Kryptografia Projekt

Wiktor Skrzypczak

Algorytm szyfrowania

Szyfr zaimplementowany w programie jest szyfrem symetrycznym – nadawca i odbiorca wiadomości używają tego samego klucza do kodowania i dekodowania szyfrogramu. Szyfrowanie odbywa się w dwóch etapach.

Pierwszy etap opiera się na wariacji kolumnowego szyfru przestawieniowego – tekst jawny dzielony jest (z uwzględnieniem spacji i znaków specjalnych) na segmenty o długości użytego klucza symetrycznego. Następnie znaki zawarte w segmentach przestawiane są zgodnie z kluczem.

Drugi etap polega na przesunięciu pozycji znaków uzyskanego po pierwszym etapie kryptogramu w tablicy ASCII o określone w programie wartości.

Kodowanie:

Etap I:

Przykładowy tekst jawny: *Konstantynopolitańczykowianeczka*

Przykładowy klucz: [5, 3, 6, 4, 2, 1]

Długość klucza: 6

Tekst dzielony jest na 6cio znakowe segmenty:

Konsta-ntynop-olitan-czykow-ianecz-ka

Jeśli tekst jawny nie jest podzielny przez długość klucza program powiększa szyfrowany ciąg znakowy o brakującą ilość spacji.

Następnie znaki w segmentach przestawiane są na podstawie klucza w następujący sposób:

K → Przestawienie na pozycje 5tą

o → Przestawienie na pozycje 3cią

n → Przestawienie na pozycje 6tą

s → Przestawienie na pozycje 4tą

t → Przestawienie na pozycję 2gą

a → Przestawienie na pozycję 1szą

Zaszyfrowany segment: *atosKn*

W taki sam sposób szyfrowana jest reszta segmentów.

Otrzymany kryptogram: *tnasoKoypntnaintlooywkzccnzeai ak*

Etap II:

Otrzymany po etapie I kryptogram: *tnasoKoypntnaintlooywkzccnzeai ak*

W celu ukrycia znaków faktycznie składających się na zaszyfrowany tekst jawny etap II polega na szyfrowaniu przy pomocy tablicy ASCII.

Znaki otrzymanego kryptogramu przestawiane są w tablicy ASCII w następujący sposób:

- Znaki na pozycji ASCII > 79 przestawiane są o 48 pozycji w dół
- Znaki na pozycji ASCII < 79 przestawiane są o 48 pozycji w górę
- Znak na pozycji ASCII = 79 nie jest przestawiany.

Otrzymany kryptogram: D>1C?{?I@>D>19>D<??IG;J33>J519PPPP1;

Dekodowanie:

W celu odszyfrowania kryptogramu odbiorca wiadomości potrzebuje użytego do zaszyfrowania klucza symetrycznego. Program pozwala użytkownikowi na generowanie kluczy o zadanej długości, a także na ich ręczne wprowadzenie.

Funkcje:

Funkcja *cipher* – funkcja kodująca tekst jawny

```
def cipher(x, y)
    v=0
    newString1=""
    for ix in range(int(len(x)/len(y))): # Przestaw znaki wedl
ug klucza
        for iy in y:
            pos=v-1+iy
            newString1+=x[pos]
        v=(ix+1)*len(y)
    newString2=""
    for iz in newString1: # Przesun znaki w tablicy ASCII
        if ord(iz) > 79:
            newString2+=chr(ord(iz)-48)
        elif ord(iz) < 79:
            newString2+=chr(ord(iz)+48)
        else:
            newString2+=iz
    return newString2
```

Funkcja *appender* – funkcja dodająca spacje na końcu tekstu jawnego

```
def appender(x, y):
    while (float(len(x))%float(len(y))!=0): # Sprawdz czy teks
t jawny jest podzielny przez dlugosc klucza
        x=x+" " # Dodaj spacje i powtorz sprawdzenie
    return x
```

Funkcja *decipher* – funkcja dekodujaca kryptogram

```
def decipher(x, y):
    v=0
    newString1=""
    for ix in x: # Przesun znaki w tablicy ASCII
        if ord(ix) > 79:
            newString1+=chr(ord(ix)-48)
        elif ord(ix) < 79:
            newString1+=chr(ord(ix)+48)
        else:
            newString1+=ix
    newList=list(newString1)
    for ix in range(int(len(x)/len(y))): # Przestaw znaki wedl
ug klucza
       c=0
       for iy in y:
           pos=v-1+iy
           newList[pos]=newString1[v+c]
           c+=1
       v=(ix+1)*len(y)
    newString2=""
    return newString2.join(newList)
```

Funkcja *keyGenerator* – Funkcja automatycznie generująca losowy klucz symetryczny

```
def keyGenerator(xStop, defKey):
    xStart=1
    if 1 <= (xStop-1):
        newKey=random.sample(range(xStart, xStop), (xStop-1))
        print(f"Twoj nowy klucz to: {newKey}")
        return newKey
    else:
        print("Nieprawidlowa wielkosc klucza. Sprobuj ponownie
.")
        return defKey
```

Funkcja *keyImplementer* – funkcja implementująca instniejący klucz/pozwalająca na utworzenie własnego.

```
def keyImplementer(xKeyLength, defKey):
    ans2=True
    if 1 <= xKeyLength:
        newKey=[]
        for ik in range(xKeyLength):
            newKey.append(int(input(f"Podaj liczbe na pozycji
{ik+1}: ")))
        for i2k in range(xKeyLength):
            if (i2k+1) not in newKey:
                ans2=False
        if ans2==True:
            print(f"Zmieniono klucz. Twoj nowy klucz to: {newK
ey}")
            return newKey
        elif ans2==False:
            print(f"Klucz nieprawidlowy. Musi zawierac liczby
z przedzialu od 1 do {xKeyLength}, bez powtorzen. Sprobuj pono
wnie")
    else:
        print("Nieprawidlowa wielkosc klucza. Sprobuj ponownie
.")
        return defKey
```