Nev: Hegyesi Akos

NEPTUN kod: HSFPOJ

One Time Pad

A következő titkosítási módszert szeretnénk megtámadni.

A plaintext-tér a 200 byte hosszú sztringek halmaza. A kulcstér szintén a 200 byte hosszú kulcsok halmaza, viszont ezek közül nem egyenletesen véletlenül választunk, hanem úgy generáljuk a kulcsot, hogy minden bitje 55%-kal 0, 45%-kal 1 (tehát várhatóan kb. az 1600 bitből 880 db 0 és csak 720 db egyes).

A titkosítás ezután "sima bitenkénti xor", ld. pl. az órai encrypt_xor függvényt.

A cél, hogy a következő játékot minél jobb valószínűséggel megnyerjük:

a "csupa A" és "csupa B", azaz az

m0 = "A" * 200

m1 = "B" * 200

sztringek közül valamelyiket véletlen kulccsal letitkosították. Meg kell tippelnünk, melyiket. Találjunk ki olyan módszert, mely az esetek minél nagyobb hányadában jól tippel.

In [1]:
```
# One-time pad -> ez adja az alap otletet
def encrypt_xor(plain, key):
    n = min(len(plain), len(key))
    ascii_codes = [ord(plain[i]) ^ ord(key[i])  for i in range(n)]
    return ''.join([chr(x) for x in ascii_codes])
```

In [2]:
```
# Fontos fogalom lesz itt a predikalhatosag

# Predikalhatosag alatt azt ertjuk, hogy letezik olyan algoritmus, amely Xi
#   j bites prefixe alapjan a (j+1)-edik bitet nem elhanyagolhato valoszin

# mivel itt tudjuk, hogy a kulcs minden bitje 55%-kal 0, es 45%-kal 1, igy
#   kulcs kovetkezo bitjenek erteket

# alveletlen valoszinusegi valtozok egyutteset akkor kaphatnank csak, ha n
```

In [3]:
```
# titkositas visszafejtesenek alap otlete:
#    ha egy mar titkositott szoveget a kulccsal ismet XOR-olunk, visszakap
# az alabbi pelda bemutatja a fentebb emlitett visszfejtesi strategiat
print(ord('A'))
print(ord('A') ^ 1)
print((ord('A') ^ 1) ^ 1)
```

```
65
64
65
```

Ez a megoldas nem bitenkenti titkositas, itt egy adott karaktert toltunk el 0 vagy 1 szammal

A RENDES MEGOLDAS LENTEBB LATHATO!!

In [4]:
```
# A "rossz" pseudo random sorozat generalasa -> 1. modszer
# 1 es 100 kozotti szam, ha kisebb, mint 56, akkor 0, ha nagyobb, akkor 1 

import math
import random

key = []

def generateRandomBitList (key, x):

    for i in range(0, x):
        rnd = random.randint(1,100)
        if (rnd < 56):
            tmp = 0
            key.append(tmp)
        else:
            tmp = 1
            key.append(tmp)
    return key

key = generateRandomBitList(key, 200)
print("0 aranya: ", key.count(0) / len(key))
print("1 aranya: ", key.count(1) / len(key))
```

```
0 aranya:  0.535
1 aranya:  0.465
```

In [5]:
```
# Egyeb rossz pseudo random szam generalas otlet -> 100 es 1 000 000 kozot
# Ha n % 100 < 55 -> bit = 0, else bit = 1

def otherBitListGeneration(key, x):
    for i in range(x):
        rnd = random.randint(100, 1000000)
        if (rnd % 100) < 55:
            key.append(0)
        else:
            key.append(1)

    return key

key2 = []
key2 = otherBitListGeneration(key2, 1000)

print("0 aranya: ", key2.count(0) / len(key2))
print("1 aranya: ", key2.count(1) / len(key2))
```

```
0 aranya:  0.573
1 aranya:  0.427
```

In [6]:
```
# otlet a jatekhoz:
#   -> amennyiben az XOR muveletnel 1-es bittel csokkentenenk az 'A' unico
#   -> igy mar csak a statisztikai alapokra tudunk tamaszkodni, es megnezn

#   -> amennyiben az eredeti szoveg "A"*200 volt, ugy a chiper textben nag
#   -> amennyiben az eredeti szoveg "B"*200 volt, ugy a chiper textben nag

# a kulcsba generalt gyakorisag ismereteben igy mar tamadhatova valik a ch


def xorEncription(plain, key):
    n = min(len(plain), len(key))
    if plain[0] == 'A':
        ascii_codes = [(ord(plain[i]) ^ key[i]) if key[i] == 0 else ((ord(
        return ''.join([chr(x) for x in ascii_codes])
    else:
        ascii_codes = [(ord(plain[i]) ^ key[i]) if key[i] == 0 else ((ord(
        return ''.join([chr(x) for x in ascii_codes])


def generateChiperText():
    chiper = ""
    m0 = 200*'A'
    m1 = 200*'B'
    key = []
    key = generateRandomBitList(key, 1600)

    rand_for_chiper = random.randint(0,1)

    if rand_for_chiper == 0:
        chiper = xorEncription(m0, key)
    else:
        chiper = xorEncription(m1, key)

    return chiper

chiper = generateChiperText()

print("Titkositott szoveg:")
print()
print (chiper)
```

```
Titkositott szoveg:

AAAABABBBAAABAABAAAAABAAABBAABABBABAAAABABBBABBBAABBAABABBBBBAABBBABAABBAAA
ABABAABBAABABAABABABBBABABAAABBABBBAAABBABBBAABBBAABABBBABBABABBBBBAABBBABB
BAAAABBABBBAABBBBABBAABBBBAAABBAAAABABAAAABBBBBBB
```

In [7]:
```
# Megoldas a karakterenkenti eltolasra
def guessingOriginalMessage(chiper):
    if chiper.count('A') > chiper.count('B'):
        return "200*A"
    else:
        return "200*B"


answer = guessingOriginalMessage(chiper)
print("'A' betuk szama: ", chiper.count('A'))
print("'A' betuk aranya: ", chiper.count('A') / len(chiper))
print()
print("'B' betuk szama: ", chiper.count('B'))
print("'B' betuk aranya: ", chiper.count('B') / len(chiper))
print()
print("Az eredeti szoveg a: ", answer, " volt!")
```

```
'A' betuk szama:  96
'A' betuk aranya:  0.48

'B' betuk szama:  104
'B' betuk aranya:  0.52

Az eredeti szoveg a:  200*B  volt!
```

RENDES MEGOLDAS -> N es M karakterek vizsgalataval

In [8]:
```
# bitenkenti titkositas
# szeretnem kihasznalni az asszimetrikussagot, hogy tobb 0-s van, mint aha

def convert_binary_list_to_str(L):
    ret = ""
    for i in range(math.floor(len(L) / 8)):
        ret += chr( sum([2^k * L[8*i+k] for k in range(8)] ) )
    return ret

def output_random_ciphertext_ex1():
    m0 = "A"*200
    m1 = "B"*200
    plain = m0 if random.randint(0,1) == 0 else m1

    key_new = []
    key_new = generateRandomBitList(key_new, 1600)

    k = convert_binary_list_to_str(key_new)
    return encrypt_xor(plain, k), plain[0]

chiper_new, letter = output_random_ciphertext_ex1()
print("Titkositott szoveg:")
print()
print(chiper_new)
print()
print("Az eredeti szoveg: ", letter, "* 200")
print()

# Counter seged konyvtar hasznalata, hogy a leggyakoribb eseteket megtalal
from collections import Counter

def lettersOfChiperText() :
    letters = []
    letters.append(chiper_new[0])
    for i in range(len(chiper_new)):
        for j in range(len(letters)):
            if chiper_new[i] not in letters:
                letters.append(chiper_new[i])

    return letters
```

```
Titkositott szoveg:

M_[LVO\RWTLPRUU\QPQZZRT^QVWQV[UTRXZXWPVUQUQWVYUWPMRQ_\WPWRQQXLQUQRMPPZWSTQT
LQSZZQWRV^[PUYUWQQMZ^WPXZZSXZPZTZZXXXZUZ\P[PQXRRYYUUWYPRULPRTX^SR[QTMMWPXMX
QSURZVTO[PTVV[ZVOZWSMUTTUYSZXUPYZQUTVPWTVURUYTWW[W

Az eredeti szoveg:  B * 200
```

In [9]:
```
# leggyakoribb elemeket vizsgalom a titkositott szovegben
def mostCommonLetters(chiper):
    return Counter(chiper).most_common()

# gyakorisag viszgalat a leggyakrabban elofordulo elemekre
print("Leggyakoribb elemek:")
print()
most_common = mostCommonLetters(chiper_new)
print(most_common)
```

```
Leggyakoribb elemek:

[('U', 20), ('Q', 20), ('Z', 20), ('W', 18), ('P', 18), ('R', 15), ('T', 15
), ('X', 13), ('V', 12), ('M', 8), ('[', 8), ('Y', 8), ('S', 7), ('L', 5),
('\\', 4), ('^', 4), ('O', 3), ('_', 2)]
```

In [10]:
```
# MEGOLDAS!!

# megfigyeles -> gyakran ismetlodo lepes, hogy ha 'A'*200 volt az eredeti
# a megfigyelesekbol azt a kovetkeztetest vontam le, hogy valamelyik edge
# mivel vagy az egyik, vagy csak a masik fordul elo, igy eleg jo megbizhat

def letterNOrletterM(chiper):
    counter_n = 0
    counter_m = 0
    for i in range(len(chiper)):
        if chiper[i] == 'N':
            counter_n += 1
        elif chiper[i] == 'M':
            counter_m = 0
    if counter_n > counter_m:
        return 'A'
    else:
        return 'B'
```

In [27]:
```
# A legjobb megbizhatosagot az hozta, amelyik esetben azt nezzuk, hogy van
# HA
#  -> N betut tartalmaz a titkositott szoveg, akkor feltetelezzuk, hogy cs
#  -> M betut tartalmaz a titkositott szoveg, akkor feltetelezzuk, hogy cs
# amiatt megbizhato, mert velhetoleg van egy olyan minimalis/maximalis elt

def ratio (N):
    ret = 0
    for i in range(N):
        c,m = output_random_ciphertext_ex1()
        s = letterNOrletterM(c)
        if s == m:
            ret += 1
    return ret

ret = ratio(1000)
print()
print('A keresett karakter:', letter)
print('1000 esetbol az eltalalas aranya a specialis N es M karakter keresés
```

```
A keresett karakter: B
1000 esetbol az eltalalas aranya a specialis N es M karakter keresesevel:
0.997
```

Tovabbi otletelesek, melyeken egyeb irant gondolkoztam a megoldas keresese kozben

In [12]:
```
# otlet -> gyakorisagi vizsgalat a leggyakoribb elem oszthatosagara, ez egy
# otlet: ord('A') paratlan, es ord('B') paros
# probalgatasok alapjan azt figyeltem meg, ha ord('A')-hoz adok kevesebb 1-
# ezt kiprobalva azt figyelem meg, hogy a leggyakoribb elem, amit kaptunk,


def commonAOrCommonB(most_common):
    counter_a = 0
    counter_b = 0
    if (ord(most_common[0][0]) % 2 == 0):
        return 'A'
    elif (ord(most_common[0][0]) % 2 == 1):
        return 'B'
```

In [20]:
```
# Ez a megoldas nagyjabol 55%-60% kornyeki megbizhatosagot ad
def ratio (N):
    ret = 0
    for i in range(N):
        c,m = output_random_ciphertext_ex1()
        comm = mostCommonLetters(c)
        s = commonAOrCommonB(comm[0][0])
        if s == m:
            ret += 1
    return ret

print()
print('A keresett karakter:', letter)
print('1000 esetbol az eltalalas aranya az osztasi maradekos modszerrel: '
```

```
A keresett karakter: B
1000 esetbol az eltalalas aranya az osztasi maradekos modszerrel:  0.564
```

In [14]:
```
def badTipp(most_common):
    first1 = ord(most_common[0][0]) ^ ord('B')
    first2 = ord(most_common[0][0]) ^ ord('A')
    if (first1 > first2):
        return 'A'
    else:
        return 'B'
```

In [15]:
```
# tipp -> tfh, mivel 0 valoszinusege 0,55, igy elkepzelheto, hogy letezik
# az xor tulajdonsagat kihasznalva, miszerint, ha egy karakter ketszer egy
# elkeszitjuk az ord() fuggveny meghivasaval a [0,0,0,0,0,0,0,0] bitsoroza
# a chiper textet vegig megyunk, es minden karakterhez hozza xor-oljuk, es

asd = [0,0,0,0,0,0,0,0]

def AorB(chiper):
    counter_a = 0
    counter_b = 0
    for i in range(len(chiper)):
        if (ord(chiper[i]) ^ ord(convert_binary_list_to_str(asd))) == ord(
            counter_a += 1
        elif (ord(chiper[i]) ^ ord(convert_binary_list_to_str(asd))) == ord
            counter_b += 1
    if counter_a > counter_b:
        return 'A'
    else:
        return 'B'

# otlet -> ord('A') paratlan, ord('B') paros, igy ha ord('A')-hoz adok hoz
def evenOrOdd(chiper):
    counter_a = 0
    counter_b = 0
    for i in range(len(chiper)):
        if ((ord(chiper[i]) ^ ord('A')) % 2 == 1):
            counter_a += 1
        elif ((ord(chiper[i]) ^ ord('B')) % 2 == 0):
            counter_b += 1
    if counter_a > counter_b:
        return 'A'
    else:
        return 'B'
```

In [16]:
```
print('a leggyakoribb elem vizsgalataval kapott tipp: ', badTipp(most_comm
print("'A' es 'B' szamlalasabol kapott eredmeny: ", AorB(chiper_new))
print("A paros es paratlan szamlalasbol kijott eredmeny: ", evenOrOdd(chip
```

```
a leggyakoribb elem vizsgalataval kapott tipp:  A
'A' es 'B' szamlalasabol kapott eredmeny:  A
A paros es paratlan szamlalasbol kijott eredmeny:  A
```