**TÜRKİYE CUMHURİYETİ**

**YILDIZ TEKNİK ÜNİVERSİTESİ**

**BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ**



**YAPISAL PROGRAMLAMA FİNAL PROJESİ**

Öğrenci No: 20011910 Öğrenci Adı Soyadı: Ali Albayrak

Öğrenci e-posta: l1120910@std.yildiz.edu.tr

Ders Yürütücüsü

Öğr.Gör. Ahmet ELBİR

**Haziran, 2021**

İçindekiler

Algoritma tanıtımı ..……………………………………………………………………………………3

Amaç .……………………………………………………………………………………………..3

Encryption .……………………………………………………………………………………..3

Decryption ..…………………………………………………………………………………….3

Karşılaştırma ..……………………………………………………………………………………………4

Avantajlar ..……………………………………………………………………………………..4

Dezavantajlar ..………………………………………………………………………………..4

Complexity ..………………………………………………………………………………………………4

Zaman karmaşıklığı ..………………………………………………………………………..4

Memory karmaşıklığı .……………………………………………………………………..4

Uygulama ..………………………………………………………………………………………………..5

Normal kullanıcı için çıktı ..……………………………………………………………….5

sayaçlarla olan çıktı ..……………………..………………………………………………..5

C program kodu ..…………….………………………………………………………………………..6

* **Algoritma tanitimi**
* **Amac:** Hill cipher bir sifreleme algoritmasi. Sifreleme algoritmalari iki turu var biri Semitrik diğeri Asimetrik bizimki simetrik turunden bir algoritmadır.

Simetrik algoritmalar ayni anahtara bagli bir decryption islemi olan algoritmalar dir

Hill cipher algoritması girilen mesaji matrisi çevirerek işlem yapar. Yani anahtarimiz da bir matris olacak.

* **Çalismasi:**

**1.adim:** alphabe tablomuzu oluşturup character sayisi bizim mod sayimiz olacak (k diyelim ona)

**2.adim:** anahtar olarak bir kare matris belirlenir.

**3.adim:** girilen mesaj  bloklara bolunur ve her blok anahtar matrisin mertebesi kadar harf içerir.

* **Şifreleme (Encryption)** **:** ayirdigimiz bloklar her birini alip anahtar matrisi ile carpariz, cikan sonucun da mod sayımıza (k) göre modunu alarak yeni sifreli blogumuz oluşmuş olur.

* **Şifreyi çözme (Decrypyion):** ilk önce anahtar matrisinin inversini kofaktörler matrisi ile buluruz.(NOT:inversi bulurken tam sayi çıkmayabilir bu sebeple (determinant \* x mod k = 1)olacak şekilde bir x degeri bulup kofactorler matrisine carpariz).

Sonra girilen bloklara ayrılmış şifreli metni invers matris ile çarpıp asil metni elde etmiş oluruz.

* **Karşılaştırma**
* **Avantajlar**

- Ayni harfler şifreledikten sonra genellikle farkli cikar

- Blokta ki herhangi bir harf değişirse tüm şifreli blok degisir

- Anahtari mertebesi ne kadar büyük olursa o kadar tahmin edilmesi zor olur

- Asimetrik algoritmalara göre daha hizli

* **Dezavantajlar**

- Simetrik şifreleme algoritmaların en büyük sıkıntısı anahtar her iki tarafta olması gerektigi icin taşınırken yanlış yere gidebilir veya saldırılabilir.

- kimlik doğrulama testi içermez. yani anahtara sahip her kimse şifreleyip gönderebilir

* **Karmaşıklığı (complexity)**
* **Zaman :** algoritma matris çarpımı ile gerçekleşir ancak n3 değil çünkü şu şekilde oluyor

Metin uzunluğu = n

Anahtar mertebesi = k

Üç tane for loop olur yine de ama en dıştaki for n/k olur ve içerdeki kalan iki for da k defa döner

Yani n/k \* k \*k = k\*n = O(n2) olur

Not : algoritmada tek seferlik O(k3) karmaşıklığa sahip anahtarın inversi bulma var.

* **Memory :**

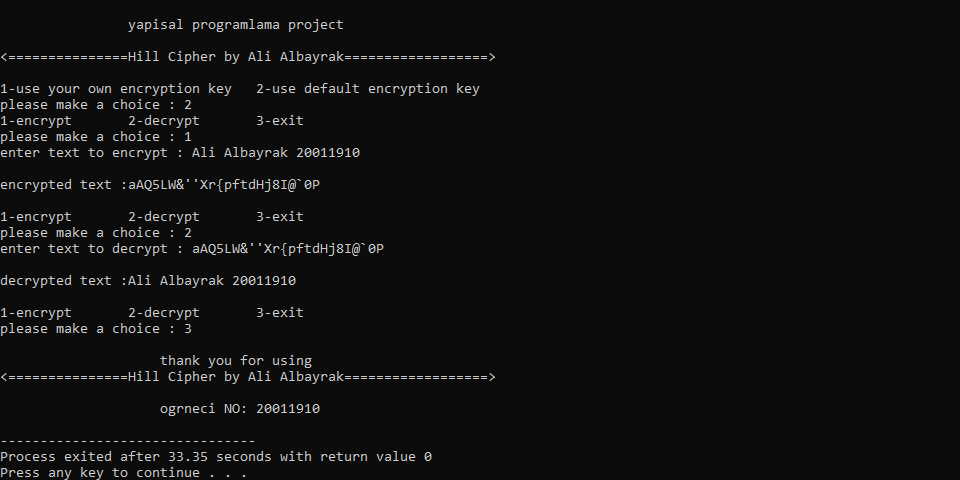
anahtar ve inverse anahtar = 2K2

inverse bulmak için kullanılan matrisler = (k-1)2 + k2

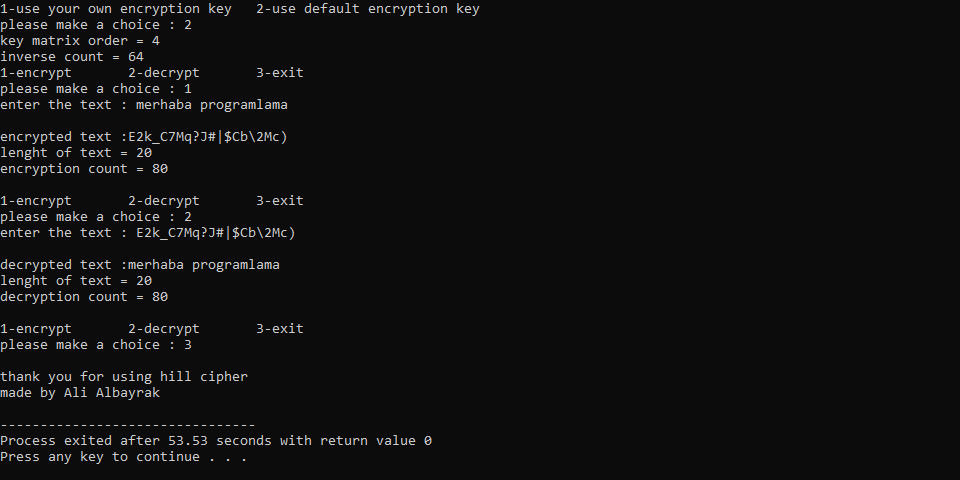
metin ve şifreli metin = 2n

toplamda 2n+4k2 = (n2)’lik bir alan

* **Uygulama**
* normal kullanici icin cikti



* sayaçlarla olan cikti



* **C program kodu:**

**#include <stdio.h>**

**#include <math.h>**

**#include <stdlib.h>**

**#include <string.h>**

**#define SIZE 100**

**#define N 4**

**//#define analiz 1**

**int modk = 94; //define'yi kullanmadim cunku anahtar uygun degilse toplam harf sayisi bir arttirip uygun olma sansini yukseltiriz**

**#ifdef analiz**

**int count1 = 0;**

**int count2 = 0;**

**#endif**

**void inverse(int\*\*, int\*\*);**

**float det(int , int\*\*);**

**float cofactor(int, int\*\*, int\*\*, int, int);**

**void transpose(int , int\*\*);**

**int mod\_inv(int, int);**

**void encrypt(char[], char[], int\*\*);**

**void decrypt(char[], char[], int\*\*);**

**int main(){**

**///////////////default keys////////////////////**

**const int anahtar2[2][2] = {{9, 4},**

**{4, 3}};**

**const int anahtar3[3][3] = {{5,-2, 7},**

**{6, 1, 5},**

**{4,-3, 8}};**

**const int anahtar4[4][4] = {{2, 4, 1, 3},**

**{7, 2, 2, 2},**

**{3, 3, 2, 2},**

**{0, 5, 1, 0}};**

**/////////////////////////////////////////////////////**

**int \*\*anahtar, \*\*in\_anahtar;**

**int i, j, k;**

**int len, satir, choice;**

**char metin[SIZE], sifreli\_metin[SIZE];**

**//################**

**// Bellek tahsisi#**

**//################**

**anahtar = (int \*\*) malloc(N \* sizeof(int));**

**for(i=0;i<N;i++) {**

**anahtar[i] = malloc( N \* sizeof(int));**

**}**

**//yeterli bellek kontororlu**

**if(!anahtar){**

**printf("Memory Eror");**

**exit(1);**

**}**

**in\_anahtar = (int \*\*) malloc( (N) \* sizeof(int));**

**for(i=0;i<N;i++) {**

**in\_anahtar[i] = malloc( (N) \* sizeof(int));**

**}**

**//yeterli bellek kontororlu**

**if(!in\_anahtar){**

**printf("Memory Eror");**

**exit(1);**

**}**

**///////////////////////**

**printf("\n\t\tyapisal programlama project\n");**

**printf("\n<===============Hill Cipher by Ali Albayrak==================>\n\n");**

**//#########################**

**// choosing encryption key#**

**//#########################**

**printf("1-use your own encryption key\t2-use default encryption key\nplease make a choice : ");**

**scanf("%d",&choice);**

**if(choice == 1){**

**printf("enter elements of key matrix(%dx%d)\n",N,N);**

**for(i=0;i<N;i++){**

**for(j=0;j<N;j++){**

**scanf("%d", &anahtar[i][j]);**

**}**

**}**

**}else{**

**switch(N){**

**case 2:**

**for(i=0;i<N;i++){**

**for(j=0;j<N;j++){**

**anahtar[i][j] = anahtar2[i][j];**

**}**

**}**

**break;**

**case 3:**

**for(i=0;i<N;i++){**

**for(j=0;j<N;j++){**

**anahtar[i][j] = anahtar3[i][j];**

**}**

**}**

**break;**

**case 4:**

**for(i=0;i<N;i++){**

**for(j=0;j<N;j++){**

**anahtar[i][j] = anahtar4[i][j];**

**}**

**}**

**break;**

**default:**

**printf("Error: N value (%d) has no defult matrix", N);**

**exit(1);**

**}**

**}**

**///////////////////////////////////////////**

**///////////////////////////////////////////**

**inverse(anahtar,in\_anahtar);**

**#ifdef analiz**

**printf("inverse count = %d\n",count1);**

**#endif**

**///////////////////////////////////////////**

**///////////////////////////////////////////**

**do{**

**printf("1-encrypt\t2-decrypt\t3-exit\nplease make a choice : ");**

**scanf("%d",&choice);**

**switch(choice){**

**case 1:**

**printf("enter text to encrypt : ");**

**fflush(stdin);**

**scanf("%[^\n]s", metin);**

**len = strlen(metin);**

**if(len%N!=0)**

**len += N-len%N;**

**encrypt(metin,sifreli\_metin,anahtar);//encrypt yerine decrypt de kullanilabilir cunku ayni islemler ama farkli dizilerde**

**printf("\nencrypted text :");**

**for(i=0;i<len;i++){**

**printf("%c",sifreli\_metin[i]);**

**}**

**printf("\n\n");**

**#ifdef analiz**

**printf("\nencryption count = %d\n\n",count2);**

**#endif**

**strcpy(sifreli\_metin,"");**

**strcpy(metin,"");**

**break;**

**case 2:**

**printf("enter text to decrypt : ");**

**fflush(stdin);**

**scanf("%[^\n]s", sifreli\_metin);**

**len = strlen(sifreli\_metin);**

**if(len%N!=0)**

**len += N-len%N;**

**decrypt(sifreli\_metin,metin,in\_anahtar);//decrypt yerine encrypt de kullanilabilir cunku ayni islemler ama farkli dizilerde**

**printf("\ndecrypted text :");**

**for(i=0;i<len;i++){**

**printf("%c",metin[i]);**

**}**

**printf("\n\n");**

**#ifdef analiz**

**printf("\ndecryption count = %d",count2);**

**#endif**

**strcpy(metin,"");**

**strcpy(sifreli\_metin,"");**

**break;**

**case 3:**

**printf("\n\t\t thank you for using\n");**

**printf("<===============Hill Cipher by Ali Albayrak==================>\n\n");**

**printf("\t\t ogrneci NO: 20011910\n");**

**break;**

**default:**

**printf("\nError:invalid choice\n");**

**break;**

**}**

**}while(choice != 3);**

**free(anahtar);**

**free(in\_anahtar);**

**return 0;**

**}**

**void inverse(int \*\*anahtar, int \*\*in\_anahtar){**

**float yeni[SIZE][SIZE];**

**float d;**

**int mod;**

**int i, j;**

**for(i=0;i<N;i++){**

**for(j=0;j<N;j++){**

**yeni[i][j] = cofactor(N,anahtar,in\_anahtar,i,j) \* pow(-1,i+j);**

**}**

**}**

**d = det(N,anahtar);**

**mod = mod\_inv(fabs(d),modk);**

**if(mod != 0){**

**if(d<0){**

**mod \*= -1;**

**mod += modk;**

**}**

**for(i=0;i<N;i++){**

**for(j=0;j<N;j++){**

**in\_anahtar[i][j] = yeni[i][j] \* mod;**

**}**

**}**

**transpose(N,in\_anahtar);**

**}else{**

**modk++;**

**mod = mod\_inv(fabs(d),modk);**

**if(mod != 0){**

**if(d<0){**

**mod \*= -1;**

**mod += modk;**

**}**

**for(i=0;i<N;i++){**

**for(j=0;j<N;j++){**

**in\_anahtar[i][j] = yeni[i][j] \* mod;**

**in\_anahtar[i][j] = in\_anahtar[i][j]%modk;**

**if(in\_anahtar[i][j]<0){**

**in\_anahtar[i][j] += modk;**

**}**

**}**

**}**

**transpose(N,in\_anahtar);**

**}else{**

**printf("this encryption key is not valid\n");**

**exit(1);**

**}**

**}**

**}**

**float det(int n, int \*\*matris){**

**int i, j, k, m;**

**int \*\*minor;**

**float deter = 0.0;**

**if(n==1){**

**return matris[0][0];**

**}**

**if(n==2){**

**deter = (matris[0][0] \* matris[1][1]) - (matris[0][1] \* matris[1][0]);**

**}**

**else{**

**i=0;**

**for(j=0;j<n;j++){**

**minor = (int \*\*) malloc( (n-1) \* sizeof(int));**

**for(k=0;k<n-1;k++) {**

**minor[k] = malloc( (n-1) \* sizeof(int));**

**}**

**if(!minor){**

**printf("Eror");**

**exit(1);**

**}**

**#ifdef analiz**

**count1++;**

**#endif**

**deter += matris[i][j] \* pow(-1,i+j) \* cofactor(n,matris,minor,i,j);**

**// free(minor);**

**}**

**}**

**free(minor);**

**return deter;**

**}**

**float cofactor(int n, int \*\*matris, int \*\*tmp, int rindex, int cindex){**

**int i, j;**

**int r=0, c=0;**

**for(i=0;i<n;i++){**

**for(j=0;j<n;j++){**

**if(i != rindex && j != cindex){**

**tmp[r][c] = matris[i][j];**

**c++;**

**if(c == n-1){**

**c=0;**

**r++;**

**}**

**}**

**}**

**}**

**return det(n-1,tmp);**

**}**

**void transpose(int n, int \*\*matris){**

**int i, j;**

**int tmp[SIZE][SIZE];**

**for(i=0;i<n;i++){**

**for(j=0;j<n;j++){**

**tmp[j][i] = matris[i][j];**

**}**

**}**

**for(i=0;i<n;i++){**

**for(j=0;j<n;j++){**

**matris[i][j] = tmp[i][j];**

**}**

**}**

**}**

**int mod\_inv(int d, int key){**

**int i = 1;**

**while((i\*d)%key != 1 && i<key){**

**i++;**

**}**

**if(i<key){**

**return i;**

**}**

**return 0;**

**}**

**void encrypt(char metin[], char sifreli\_metin[], int \*\*anahtar){**

**int i, j, k;**

**int len, satir, choice;**

**int metin\_s[SIZE][N], sifreli\_metin\_s[SIZE][N];**

**#ifdef analiz**

**count2 = 0;**

**#endif**

**len = strlen(metin);**

**j=0;**

**satir=0;**

**for(i=0;i<len;i++){**

**metin\_s[satir][j++] = metin[i]-32;**

**if(j==N){**

**j=0;**

**satir++;**

**}**

**}**

**if(len%N != 0){**

**for(i=0;i<N-(len%N);i++){**

**metin\_s[satir][j++] = 0;**

**}**

**len += N-(len%N);**

**}**

**satir = ceil(len/N);**

**for(i=0;i<satir;i++){**

**for(j=0;j<N;j++){**

**int sum=0;**

**for(k=0;k<N;k++){**

**sum += metin\_s[i][k]\*anahtar[k][j];**

**#ifdef analiz**

**count2++;**

**#endif**

**}**

**sifreli\_metin\_s[i][j] = (sum%modk);**

**if(sifreli\_metin\_s[i][j]<0){**

**sifreli\_metin\_s[i][j] += modk;**

**}**

**}**

**}**

**for(i=0;i<satir;i++){**

**for(j=0;j<N;j++){**

**sifreli\_metin[j+i\*N] = sifreli\_metin\_s[i][j]+32;**

**}**

**}**

**}**

**void decrypt(char sifreli\_metin[], char metin[], int \*\*in\_anahtar){**

**int i, j, k;**

**int len, satir;**

**int metin\_s[SIZE][N], sifreli\_metin\_s[SIZE][N];**

**#ifdef analiz**

**count2 = 0;**

**#endif**

**len = strlen(sifreli\_metin);**

**j=0;**

**satir=0;**

**for(i=0;i<len;i++){**

**sifreli\_metin\_s[satir][j++] = sifreli\_metin[i]-32;**

**if(j==N){**

**j=0;**

**satir++;**

**}**

**}**

**if(len%N != 0){**

**for(i=0;i<N-(len%N);i++){**

**sifreli\_metin\_s[satir][j++] = 32;**

**}**

**len += N-(len%N);**

**}**

**satir = len/N;**

**for(i=0;i<satir;i++){**

**for(j=0;j<N;j++){**

**int sum=0;**

**for(k=0;k<N;k++){**

**sum += sifreli\_metin\_s[i][k]\*in\_anahtar[k][j];**

**#ifdef analiz**

**count2++;**

**#endif**

**}**

**metin\_s[i][j] = (sum%modk);**

**if(metin\_s[i][j]<0){**

**metin\_s[i][j] += modk;**

**}**

**}**

**}**

**for(i=0;i<satir;i++){**

**for(j=0;j<N;j++){**

**metin[j + i\*N] = metin\_s[i][j]+32;**

**}**

**}**

**}**