Assignment 2

IT851: Information and Systems Security Lab

**ANINDYA KUNDU**

IT, 8th Semester

ID 510817020

Repository:  
[github.com/meganindya/btech-assignments/information-and-systems-security/assg-2](https://github.com/meganindya/btech-assignments/tree/main/information-and-systems-security/assg-2)

Implement the following traditional symmetric ciphers:

1. Shift Cipher

Source: 1-a-cipher-shift.c

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

*#include* <stdlib.h> *// abs*

*#include* "utils.h"

*#define* MOD 26

*/\**

*\* Cipher shifts (in place) all characters of a string by a distance.*

*\**

*\* s: the string*

*\* z: shift distance*

*\*/*

void encrypt(char \*s, int z)

{

*for* (int i = 0; s[i] != '\0'; i++)

    {

        char c = s[i];

        int c\_n = c - 'A';

        int add\_n = c\_n + z;

        int n\_enc = mod\_26(add\_n);

        char c\_enc = n\_enc + 'A';

        printf(

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

            c,

            c\_n,

            c\_n,

            z,

            MOD,

            add\_n,

            MOD,

            c\_enc,

            n\_enc);

        s[i] = c\_enc;

    }

}

*/\**

*\* Cipher unshifts (in place) all characters of a string by a distance.*

*\**

*\* s: the string*

*\* z: shift distance*

*\*/*

void decrypt(char \*s, int z)

{

*for* (int i = 0; s[i] != '\0'; i++)

    {

        char c = s[i];

        int c\_n = c - 'A';

        int sub\_n = c\_n - z;

        int n\_dec = mod\_26(sub\_n);

        char c\_dec = n\_dec + 'A';

        printf(

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

            c,

            c\_n,

            c\_n,

            z,

            MOD,

            sub\_n,

            MOD,

            c\_dec,

            n\_dec);

        s[i] = c\_dec;

    }

}

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

int main(int argc, char \*argv[])

{

    char s[256];

    int z;

    printf("\nImplementation of Shift (Caesar) 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);

    printf("Enter cipher shift key: ");

    scanf("%d", &z);

    printf("\nEncryption:\n");

    encrypt(s, z);

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

    printf("\nDecryption:\n");

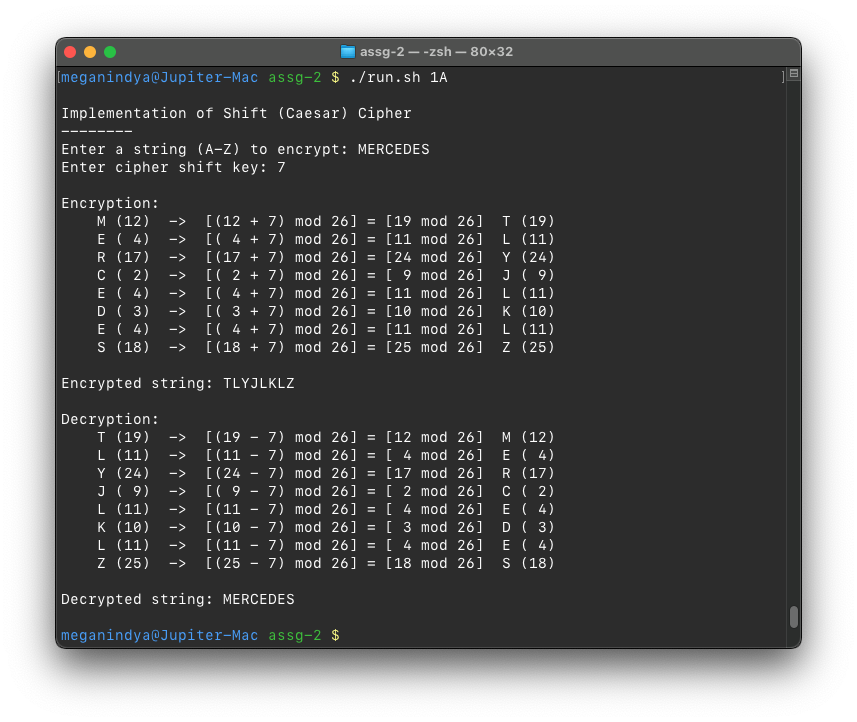
    decrypt(s, z);

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

    printf("\n");

}

Sample run



1. Multiplicative cipher

Source: 1-b-cipher-multiplicative.c

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

*#include* <stdlib.h> *// abs*

*#include* "utils.h"

*#define* MOD 26

*/\**

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

*\**

*\* s: the string*

*\* key: cipher key*

*\*/*

void encrypt(char \*s, int key)

{

    printf("    for key = %d\n\n", key);

*for* (int i = 0; s[i] != '\0'; i++)

    {

        char enc\_char = 'A' + mod\_26((s[i] - 'A') \* key);

        printf(

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

            s[i],

            s[i] - 'A',

            s[i] - 'A',

            key,

            MOD,

            (s[i] - 'A') \* key,

            MOD,

            enc\_char,

            enc\_char - 'A');

        s[i] = enc\_char;

    }

}

*/\**

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

*\**

*\* s: the string*

*\* key: cipher key*

*\*/*

void decrypt(char \*s, int key)

{

    int key\_inv = mod\_26\_mul\_inv(key);

    printf("    for key = %d\n    key multiplicative inverse (mod %d) = %d\n\n", key, MOD, key\_inv);

*for* (int i = 0; s[i] != '\0'; i++)

    {

        char dec\_char = 'A' + mod\_26((s[i] - 'A') \* key\_inv);

        printf(

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

            s[i],

            s[i] - 'A',

            s[i] - 'A',

            key\_inv,

            MOD,

            (s[i] - 'A') \* key\_inv,

            MOD,

            dec\_char,

            dec\_char - 'A');

        s[i] = dec\_char;

    }

}

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

int main(int argc, char \*argv[])

{

    char s[256];

    int key;

    printf("\nImplementation of Multiplicative 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 cipher multiplicative key: ");

        scanf("%d", &key);

        repeat = 0;

*if* (mod\_26\_mul\_inv(key) == -1)

        {

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

            repeat = 1;

        }

    } *while* (repeat);

    printf("\nEncryption:\n");

    encrypt(s, key);

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

    printf("\nDecryption:\n");

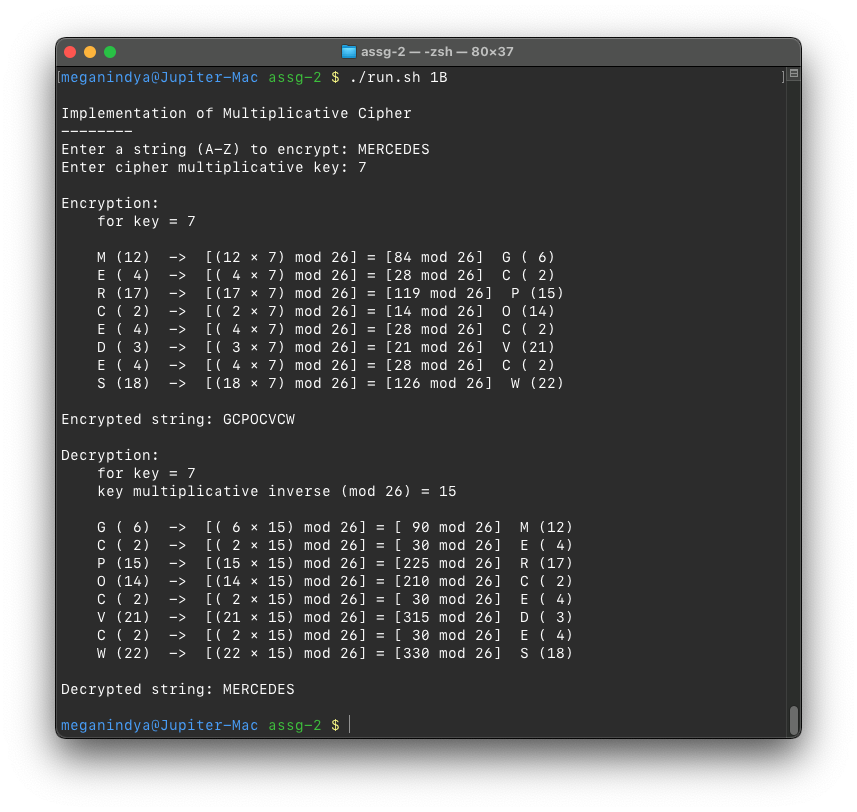
    decrypt(s, key);

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

    printf("\n");

}

Sample run



1. Affine Cipher

Source: 1—c-cipher-affine.c

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

*#include* <stdlib.h> *// abs*

*#include* "utils.h"

*#define* MOD 26

*/\**

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

*\**

*\* s: the string*

*\* key: cipher key*

*\*/*

void encrypt(char \*s, int key\_mul, int key\_add)

{

    printf("    for key (a, b) = (%d, %d)\n\n", key\_mul, key\_add);

*for* (int i = 0; s[i] != '\0'; i++)

    {

        char enc\_char = 'A' + mod\_26((s[i] - 'A') \* key\_mul + key\_add);

        printf(

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

            s[i],

            s[i] - 'A',

            s[i] - 'A',

            key\_mul,

            key\_add,

            MOD,

            (s[i] - 'A') \* key\_mul + key\_add,

            MOD,

            enc\_char,

            enc\_char - 'A');

        s[i] = enc\_char;

    }

}

*/\**

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

*\**

*\* s: the string*

*\* key: cipher key*

*\*/*

void decrypt(char \*s, int key\_mul, int key\_add)

{

    int key\_add\_inv = mod\_26\_add\_inv(key\_add);

    int key\_mul\_inv = mod\_26\_mul\_inv(key\_mul);

    printf("    for key (a, b) = (%d, %d)\n", key\_mul, key\_add);

    printf("    key (a = %d) multiplicative inverse (mod %d) = %d\n", key\_mul, MOD, key\_mul\_inv);

    printf("    key (b = %d) additive inverse (mod %d) = %d\n\n", key\_add, MOD, key\_add\_inv);

*for* (int i = 0; s[i] != '\0'; i++)

    {

        char dec\_char = 'A' + mod\_26(((s[i] - 'A') + key\_add\_inv) \* key\_mul\_inv);

        printf(

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

            s[i],

            s[i] - 'A',

            s[i] - 'A',

            key\_add\_inv,

            key\_mul\_inv,

            MOD,

            ((s[i] - 'A') + key\_add\_inv) \* key\_mul\_inv,

            MOD,

            dec\_char,

            dec\_char - 'A');

        s[i] = dec\_char;

    }

}

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

int main(int argc, char \*argv[])

{

    char s[256];

    int key\_add, key\_mul;

    printf("\nImplementation of Affine 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);

    printf("Enter cipher additive key: ");

    scanf("%d", &key\_add);

*do*

    {

        printf("Enter cipher multiplicative key: ");

        scanf("%d", &key\_mul);

        repeat = 0;

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

        {

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

            repeat = 1;

        }

    } *while* (repeat);

    printf("\nEncryption:\n");

    encrypt(s, key\_mul, key\_add);

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

    printf("\nDecryption:\n");

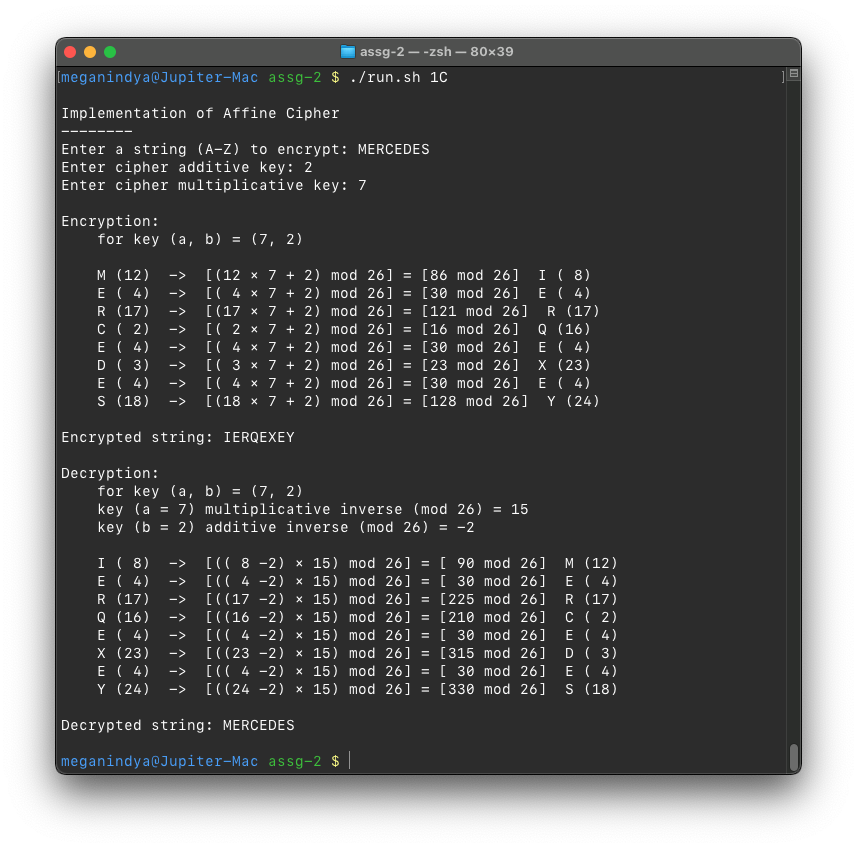
    decrypt(s, key\_mul, key\_add);

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

    printf("\n");

}

Sample run



1. Playfair Cipher

Source: 1-d-cipher-playfair.c

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

*#include* <stdlib.h> *// abs*

*#include* <string.h> *// strlen*

*#include* "utils.h"

*#define* MOD 26

*/\**

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

*\**

*\* s: the string*

*\* key: cipher key matrix*

*\*/*

void encrypt(char \*s, int \*\*key)

{

    int len = strlen(s);

*if* ((len & 1) == 1)

    {

        printf("  Salting odd length string with 'Z' at end\n\n");

        s[len] = 'Z';

        s[len + 1] = '\0';

    }

    int map[26][2];

    map[9][0] = -1; *// row of 'J'*

    map[9][1] = -1; *// col of 'J'*

*for* (int r = 0; r < 5; r++)

    {

*for* (int c = 0; c < 5; c++)

        {

            map[key[r][c]][0] = r;

            map[key[r][c]][1] = c;

        }

    }

    printf("  Pairwise transformation:\n");

*for* (int i = 0; s[i] != '\0'; i += 2)

    {

        int a\_r = map[s[i] - 'A'][0];

        int a\_c = map[s[i] - 'A'][1];

        int b\_r = map[s[i + 1] - 'A'][0];

        int b\_c = map[s[i + 1] - 'A'][1];

        int r\_a, r\_b;

*if* (a\_r == b\_r)

        {

            r\_a = key[a\_r][mod(a\_c + 1, 5)];

            printf("    %c (%d, %d) -> %c (%d, %d)\n", s[i], a\_r, a\_c, r\_a + 'A', a\_r, mod(a\_c + 1, 5));

            r\_b = key[a\_r][mod(b\_c + 1, 5)];

            printf("    %c (%d, %d) -> %c (%d, %d)\n", s[i + 1], b\_r, b\_c, r\_b + 'A', a\_r, mod(b\_c + 1, 5));

        }

*else* *if* (a\_c == b\_c)

        {

            r\_a = key[mod(a\_r + 1, 5)][a\_c];

            printf("    %c (%d, %d) -> %c (%d, %d)\n", s[i], a\_r, a\_c, r\_a + 'A', mod(a\_r + 1, 5), a\_c);

            r\_b = key[mod(b\_r + 1, 5)][a\_c];

            printf("    %c (%d, %d) -> %c (%d, %d)\n", s[i + 1], b\_r, b\_c, r\_b + 'A', mod(b\_r + 1, 5), a\_c);

        }

*else*

        {

            r\_a = key[a\_r][b\_c];

            printf("    %c (%d, %d) -> %c (%d, %d)\n", s[i], a\_r, a\_c, r\_a + 'A', a\_r, b\_c);

            r\_b = key[b\_r][a\_c];

            printf("    %c (%d, %d) -> %c (%d, %d)\n", s[i + 1], b\_r, b\_c, r\_b + 'A', b\_r, a\_c);

        }

        s[i] = r\_a + 'A';

        s[i + 1] = r\_b + 'A';

    }

}

*/\**

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

*\**

*\* s: the string*

*\* key: cipher key matrix*

*\*/*

void decrypt(char \*s, int \*\*key)

{

    int map[26][2];

    map[9][0] = -1; *// row of 'J'*

    map[9][1] = -1; *// col of 'J'*

*for* (int r = 0; r < 5; r++)

    {

*for* (int c = 0; c < 5; c++)

        {

            map[key[r][c]][0] = r;

            map[key[r][c]][1] = c;

        }

    }

    printf("  Pairwise transformation:\n");

*for* (int i = 0; s[i] != '\0'; i += 2)

    {

        int a\_r = map[s[i] - 'A'][0];

        int a\_c = map[s[i] - 'A'][1];

        int b\_r = map[s[i + 1] - 'A'][0];

        int b\_c = map[s[i + 1] - 'A'][1];

        int r\_a, r\_b;

*if* (a\_r == b\_r)

        {

            r\_a = key[a\_r][mod(a\_c - 1, 5)];

            printf("    %c (%d, %d) -> %c (%d, %d)\n", s[i], a\_r, a\_c, r\_a + 'A', a\_r, mod(a\_c - 1, 5));

            r\_b = key[a\_r][mod(b\_c - 1, 5)];

            printf("    %c (%d, %d) -> %c (%d, %d)\n", s[i + 1], b\_r, b\_c, r\_b + 'A', a\_r, mod(b\_c - 1, 5));

        }

*else* *if* (a\_c == b\_c)

        {

            r\_a = key[mod(a\_r - 1, 5)][a\_c];

            printf("    %c (%d, %d) -> %c (%d, %d)\n", s[i], a\_r, a\_c, r\_a + 'A', mod(a\_r - 1, 5), a\_c);

            r\_b = key[mod(b\_r - 1, 5)][a\_c];

            printf("    %c (%d, %d) -> %c (%d, %d)\n", s[i + 1], b\_r, b\_c, r\_b + 'A', mod(b\_r - 1, 5), a\_c);

        }

*else*

        {

            r\_a = key[a\_r][b\_c];

            printf("    %c (%d, %d) -> %c (%d, %d)\n", s[i], a\_r, a\_c, r\_a + 'A', a\_r, b\_c);

            r\_b = key[b\_r][a\_c];

            printf("    %c (%d, %d) -> %c (%d, %d)\n", s[i + 1], b\_r, b\_c, r\_b + 'A', b\_r, a\_c);

        }

        s[i] = r\_a + 'A';

        s[i + 1] = r\_b + 'A';

    }

}

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

int main(int argc, char \*argv[])

{

    char s[256], key\_s[16];

    printf("\nImplementation of Playfair 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 cipher key string (A-Z except J): ");

        scanf("%s", key\_s);

        repeat = 0;

*for* (int i = 0; s[i] != '\0'; i++)

        {

*if* (s[i] < 'A' || s[i] > 'Z' || s[i] == 'J')

            {

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

                repeat = 1;

*break*;

            }

        }

    } *while* (repeat);

    int \*\*key;

    key = malloc(5 \* sizeof \*key);

*for* (int i = 0; i < 5; i++)

    {

        key[i] = malloc(5 \* sizeof \*key[i]);

    }

    int flags[26];

*for* (int i = 0; i < 26; i++)

    {

        flags[i] = 0;

    }

    flags[9] = 1;

*for* (int i = 0; i < strlen(key\_s); i++)

    {

        key[i / 5][i % 5] = key\_s[i] - 'A';

        flags[key\_s[i] - 'A'] = 1;

    }

    int k = strlen(key\_s), ki = 0;

*while* (k < 25)

    {

*while* (flags[ki] == 1)

        {

            ki++;

        }

        key[k / 5][k % 5] = ki++;

        k++;

    }

    printf("\nKey:\n");

*for* (int i = 0; i < 5; i++)

    {

*for* (int j = 0; j < 5; j++)

        {

            printf("%3d (%c) ", key[i][j], key[i][j] + 'A');

        }

        printf("\n");

    }

    printf("\nEncryption:\n");

    encrypt(s, key);

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

    printf("\nDecryption:\n");

    decrypt(s, key);

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

    printf("\n");

*for* (int i = 0; i < 5; i++)

    {

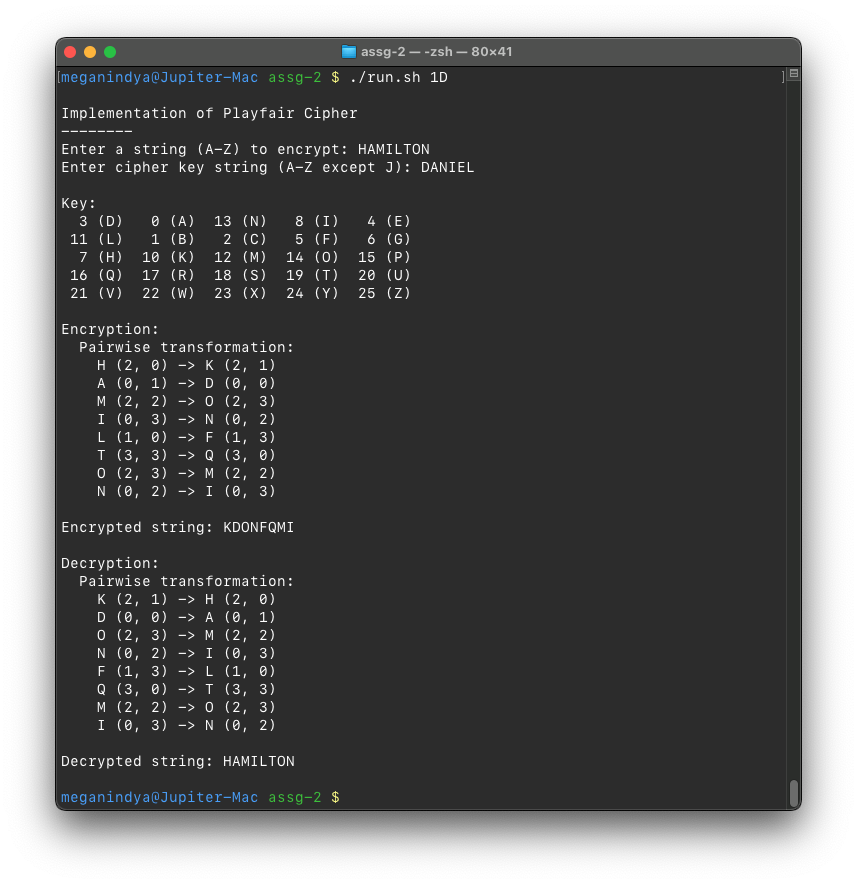
        free(key[i]);

    }

    free(key);

}

Sample run



5. Hill Cipher

Source: 1-e-cipher-hill.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 hill 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)

{

    printf("  Original string:\n");

*for* (int i = 0; i < strlen(s); i++)

    {

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

    }

    int temp[ord\_key];

    printf("\n  Multiplied:\n");

*for* (int r = 0; r < ord\_key; r++)

    {

        printf("    ");

*for* (int c = 0; c < ord\_key; c++)

        {

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

        }

        printf("%7d", s[r] - 'A');

        int sum = 0;

*for* (int c = 0; c < ord\_key; c++)

        {

            sum += key[r][c] \* (s[c] - 'A');

        }

        temp[r] = sum;

        printf("   |    %3d\n", sum);

    }

    printf("\n  Multiplied matrix (mod 26):\n");

*for* (int i = 0; i < ord\_key; i++)

    {

        s[i] = mod\_26(temp[i]) + 'A';

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

    }

}

*/\**

*\* Cipher hill 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)

{

    printf("  Encrypted string:\n");

*for* (int i = 0; i < strlen(s); i++)

    {

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

    }

    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);

    printf("\n  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");

    }

    int temp[ord\_key];

    printf("\n  Multiplied:\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("%7d", s[r] - 'A');

        int sum = 0;

*for* (int c = 0; c < ord\_key; c++)

        {

            sum += key\_inv[r][c] \* (s[c] - 'A');

        }

        temp[r] = sum;

        printf("   |    %3d\n", sum);

    }

    printf("\n  Multiplied matrix (mod 26):\n");

*for* (int i = 0; i < ord\_key; i++)

    {

        s[i] = mod\_26(temp[i]) + 'A';

        printf("    %c (%2d)\n", s[i], s[i] - '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 Hill 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);

    int ord\_key = strlen(s);

    int \*\*key;

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

*for* (int i = 0; i < ord\_key; i++)

    {

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

    }

*// Key manually provided for cases 2 and 3 for debugging and reviewing*

*if* (ord\_key == 2)

    {

        key[0][0] = 25;

        key[0][1] = 22;

        key[1][0] = 4;

        key[1][1] = 21;

    }

*else* *if* (ord\_key == 3)

    {

        key[0][0] = 6;

        key[0][1] = 24;

        key[0][2] = 1;

        key[1][0] = 13;

        key[1][1] = 16;

        key[1][2] = 10;

        key[2][0] = 20;

        key[2][1] = 17;

        key[2][2] = 15;

    }

*else*

    {

        srand(time(0));

*do*

        {

*for* (int i = 0; i < ord\_key; i++)

            {

*for* (int j = 0; j < ord\_key; j++)

                {

                    key[i][j] = rand() % 26;

                }

            }

            repeat = mat\_invert\_check(ord\_key, key);

        } *while* (repeat);

    }

    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");

*for* (int i = 0; i < ord\_key; i++)

    {

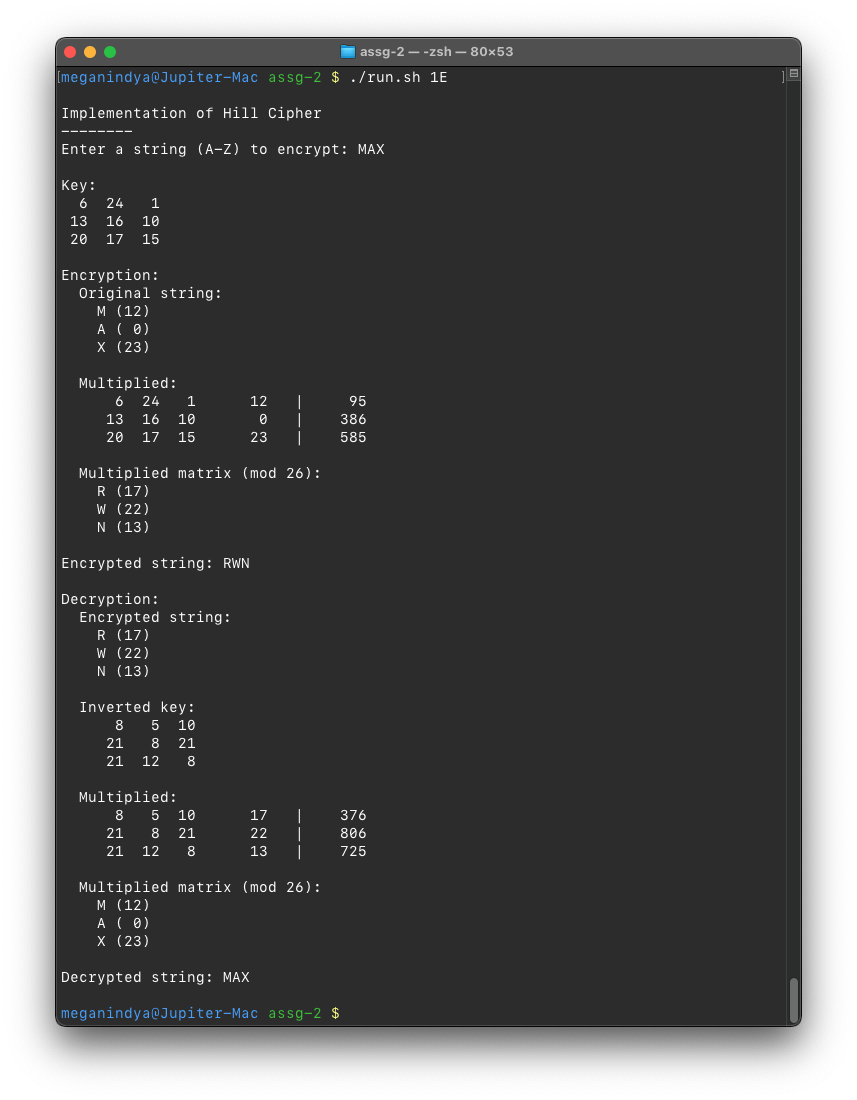
        free(key[i]);

    }

    free(key);

}

Sample run



2. Write programs to carry out exhaustive key search attacks on the Shift Cipher, Multiplicative Cipher and Affine Cipher that you have implemented. (Aim to attack a cipher is to break its key.)

a. Hence use an exhaustive key search to decrypt the following ciphertext, which was encrypted using a Shift Cipher:

BMMTDXLTANZXXYYHKMMHYKXXRHNKLXEYYKHFFXFHKR

Source: 2-a-attack-cipher-shift.c

*#include* <stdio.h>

*#include* <stdlib.h>

*#include* <string.h>

*#include* "utils.h"

*#define* MOD 26

*/\**

*\* Performs shift cipher decripytion against all additive inverse values in [0, m - 1].*

*\**

*\* s: the encoded string*

*\*/*

void attack(char \*s)

{

*for* (int i = 0; i < MOD; i++)

    {

        int key\_inv = mod\_26\_add\_inv(i);

        printf("    for key = %d\n    key additive inverse (mod %d) = %d\n", i, MOD, key\_inv);

        char enc[64];

        strcpy(enc, s);

*for* (int j = 0; enc[j] != '\0'; j++)

        {

            enc[j] = 'A' + mod\_26((enc[j] - 'A') + key\_inv);

        }

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

    }

}

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

int main(int argc, char \*argv[])

{

    char s[64] = "BMMTDXLTANZXXYYHKMMHYKXXRHNKLXEYYKHFFXFHKR";

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

    attack(s);

}

Sample run





b. Use an exhaustive key search to decrypt the following ciphertext, which was encrypted using a Multiplicative Cipher:

WFEJBYOFAJZEYDCMRBKJRKWABKXSWKJZSFQ

Source: 2-b-attack-cipher-multiplicative.c

*#include* <stdio.h>

*#include* <stdlib.h>

*#include* <string.h>

*#include* "utils.h"

*#define* MOD 26

*/\**

*\* Performs shift cipher decripytion against all valid multiplicative inverse values in [0, m - 1].*

*\**

*\* s: the encoded string*

*\*/*

void attack(char \*s)

{

*for* (int i = 0; i < MOD; i++)

    {

        int key\_inv = mod\_26\_mul\_inv(i);

*if* (key\_inv == -1)

        {

            printf("    for key = %d\n", i);

            printf("    key multiplicative inverse doesn't exist, this key isn't valid\n\n");

*continue*;

        }

        printf("    for key = %d\n    key multiplicative inverse (mod %d) = %d\n", i, MOD, key\_inv);

        char enc[64];

        strcpy(enc, s);

*for* (int j = 0; enc[j] != '\0'; j++)

        {

            enc[j] = 'A' + mod\_26((enc[j] - 'A') \* key\_inv);

        }

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

    }

}

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

int main(int argc, char \*argv[])

{

    char s[64] = "WFEJBYOFAJZEYDCMRBKJRKWABKXSWKJZSFQ";

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

    attack(s);

}

Sample run





c. Use an exhaustive key search to decrypt the following ciphertext, which was encrypted using a Affine Cipher:

EFXECFBDQGGXRADQTFFUFSPGAHQTDGGAFZDJFGHJFBDQGHGDCCGXSFJDHQGAFZDJF

Source: 2-c-attack-cipher-affine.c

*#include* <stdio.h>

*#include* <stdlib.h>

*#include* <string.h>

*#include* "utils.h"

*#define* MOD 26

*/\**

*\* Performs shift cipher decripytion against all valid multiplicative inverse values in [0, m - 1].*

*\**

*\* s: the encoded string*

*\*/*

void attack(char \*s)

{

*for* (int i = 0; i < MOD; i++)

    {

        int key\_mul\_inv = mod\_26\_mul\_inv(i);

*if* (key\_mul\_inv == -1)

        {

            printf("for multiplicative key = %d\n", i);

            printf("key multiplicative inverse doesn't exist, this key isn't valid\n\n");

*continue*;

        }

*for* (int j = 0; j < MOD; j++)

        {

            int key\_add\_inv = mod\_26\_add\_inv(j);

            printf("    for key (a, b) = (%d, %d)\n", i, j);

            printf("    key (a = %d) multiplicative inverse (mod %d) = %d\n", i, MOD, key\_mul\_inv);

            printf("    key (b = %d) additive inverse (mod %d) = %d\n", j, MOD, key\_add\_inv);

            char enc[80];

            strcpy(enc, s);

*for* (int x = 0; enc[x] != '\0'; x++)

            {

                enc[x] = 'A' + mod\_26(((enc[x] - 'A') + key\_add\_inv) \* key\_mul\_inv);

            }

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

        }

    }

}

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

int main(int argc, char \*argv[])

{

    char s[80] = "EFXECFBDQGGXRADQTFFUFSPGAHQTDGGAFZDJFGHJFBDQGHGDCCGXSFJDHQGAFZDJF";

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

    attack(s);

}

Sample run

