
# BİTİRME PROJESİ

Haftalık Rapor – 22.10.2021

22 EKİM 2021

KIRIKKALE ÜNİVERSİTESİ – BİLGİSAYAR MÜHENDİSLİĞİ İÖ

AHMET MUNGAN – 160255081

# İÇİNDEKİLER

# ÖZET

Geçen hafta itibariyle hill algoritmasının matematiğinin anlaşılması sağlandı. Bu hafta ile birlikte kodlaması yapılmıştır. Bu kodlama daha tutarlı bir yaklaşımla yapılıp, bu kriptosistemin tersine mühendislik ile incelemesi yapılmıştır. Tersine mühendislik kısmında hill algoritmasının kırılabilir şifre sorunlarının aşımı için birkaç model bulunmuştur.

## HILL ALGORİTMASININ KODLANMASI

Geçen hafta[1] itibariyle hill kriptosistemi sesin matematikselleştirildiği takdirde şifrelemek için iyi bir teknik olduğu düşünülmüştür. Bu kanı, doğru ya da yanlış değildir. Sadece bir denemedir ve bu deneme sonucunda şifreleme kısmı temel anlamda hakkında bilgi edinilen bir alan olacaktır. Ayrıca geçen hafta matematiksel çözümü gösterilen hill algoritmasının veri bilimine yönelik olarak python dili ile kodlaması yapılacaktır.

Algoritma kodlanmadan evvel dikkat edilmesi gereken en önemli hususlardan birisi alfabedir. Hill algoritmasında gizli anahtar $mod(29)$'da terslenebilir bir matris seçildiği için, tersi olan matris bulmak ingiliz alfabesine nazaran daha kolaydır. Çünkü ingiliz alfabesindeki 26 harf sayısı, asal değildir ve çarpanları matrisin tersinin yanlış bulunmasına etki edebilir. Dolayısıyla türk alfabesi aslında bu şifreleme algoritması için uygundur denebilir.

Algoritmanın kodlanması için modül bazlı geliştirme yapılarak gidilirse, alfabe ve gereksinimleri ilk başta kodlanabilir. Spesifik işler yapılmayacağı için Jupyter Notebook gibi veri bilimine yönelik bir ortam yerine; daha esnek ve kodun bir arada incelenmesini kolaylaştıran Visual Studio Code üzerinde geliştirme yapılacaktır.

*VS Code 1*

```
turkceAlfabe = [["a",0],["b",1],["c",2],["ç",3],["d",4],["e",5],
["f",6],["g",7],["ğ",8],["h",9],["i",10],["ı",11],["j",12],["k",13],
["l",14],["m",15],["n",16],["o",17],["ö",18],["p",19],["r",20],
["s",21],["ş",22],["t",23],["u",24],["ü",25],["v",26],["y",27],
["z", 28]]
def bul(list1, list2):
    return turkceAlfabe[list1][list2]
def yazdir():
    print(turkceAlfabe)
def alfabeSayisi():
    return turkceAlfabe.__len__()
def numerikCevir(metin):
    liste = []
```

[1] Geçen haftalara ait tüm raporlar ekler kısmında paylaşılan kişisel git uzantılı link ile erişilebilirdir. (Bkz. EKLER)

```
    for i in metin:
        for j in range(0, alfabeSayisi()):
            if bul(j, 0) == i.lower():
                liste.append(j)
    return liste
def karakterCevir(liste):
    metin = ""
    for i in liste:
        for j in range(0, alfabeSayisi()):
            if bul(j, 1) == i:
                metin += bul(j, 0)
    return metin
```

VS Code 1'de görüldüğü gibi türkçe alfabenin modülü ve gerekli fonksiyonlar oluşturulmuştur. Türkçe alfabenin hill algoritmasına etkisinin pozitif olamasına rağmen, diğer şifreleme sistemlerinin de denenmesi için ingilizce alfabe[2] de oluşturulmuştur.

VS Code 1'de verinin nasıl geldiğinden bağımsız nümerik ve karakter dönüşümleri için fonksiyonlar tanımlanmıştır. Bu sözlük fonksiyonlar algoritma uygulanırken işi kolaylaştırıp asıl algoritmaya odaklanmayı sağlayacaktır. Buna en iyi örnek fakat ses için kullanılması pek doğru olmayan sezar şifreleme algoritmalarıdır.

*VS Code 2*

```
#SEZAR ALGORİTMASI ŞİRELEME
from alfabeler import turkceAlfabe as alfabe
import random
M = "ahmetmungan"
C = ""
MListOtelenmis = []
otelemeSayisi = random.randint(1,alfabe.alfabeSayisi())
for i in alfabe.numerikCevir(M):
    i = (i + otelemeSayisi) % alfabe.alfabeSayisi()
    MListOtelenmis.append(i)
print("öteleme sayısı: ", otelemeSayisi)
print("MList ötelenmiş: ", MListOtelenmis)
for i in MListOtelenmis:
    C += alfabe.bul(i, 0)
```

[2] Alfabe türlerinin alfabe class'larından inheritance edildiği gibi kodlamanın nesneye ve geliştirmeye yönelik detayları paylaşılmamıştır. Paylaşılmayan programlama bilgilerinin varsayılan olarak bilinmesi gerektiği için gereksinim duyulmamıştır.

```
print("Şifreli metin: ", C)
print("Alfabedeki karakter sayısı: ", alfabe.alfabeSayisi())

#SEZAR ALGORİTMASI DEŞİFRELEME
desifre = []
for i in alfabe.numerikCevir(C):
    i = (i - otelemeSayisi) % (alfabe.alfabeSayisi())
    desifre.append(i)
print("Deşifre: ", alfabe.karakterCevir(desifre))
```

VS Code 2'de sezar algoritması kodlanmıştır. Alfabe kısmının iyi tasarlandığı çerçevede VS Code 2'de görüldüğü gibi algoritmaya odaklanılmış ve algoritmaya yönelik karmaşık kısımların yönetimi kolaylaşmıştır.

VS Code 2'de sezar algoritmasının şifreleme ve deşifreleme işlemleri için tekli bloklar kullanılmıştır. Ayrıca sezar algoritmasının en büyük özelliklerinden öteleme işlemi yapılmıştır. Burada öteleme işlemi rastgeleliğe[3] bırakılmıştır. Ayrıca türk alfabesi için $mod(29)$ gibi sayı içeren bir formülizasyona gidilmemiştir. Burada *alfabeSayisi()* fonksiyonu ile dönen, alfabe içerisindeki harf sayısı mod olarak kullanılmıştır. Deşifrelemede ise öteleme değil, eksiltili modülasyon yapılıp metnin kendisine ulaşılır. Bu algoritma modeli ile geliştirme yapılabildiği anlaşılmıştır. Dolayısıyla hill algoritması bu temel üzerine inşa edilecektir.

*VS Code 3*

```
#HILL ALGORİTMASI ŞİFRELEME
from alfabeler import turkceAlfabe as alfabe
import random
M = "ahmetmunganj"
A = [[3, 7], [1, 2]]
A_1 = [[27, 7],[1, 26]]
M = alfabe.numerikCevir(M)
sonuc = []
print(M)
for i in range(0, len(M), 2):
    sonuc.append((A[0][0]*M[i] + A[0][1]*M[i+1]) %
alfabe.alfabeSayisi())
    sonuc.append((A[1][0]*M[i] + A[1][1]*M[i+1]) %
alfabe.alfabeSayisi())
```

[3] Herhangi bir büyüklük belirlemeden, sadece sınır değerlerin algoritmada kullanılıp rasgeleliğe esnek bir geliştirme olduğunun göstergesidir.

```
C = alfabe.karakterCevir(sonuc)
print("Şifreli metin: ", C)

#HILL ALGORİTMASI DEŞİFRELEME
C = alfabe.numerikCevir(C)
sonuc = []
for i in range(0, len(C), 2):
    sonuc.append((A_1[0][0]*C[i] + A_1[0][1]*C[i+1]) %
alfabe.alfabeSayisi())
    sonuc.append((A_1[1][0]*C[i] + A_1[1][1]*C[i+1]) %
alfabe.alfabeSayisi())
yeni = alfabe.karakterCevir(sonuc)
print("Deşifreli metin: ", yeni)
```

VS Code 1'deki alfabe kullanılarak, VS Code 3'teki hill algoritması oluşturulmuştur. Hill algoritmasının ilk hali VS Code 3'te bilinen ve $mod(29)$'da tersi bilinen bir matris anahtar seçilmiştir. Bu çerçevede şifrelenecek metinler sezar algoritmasına nazaran ikili gruplar halinde seçilmiştir. Fakat anahtarın; VS Code 2'de sezar algoritmasında olduğu gibi rastgelelik taşıması gerekir. Bu gereklilik farklı anahtarların üretilmesi için gereklidir. Farklı anahtarlar farklı çözümler anlamına geleceğinden, rastgelelik hill algoritması için gerekli bir öz niteliktir.

VS Code 3'teki rastgelelik sorununun çözümü için:

1. Rastgele sayıları matris içerisine doldurup tek tek her bir matrisin $mod(29)$'da tersi var mı diye kontrol edilebilir.
2. Rastgele sayıları matrisin tersi olarak kabul edilip tersi işlemler yapılarak asıl matris elde edilebilir. Bu sayede tersini bulmak için yapılabilecek işlem dizilerinin tamamı bir araya getirilebilir.
3. Genetik algoritmalar kullanılarak; popülasyon ve kromozom dağılımları göz önünde bulundurularak ondalıklı ve çözüm uzayında kabul görmüş lineer sistemleri üretmek mümkündür. Büyük tıkanmalarda mutasyon sayısı ile başka bir çözüm uzayına atlama yöntemi kullanılabilir.

Yukarıda bahsi geçen 3 adet çözüm yöntemlerinden herhangi biri örnek olarak, işe yaradığı düşünülürse başka temel bir problem ile karşılaşılır. Daha evvelki raporlarda yöntemlerin tersine mühendislik ile yorumlama kısımlarında yaşanan bir

problem vardı. Bu yöntemin de güvenlik açığı tersine mühendislik kısmında paylaşılacaktır.

## HILL ALGORİTMASININ TERSİNE MÜHENDİSLİK İLE YORUMLANMASI

Hill algoritmasını ikili bloklar halinde çalıştırarak elde edilen sonuçlarda verilen açık metin "ahmetmunganj" idi. Son bloğu boşta kalmasın diye türk alfabesinde frekansı en düşük harf olan 'j' harfi eklenerek bir açık metin oluşturulmuştur. Dışarıdan bakılınca her şey normal gözükse de açık metin içeriği değiştiği zaman, ikili bloklarda da tekli bloklarda uygulanan frekans çözümleme tekniği ile ciddi güvenlik zaafiyetleri bulunabilir. Dolayısıyla ikili bloklarda tekrar eden iki harfin yan yana gelmesinden oluşan içerikler indekslenerek, zaafiyet yaratılır. Bu pek tabii istenmeyen bir durumdur.

Hill algoritmasının zayıf kalan yanı aslında tam olarak algoritmasıdır. Algoritma fikren iş yapılma potansiyeli taşıyan bir algoritmadır fakat güvenli olmadığı yadsınamaz bir gerçektir. Tek başına bir sistemi koruması, şifrenin deşifre edilememesi gibi durumlar özellikle günümüz teknolojileri ile pek mümkün değildir.

Başka çözüm algoritmaları halihazırda var fakat bu algoritmanın geliştirilmesini sağlamak mümkündür. Örneğin; sezar algoritmasında öteleme sayısının her harf için değiştirildiğini düşünürsek, bu öteleme sayı dizisini kayıt altında tutarsak -ki bu kayda gizli anahtar denir- ilk akla gelmeyen, şifre kırıcıların daha evvel belki de karşılaşmadığı bir model ortaya koymuş oluruz. Başka algoritmalarda kullanılmak üzere geliştirilen modeller şunlardır:

1. Kapalı metin zincirleme modeli
2. Çıktıyı geri besleme modeli
3. Girdiyi geri besleme modeli

Birinci modelde; özet olarak şifreli metin bloğunu/bloklarını iki farklı algoritmaya tabi tutarak bir model oluşturmaktır. İkinci ve üçüncü modelde; şifreli metnin açık metni etkilediği veya açık metnin şifreli metni etkilediği ve dolayısıyla algoritmasının da değiştiği bir model oluşturmaktadır.

Tüm bu durumlar göz önüne alınarak gerçekleştirilen hill algoritması, beslemeli ya da zincire dayalı bir modelle güçlendirilebilir.

## REFERANS VE KAYNAKÇA


- P. Bassia, I. Pitas, “Robust audio watermarking in the time domain” Proc. EUSIPCO 98
- İTÜBİDB, Şifreleme Yöntemleri – Link için tıklayınız.
- W. Stallings, “Kriptografi ve Siber Güvenlik Prensipleri ve Uygulamaları”, 4. Baskı, Prentice Hall Publication, 2005
- Kriptolojiye Giriş Ders Notları, Uygulamalı Matematik Enstitüsü, Kriptografi Bölümü, ODTÜ, 2004
- C. Koçak, “Kriptografi ve stenografi yöntemlerini birlikte kullanarak yüksek güvenlikli veri gizleme”, Erciyes Üniversitesi Fen Bilimleri Enstitüsü Dergisi, 31(2):115-123, Bilgisayar Mühendisliği, Gazi Üniversitesi, 2015

## EKLER

- Geçmiş raporların paylaşıldığı github linki için tıklayınız.