        printf("\n");

    }

*for* (int r = 0; r < rlen; r++)

    {

*for* (int c = 0; c < ord\_key; c++)

        {

            s[r \* ord\_key + c] = temp[r][c] + 'A';

        }

    }

    s[ord\_key \* rlen] = '\0';

}

*/\**

*\* Cipher permutation decrypts (in place) all characters of a string w.r.t a key.*

*\**

*\* s: the string*

*\* ord\_key: order of cipher key matrix*

*\* key: cipher key matrix*

*\*/*

void decrypt(char \*s, int ord\_key, int \*\*key)

{

    int len = strlen(s);

    int rlen = len / ord\_key + (len % ord\_key == 0 ? 0 : 1);

    int mat[rlen][ord\_key];

*for* (int r = 0; r < rlen; r++)

    {

*for* (int c = 0; c < ord\_key; c++)

        {

            mat[r][c] = s[r \* ord\_key + c] - 'A';

        }

    }

    int temp[rlen][ord\_key];

    char k[ord\_key];

*for* (int c = 0; c < ord\_key; c++)

    {

*for* (int r = 0; r < ord\_key; r++)

        {

*if* (key[r][c] == 1)

            {

                k[c] = r + '1';

*break*;

            }

        }

    }

    char ki[ord\_key];

*for* (int i = 0; i < ord\_key; i++)

    {

        ki[k[i] - '1'] = i + '1';

    }

    int \*\*key\_inv;

    key\_inv = malloc((ord\_key) \* sizeof \*key\_inv);

*for* (int i = 0; i < ord\_key; i++)

    {

        key\_inv[i] = malloc(ord\_key \* sizeof \*key\_inv[i]);

    }

*// mat\_invert(ord\_key, key, key\_inv);*

*for* (int c = 0; c < ord\_key; c++)

    {

*for* (int r = 0; r < ord\_key; r++)

        {

            key\_inv[r][c] = 0;

        }

    }

*for* (int i = 0; i < ord\_key; i++)

    {

        key\_inv[ki[i] - '1'][i] = 1;

    }

    printf("  Inverted key:\n");

*for* (int r = 0; r < ord\_key; r++)

    {

        printf("    ");

*for* (int c = 0; c < ord\_key; c++)

        {

            printf("%3d ", key\_inv[r][c]);

        }

        printf("\n");

    }

    printf("\n  Multiplied:\n");

*for* (int r = 0; r < rlen; r++)

    {

*for* (int c = 0; c < ord\_key; c++)

        {

            int sum = 0;

*for* (int m = 0; m < ord\_key; m++)

            {

                sum += mat[r][m] \* key\_inv[m][c];

            }

            temp[r][c] = sum;

        }

    }

*for* (int r = 0; r < rlen; r++)

    {

        printf("    ");

*for* (int c = 0; c < ord\_key; c++)

        {

            printf("%c (%2d)  ", temp[r][c] + 'A', temp[r][c]);

        }

        printf("\n");

    }

*for* (int r = 0; r < rlen; r++)

    {

*for* (int c = 0; c < ord\_key; c++)

        {

            s[r \* ord\_key + c] = temp[r][c] + 'A';

        }

    }

*for* (int i = 0; i < ord\_key; i++)

    {

        free(key\_inv[i]);

    }

    free(key\_inv);

}

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

int main(int argc, char \*argv[])

{

    char s[256];

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

    int repeat;

*do*

    {

        printf("Enter a string (A-Z) to encrypt: ");

        scanf("%s", s);

        repeat = 0;

*for* (int i = 0; s[i] != '\0'; i++)

        {

*if* (s[i] < 'A' || s[i] > 'Z')

            {

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

                repeat = 1;

*break*;

            }

        }

    } *while* (repeat);

    char ks[] = "41627385";

    int ord\_key = 8;

    int \*\*key;

    key = malloc((ord\_key) \* sizeof \*key);

*for* (int i = 0; i < ord\_key; i++)

    {

        key[i] = malloc(ord\_key \* sizeof \*key[i]);

    }

*for* (int i = 0; i < ord\_key; i++)

    {

*for* (int j = 0; j < ord\_key; j++)

        {

            key[i][j] = 0;

        }

    }

*for* (int i = 0; i < ord\_key; i++)

    {

        key[ks[i] - '1'][i] = 1;

    }

    printf("\nKey:\n");

*for* (int i = 0; i < ord\_key; i++)

    {

*for* (int j = 0; j < ord\_key; j++)

        {

            printf("%3d ", key[i][j]);

        }

        printf("\n");

    }

    printf("\nEncryption:\n");

    encrypt(s, ord\_key, key);

    printf("\nEncrypted string: %s\n", s);

    printf("\nDecryption:\n");

    decrypt(s, ord\_key, key);

    printf("\nDecrypted string: %s\n", s);

    printf("\n--------\n\n");

    char sx[256] = "TGEEMNELNNTDROEOAAHDOETCSHAEIRLM";

    printf("For encrypted string: %s", sx);

    printf("\nDecryption:\n");

    decrypt(sx, ord\_key, key);

    printf("\nDecrypted string: %s\n", sx);

    printf("\n");

*for* (int i = 0; i < ord\_key; i++)

    {

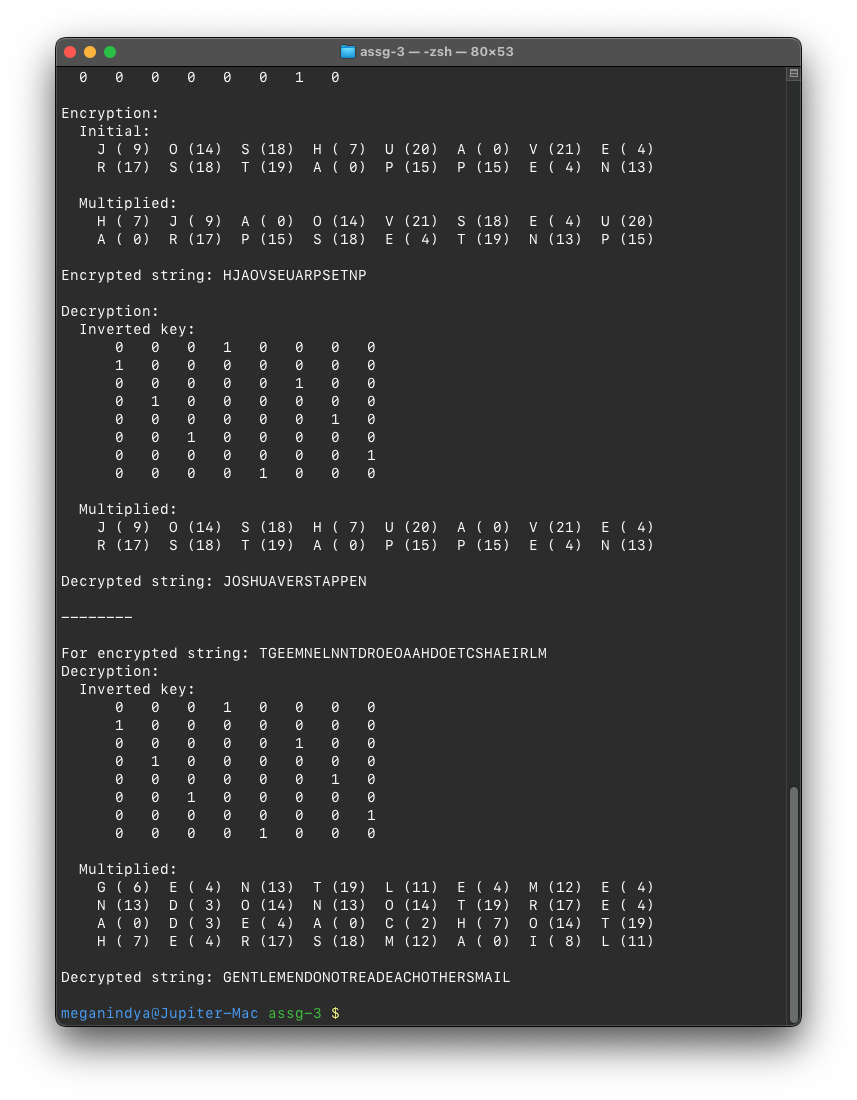
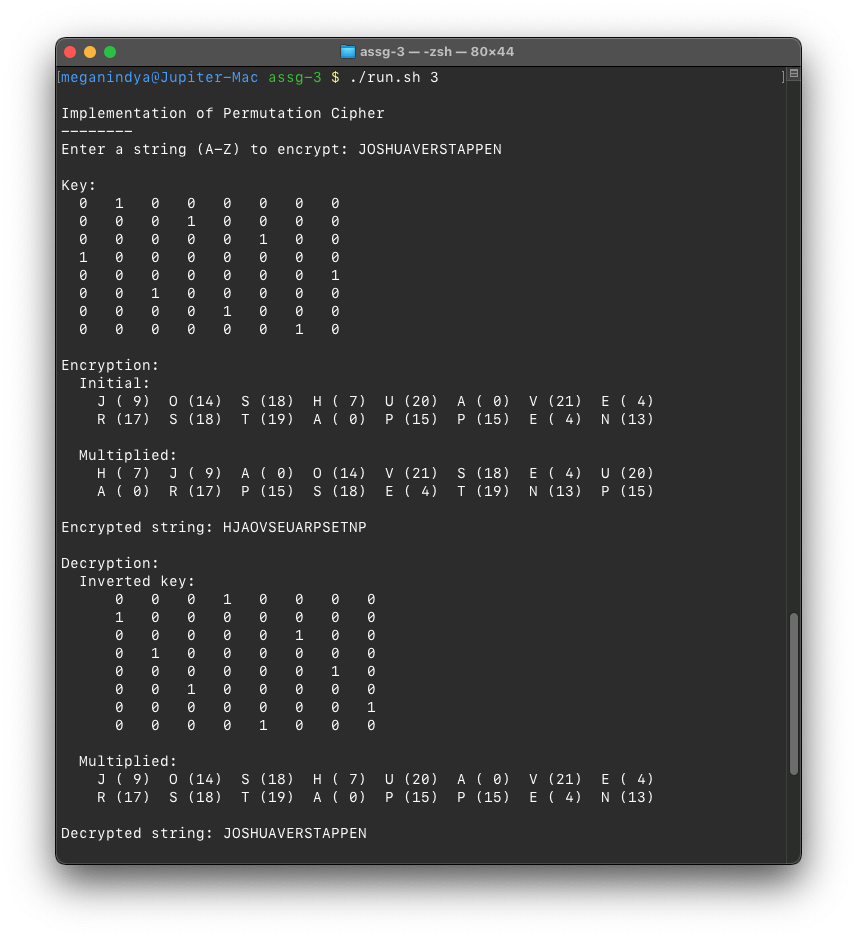
        free(key[i]);

    }

    free(key);

}

Sample run



1. A One-Time Pad (OTP) is a stream cipher which uses True Random Number Generator (TRNG) to generate its key-stream, hence the name OTP. It uses the XOR-operation as both encryption and decryption functions.

a) Implement an OTP.

(You may use any Pseudo-Random Number Generator (PRNG) to generate the key-stream here considering that following a physical process is infeasible for this laboratory.)

Source: 4-a-cipher-one-time-pad.c

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

*#include* <stdlib.h> *// abs*

*#include* <string.h> *// strlen*

*#include* <time.h>   *// time*

*#include* "../utils.h"

*#define* MOD 26

*/\**

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

*\* There is a condition that the integers represent alphabets and are therefore confined in the*

*\* range [0, 25], which can be covered in 5 bits.*

*\**

*\* n: integer number*

*\* s: character array to fill (binary) bits in (array length assumed to be 5)*

*\*/*

void int\_to\_binary(int n, char \*s)

{

    int mask = 1;

*for* (int i = 0; i < 5; i++)

    {

        s[4 - i] = (n & mask) == 0 ? '0' : '1';

        mask <<= 1;

    }

}

*/\**

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

*\* There is a condition that the integers represent alphabets and are therefore confined in the*

*\* range [0, 25], which can be covered in 5 bits.*

*\**

*\* s: character array of bits representing the binary (array length assumed to be 5)*

*\**

*\* returns:*

*\* integer (base 10) equivalent*

*\*/*

int binary\_to\_int(char \*s)

{

    int n = 0, mask = 1;

*for* (int i = 0; i < 5; i++)

    {

        n += (s[4 - 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 (char a, char b) {

*return* a == b ? '0' : '1';

}

*/\**

*\* Cipher One Time Pad encrypts (in place) all input string and generates a key string.*

*\**

*\* s: the string*

*\* k: cipher key*

*\*/*

    void encrypt(char \*s, char \*k)

{

    int len\_s = strlen(s);

    int len\_k = strlen(k);

    printf("  ");

*for* (int i = 0; i < 5; i++)

    {

        printf(" (%c)  ", s[i]);

    }

    printf("\n  ");

    char a[len\_k];

*for* (int i = 0; i < len\_s; i++)

    {

        int x = s[i] - 'A';

        char bin[5];

        int\_to\_binary(x, bin);

        printf("%s ", bin);

*for* (int j = i \* 5; j < (i + 1) \* 5; j++)

        {

            a[j] = bin[j - (i \* 5)];

        }

    }

    printf("\n  ");

*for* (int i = 0; i < len\_s; i++)

    {

*for* (int j = i \* 5; j < (i + 1) \* 5; j++)

        {

            printf("%c", k[j]);

        }

        printf(" ");

    }

    printf("\n  ");

*for* (int i = 1; i < 6 \* len\_s; i++)

    {

        printf("-");

    }

*for* (int i = 0; i < len\_k; i++)

    {

        a[i] = xor(a[i], k[i]);

    }

    printf("\n  ");

*for* (int i = 0; i < len\_s; i++)

    {

*for* (int j = i \* 5; j < (i + 1) \* 5; j++)

        {

            printf("%c", a[j]);

        }

        printf(" ");

    }

*for* (int i = 0; i < len\_s; i++)

    {

        char bin[5];

*for* (int j = i \* 5; j < (i + 1) \* 5; j++)

        {

            bin[j - (i \* 5)] = a[j];

        }

        s[i] = binary\_to\_int(bin) + 'A';

    }

    printf("\n  ");

*for* (int i = 0; i < len\_s; i++)

    {

        printf(" (%2d) ", s[i] - 'A');

    }

    printf("\n");

}

*/\**

*\* Cipher One Time Pad decrypts (in place) all characters of a string.*

*\**

*\* s: the string*

*\* k: cipher key*

*\*/*

void decrypt(char \*s, char \*k)

{

    int len\_s = strlen(s);

    int len\_k = strlen(k);

    printf("  ");

*for* (int i = 0; i < len\_s; i++)

    {

        printf(" (%2d) ", s[i] - 'A');

    }

    printf("\n  ");

    char a[len\_k];

*for* (int i = 0; i < len\_s; i++)

    {

        int x = s[i] - 'A';

        char bin[5];

        int\_to\_binary(x, bin);

*for* (int j = i \* 5; j < (i + 1) \* 5; j++)

        {

            a[j] = bin[j - (i \* 5)];

            printf("%c", a[j]);

        }

        printf(" ");

    }

    printf("\n  ");

*for* (int i = 0; i < len\_s; i++)

    {

*for* (int j = i \* 5; j < (i + 1) \* 5; j++)

        {

            printf("%c", k[j]);

        }

        printf(" ");

    }

    printf("\n  ");

*for* (int i = 1; i < 6 \* len\_s; i++)

    {

        printf("-");

    }

*for* (int i = 0; i < len\_k; i++)

    {

        a[i] = xor(a[i], k[i]);

    }

    printf("\n  ");

*for* (int i = 0; i < len\_s; i++)

    {

*for* (int j = i \* 5; j < (i + 1) \* 5; j++)

        {

            printf("%c", a[j]);

        }

        printf(" ");

    }

    printf("\n  ");

*for* (int i = 0; i < len\_s; i++)

    {

        char bin[5];

*for* (int j = i \* 5; j < (i + 1) \* 5; j++)

        {

            bin[j - (i \* 5)] = a[j];

        }

        s[i] = binary\_to\_int(bin) + 'A';

    }

*for* (int i = 0; i < len\_s; i++)

    {

        printf(" (%c)  ", s[i]);

    }

    printf("\n");

}

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

int main(int argc, char \*argv[])

{

    char s[32];

    char k[256];

    printf("\nImplementation of One Time Pad Cipher\n--------\n");

    int repeat;

*do*

    {

        printf("Enter a string (A-Z) to encrypt: ");

        scanf("%s", s);

        repeat = 0;

*for* (int i = 0; s[i] != '\0'; i++)

        {

*if* (s[i] < 'A' || s[i] > 'Z')

            {

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

                repeat = 1;

*break*;

            }

        }

    } *while* (repeat);

    srand(time(0));

*for* (int i = 0; i < strlen(s); i++)

    {

*for* (int j = i \* 5; j < (i + 1) \* 5; j++)

        {

            k[j] = '0' + rand() % 2;

        }

    }

    printf("\nPseudorandom One Time Key:\n  ");

*for* (int i = 0; i < strlen(s); i++)

    {

*for* (int j = i \* 5; j < (i + 1) \* 5; j++)

        {

            printf("%c", k[j]);

        }

        printf(" ");

    }

    printf("\n");

    printf("\nEncryption:\n");

    encrypt(s, k);

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

*for* (int i = 0; i < strlen(s); i++)

    {

        char bin[5];

        int\_to\_binary(s[i] - 'A', bin);

*for* (int j = i \* 5; j < (i + 1) \* 5; j++)

        {

            printf("%c", bin[j - (i \* 5)]);

        }

        printf(" ");

    }

    printf("\n");

    printf("\nDecryption:\n");

    decrypt(s, k);

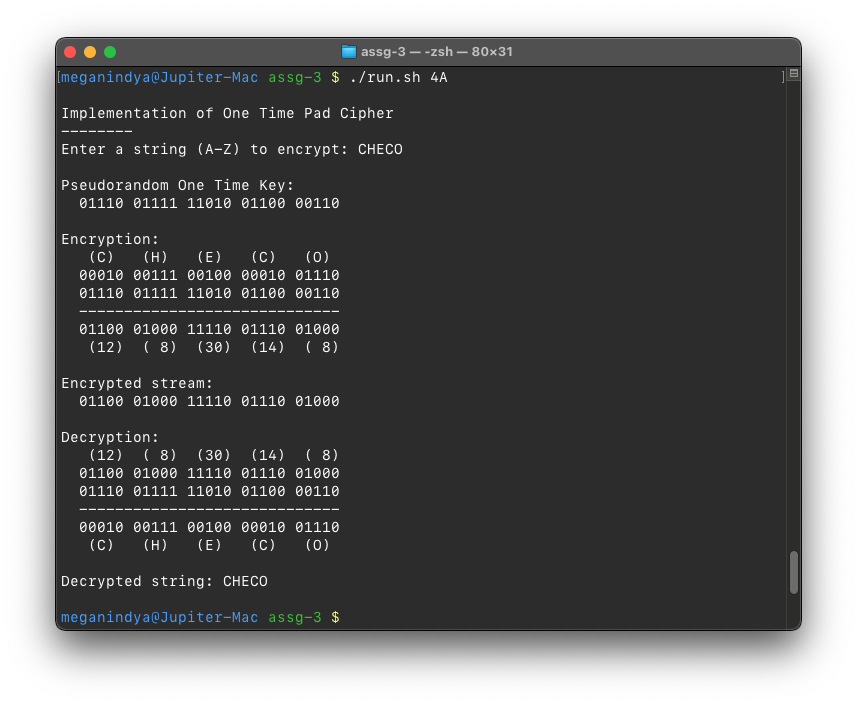
    printf("\nDecrypted string: %s\n", s);

    printf("\n");

}

[P.T.O.]

Sample run



b) Assuming the PRNG looks like:

*S0 = seed*

*Si*+1 ≡ *ASi* + *B* mod *m, i* = 0*,*1*, . . .*

where *m* =26 is public, the secrets are *A*, *B* and the seed, where all *A*, *B, Si* belong to Z26, and an outsider is provided with the knowledge of only first 15 bits of plaintext, implement a way for (known-plaintext) cryptanalysis of the OTP.

[Hint: Note that 24 < 26 < 25]

Source: 4-b-cipher-one-time-pad-cryptanalysis.c

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

*#include* <stdlib.h> *// abs*

*#include* <string.h> *// strlen*

*#include* "../utils.h"

*#define* MOD 26

*/\**

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

*\* There is a condition that the integers represent alphabets and are therefore confined in the*

*\* range [0, 25], which can be covered in 5 bits.*

*\**

*\* n: integer number*

*\* s: character array to fill (binary) bits in (array length assumed to be 5)*

*\*/*

void int\_to\_binary(int n, char \*s)

{

    int mask = 1;

*for* (int i = 0; i < 5; i++)

    {

        s[4 - i] = (n & mask) == 0 ? '0' : '1';

        mask <<= 1;

    }

}

*/\**

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

*\* There is a condition that the integers represent alphabets and are therefore confined in the*

*\* range [0, 25], which can be covered in 5 bits.*

*\**

*\* s: character array of bits representing the binary (array length assumed to be 5)*

*\**

*\* returns:*

*\* integer (base 10) equivalent*

*\*/*

int binary\_to\_int(char \*s)

{

    int n = 0, mask = 1;

*for* (int i = 0; i < 5; i++)

    {

        n += (s[4 - 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 (char a, char b) {

*return* a == b ? '0' : '1';

}

*/\**

*\* Cipher One Time Pad encrypts (in place) all input string and generates a key string.*

*\**

*\* s: the string*

*\* k: cipher key*

*\*/*

    void encrypt(char \*s, char \*k)

{

    int len\_s = strlen(s);

    int len\_k = strlen(k);

    printf("  ");

*for* (int i = 0; i < 5; i++)

    {

        printf(" (%c)  ", s[i]);

    }

    printf("\n  ");

    char a[len\_k];

*for* (int i = 0; i < len\_s; i++)

    {

        int x = s[i] - 'A';

        char bin[5];

        int\_to\_binary(x, bin);

        printf("%s ", bin);

*for* (int j = i \* 5; j < (i + 1) \* 5; j++)

        {

            a[j] = bin[j - (i \* 5)];

        }

    }

    printf("\n  ");

*for* (int i = 0; i < len\_s; i++)

    {

*for* (int j = i \* 5; j < (i + 1) \* 5; j++)

        {

            printf("%c", k[j]);

        }

        printf(" ");

    }

    printf("\n  ");

*for* (int i = 1; i < 6 \* len\_s; i++)

    {

        printf("-");

    }

*for* (int i = 0; i < len\_k; i++)

    {

        a[i] = xor(a[i], k[i]);

    }

    printf("\n  ");

*for* (int i = 0; i < len\_s; i++)

    {

*for* (int j = i \* 5; j < (i + 1) \* 5; j++)

        {

            printf("%c", a[j]);

        }

        printf(" ");

    }

*for* (int i = 0; i < len\_s; i++)

    {

        char bin[5];

*for* (int j = i \* 5; j < (i + 1) \* 5; j++)

        {

            bin[j - (i \* 5)] = a[j];

        }

        s[i] = binary\_to\_int(bin) + 'A';

    }

    printf("\n  ");

*for* (int i = 0; i < len\_s; i++)

    {

        printf(" (%2d) ", s[i] - 'A');

    }

    printf("\n");

}

*/\**

*\* Cipher One Time Pad decrypts (in place) all characters of a string.*

*\**

*\* s: the string*

*\* k: cipher key*

*\*/*

void decrypt(char \*s, char \*k)

{

    int len\_s = strlen(s);

    int len\_k = strlen(k);

    printf("  ");

*for* (int i = 0; i < len\_s; i++)

    {

        printf(" (%2d) ", s[i] - 'A');

    }

    printf("\n  ");

    char a[len\_k];

*for* (int i = 0; i < len\_s; i++)

    {

        int x = s[i] - 'A';

        char bin[5];

        int\_to\_binary(x, bin);

*for* (int j = i \* 5; j < (i + 1) \* 5; j++)

        {

            a[j] = bin[j - (i \* 5)];

            printf("%c", a[j]);

        }

        printf(" ");

    }

    printf("\n  ");

*for* (int i = 0; i < len\_s; i++)

    {

*for* (int j = i \* 5; j < (i + 1) \* 5; j++)

        {

            printf("%c", k[j]);

        }

        printf(" ");

    }

    printf("\n  ");

*for* (int i = 1; i < 6 \* len\_s; i++)

    {

        printf("-");

    }

*for* (int i = 0; i < len\_k; i++)

    {

        a[i] = xor(a[i], k[i]);

    }

    printf("\n  ");

*for* (int i = 0; i < len\_s; i++)

    {

*for* (int j = i \* 5; j < (i + 1) \* 5; j++)

        {

            printf("%c", a[j]);

        }

        printf(" ");

    }

    printf("\n  ");

*for* (int i = 0; i < len\_s; i++)

    {

        char bin[5];

*for* (int j = i \* 5; j < (i + 1) \* 5; j++)

        {

            bin[j - (i \* 5)] = a[j];

        }

        s[i] = binary\_to\_int(bin) + 'A';

    }

*for* (int i = 0; i < len\_s; i++)

    {

        printf(" (%c)  ", s[i]);

    }

    printf("\n");

}

*/\**

*\* Cryptanalyses One Time Pad from first 3 blocks of plaintext and ciphertext and generates key*

*\* stream of same length as full plaintext for decryption.*

*\**

*\* s: cipher string*

*\* p: 3 blocks (15 bits) of plaintext*

*\* k: array to fill generated key stream in*

*\*/*

void cryptanalyze(char \*s, char \*p, char \*k)

{

    int len\_s = strlen(s);

    int len\_p = strlen(p);

    printf("  XOR-ing:\n    ");

    char r[len\_p];

    int r\_n[len\_p];

*for* (int i = 0; i < 3; i++)

    {

        int x = s[i] - 'A';

        char bin[5];

        int\_to\_binary(x, bin);

*for* (int j = i \* 5; j < (i + 1) \* 5; j++)

        {

            r[j] = bin[j - (i \* 5)];

            printf("%c", r[j]);

        }

        printf(" ");

    }

    printf("\n    ");

*for* (int i = 0; i < 3; i++)

    {

*for* (int j = i \* 5; j < (i + 1) \* 5; j++)

        {

            printf("%c", p[j]);

        }

        printf(" ");

    }

    printf("\n    ");

*for* (int i = 1; i < 6 \* 3; i++)

    {

        printf("-");

    }

*for* (int i = 0; i < len\_p; i++)

    {

        r[i] = xor(r[i], p[i]);

    }

*for* (int i = 0; i < 3; i++)

    {

        char bin[5];

*for* (int j = i \* 5; j < (i + 1) \* 5; j++)

        {

            bin[j - (i \* 5)] = r[j];

        }

        r\_n[i] = binary\_to\_int(bin);

    }

    printf("\n    ");

*for* (int i = 0; i < 3; i++)

    {

*for* (int j = i \* 5; j < (i + 1) \* 5; j++)

        {

            printf("%c", r[j]);

        }

        printf(" ");

    }

    printf("\n    ");

*for* (int i = 0; i < 3; i++)

    {

        printf(" (%2d) ", r\_n[i]);

    }

    printf("\n\n  Keys:\n");

    printf("    s\_0 = %2d\n", r\_n[0]);

    printf("    s\_1 = %2d\n", r\_n[1]);

    printf("    s\_2 = %2d\n", r\_n[2]);

    int s0 = r\_n[0];

    int s1 = r\_n[1];

    int s2 = r\_n[2];

    printf("\n  Equations:\n");

    printf("    (a \* %2d + b) mod %d = %d\n", s0, MOD, s1);

    printf("    (a \* %2d + b) mod %d = %d\n", s1, MOD, s2);

    int a, b;

*if* (mod\_26\_mul\_inv(s1 - s0) == -1)

    {

        printf("\n  Cryptanalysis failed!\n\n");

        exit(-1);

    }

    a = mod\_26(mod\_26\_mul\_inv(s1 - s0) \* mod\_26(s2 - s1));

    b = mod\_26(mod\_26\_mul\_inv(s1 - s0) \* mod\_26(s1 \* s1 - s0 \* s2));

    printf("\n  Solution:\n");

    printf("    a = %d\n    b = %d\n", a, b);

    int random\_s = s0, k\_n[len\_s];

*for* (int i = 0; i < strlen(s); i++)

    {

        k\_n[i] = random\_s;

        char bin[5];

        int\_to\_binary(random\_s, bin);

*for* (int j = i \* 5; j < (i + 1) \* 5; j++)

        {

            k[j] = bin[j - (i \* 5)];

        }

        random\_s = mod\_26(a \* random\_s + b);

    }

    printf("\n  Key Stream:\n    ");

*for* (int i = 0; i < strlen(s); i++)

    {

        printf(" (%2d) ", k\_n[i]);

    }

    printf("\n    ");

*for* (int i = 0; i < strlen(s); i++)

    {

*for* (int j = i \* 5; j < (i + 1) \* 5; j++)

        {

            printf("%c", k[j]);

        }

        printf(" ");

    }

    printf("\n");

}

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

int main(int argc, char \*argv[])

{

    char s[32];

    int s0, a, b;

    char k[256];

    printf("\nImplementation of One Time Pad Cipher\n--------\n");

    int repeat;

*do*

    {

        printf("Enter a string (A-Z) to encrypt: ");

        scanf("%s", s);

        repeat = 0;

*for* (int i = 0; s[i] != '\0'; i++)

        {

*if* (s[i] < 'A' || s[i] > 'Z')

            {

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

                repeat = 1;

*break*;

            }

        }

    } *while* (repeat);

    char bin\_s[15];

*for* (int i = 0; i < 3; i++)

    {

        char bin[5];

        int\_to\_binary(s[i] - 'A', bin);

*for* (int j = i \* 5; j < (i + 1) \* 5; j++)

        {

            bin\_s[j] = bin[j - (i \* 5)];

        }

    }

    printf("\nEnter parameters for PRNG (s\_i+1 = a \* s\_i + b mod 26):\n");

    printf("  seed (s\_0):\t");

    scanf("%d", &s0);

    printf("  a:\t\t");

    scanf("%d", &a);

    printf("  b:\t\t");

    scanf("%d", &b);

    int k\_n[strlen(s)];

    printf("\nOne Time Key:\n  steps:\n");

    printf("    s\_0 = %d\n", s0);

    int random\_s = s0;

*for* (int i = 0; i < strlen(s); i++)

    {

        k\_n[i] = random\_s;

        char bin[5];

        int\_to\_binary(random\_s, bin);

*for* (int j = i \* 5; j < (i + 1) \* 5; j++)

        {

            k[j] = bin[j - (i \* 5)];

        }

*if* (i != strlen(s) - 1)

        {

            printf(

                "    s\_%d = (%d \* %2d + %d) mod %d = %2d mod %d = %2d\n",

                i + 1,

                a,

                random\_s,

                b,

                MOD,

                a \* random\_s + b,

                MOD,

                mod\_26(a \* random\_s + b));

        }

        random\_s = mod\_26(a \* random\_s + b);

    }

    printf("\n  ");

*for* (int i = 0; i < strlen(s); i++)

    {

        printf(" (%2d) ", k\_n[i]);

    }

    printf("\n  ");

*for* (int i = 0; i < strlen(s); i++)

    {

*for* (int j = i \* 5; j < (i + 1) \* 5; j++)

        {

            printf("%c", k[j]);

        }

        printf(" ");

    }

    printf("\n");

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

    encrypt(s, k);

    printf("------------------------------------------------\nEncrypted stream:\n  ");

*for* (int i = 0; i < strlen(s); i++)

    {

        char bin[5];

        int\_to\_binary(s[i] - 'A', bin);

*for* (int j = i \* 5; j < (i + 1) \* 5; j++)

        {

            printf("%c", bin[j - (i \* 5)]);

        }

        printf(" ");

    }

    printf("\n");

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

    printf("  Available plaintext stream:\n    ");

*for* (int i = 0; i < 3; i++)

    {

*for* (int j = i \* 5; j < (i + 1) \* 5; j++)

        {

            printf("%c", bin\_s[j]);

        }

        printf(" ");

    }

    printf("\n\n");

    char k\_g[256];

    cryptanalyze(s, bin\_s, k\_g);

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

    decrypt(s, k\_g);

    printf("------------------------------------------------\nDecrypted string: %s\n", s);

    printf("\n");

}

Sample run

