**Харківський національний економічний університет**

**імені Семена Кузнеця**

**ЗВІТ**

**З ВИКОНАННЯ ЛАБОРАТОРНОЇ РОБОТИ № 4**

**за дисципліною: *“*** **Теоретичні основи криптографії** ”

**на тему: “Шифрування з відкритим ключем на основі задачі рюкзака”**

**Виконав: студент факультету Інформаційних технологій**

**2 курсу, спец. Кібербезпека,**

**групи 6.04.125.010.21.2**

**Бойко Вадим Віталійович**

**Перевірив:**

**Чугай Андрій Михайлович**

**ХНЕУ ім. С. Кузнеця**

**2023**

Мета: Реалізувати Шифрування з відкритим ключем на основі задачі рюкзака

Завдання:

1. Відшукайте в Інтернет-ресурсах чисельний приклад з використання «рюкзачного» алгоритму (наприклад, в Вікіпедії ) та опрацюйте його.
2. Розробіть інтерфейс криптографічної системи для шифрування з використанням задачі рюкзака, передбачивши окремий діалог для формування відкритого ключа.
3. Розробіть методи, які б забезпечували:
   1. Генерацію пари «відкритий -закритий» ключі.
   2. Шифрування з використанням відкритого ключа.
   3. Розшифрування з використанням закритого ключа. При цьому значення t-1 вважати відомим.
4. Перевірте правильність роботи системи на основі використання даних з чисельного прикладу.

Хід роботи:

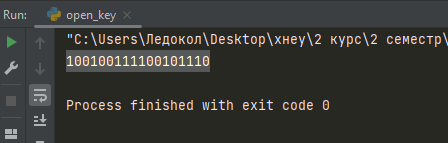
1. Я пропрацював приклад в інтернеті

2. Реалізував наступне рішення

private\_key = [1, 2, 4, 10, 20, 40]  
n = 7  
  
m = sum(private\_key) + 10  
  
*def* gcdExtended(*a*, *b*):  
 *if a* == 0 :  
 *return b*, 0, 1  
 gcd1, x1, y1 = gcdExtended(*b* % *a*, *a*)  
 x = y1 - (*b* // *a*) \* x1  
 y = x1  
 *return* gcd1, x, y  
  
gcd1, x, y = gcdExtended(n, m)  
  
mn = (x % m + m) % m  
  
*def* generate\_key\_based\_on\_private\_key(*private\_key* = []):  
 public\_key = []  
 *for* number *in private\_key*:  
 public\_key.append(number \* n % m)  
 *return* public\_key  
  
*def* myltiplication(*message* = '', *key* = []):  
 sum = 0  
 *for* i *in* range(len(*message*)):  
 *if message*[i] == '1':  
 sum += *key*[i]  
 *return* sum  
  
*def* encryption(*message\_for\_encryption* = '', *public\_key* = []):  
 l = len(*public\_key*)  
 message\_partials = [*message\_for\_encryption*[x:x + l] *for* x *in* range(0, len(*message\_for\_encryption*), l)]  
 encrypted\_value = []  
 *for* partial *in* message\_partials:  
 encrypted\_value.append(myltiplication(partial, *public\_key*))  
 *return* encrypted\_value  
  
*def* decryption(*message\_for\_decryption* = [], *private\_key* = []):  
  
 partial\_values = []  
 *for* part *in message\_for\_decryption*:  
 partial\_values.append(part \* mn % m)  
 result = ''  
 reversed\_private\_key = *private\_key*[::-1]  
 *for* partial *in* partial\_values:  
 r = ''  
 s = 0  
 *for* value *in* reversed\_private\_key:  
 *if* value + s <= partial:  
 s += value  
 r += '1'  
 *else*:  
 r += '0'  
 result += r[::-1]  
 print(result)  
  
public\_key = generate\_key\_based\_on\_private\_key(private\_key) # [31, 62, 14, 90, 70, 30]  
  
message\_for\_encryption = '100100111100101110'  
  
encrypted\_value = encryption(message\_for\_encryption, public\_key)  
  
decryption(encrypted\_value, private\_key)

цей код шифрує та шифрує дане повідомлення «100100111100101110»

а потім дешифрує



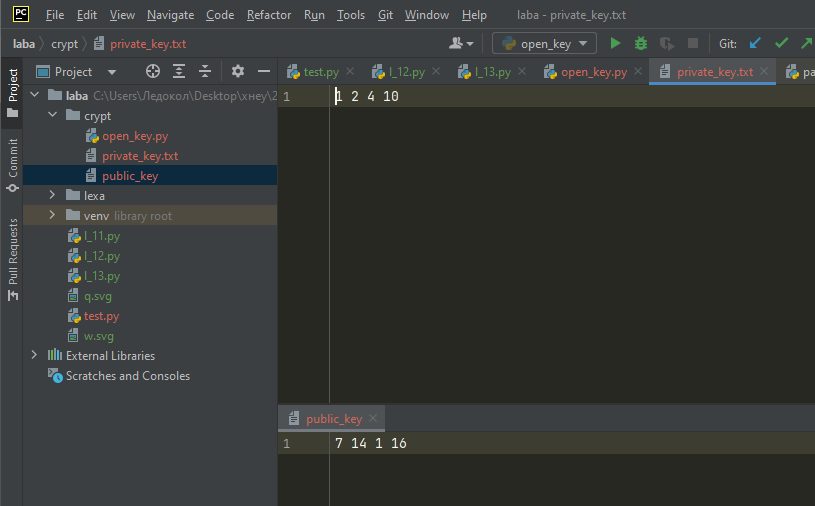
3. розроблюю інтерфейси

1. для генерації пари ключів

Я реалізував наступне рішення

*def* input\_private\_key():  
 private\_key = list(map(int, input('Input private key\n').split()))  
 s = 0  
 *for* n *in* private\_key:  
 *if* n > s:  
 s += n  
 *else*:  
 *break  
 if* s < sum(private\_key):  
 print('Private key is not correct')  
 *return* input\_private\_key()  
 *else*:  
 *with* open('private\_key.txt', 'w') *as* file:  
 file.write(' '.join(map(str, private\_key)))  
 *return* private\_key  
  
  
private\_key = input\_private\_key()  
n = 7  
  
m = sum(private\_key) + 10  
  
*def* gcdExtended(*a*, *b*):  
 *if a* == 0 :  
 *return b*, 0, 1  
 gcd1, x1, y1 = gcdExtended(*b* % *a*, *a*)  
 x = y1 - (*b* // *a*) \* x1  
 y = x1  
 *return* gcd1, x, y  
  
gcd1, x, y = gcdExtended(n, m)  
  
mn = (x % m + m) % m  
  
*def* generate\_key\_based\_on\_private\_key(*private\_key* = []):  
 public\_key = []  
 *for* number *in private\_key*:  
 public\_key.append(number \* n % m)  
 *with* open('public\_key', 'w') *as* file:  
 file.write(' '.join(map(str, public\_key)))  
 *return* public\_key  
  
*def* myltiplication(*message* = '', *key* = []):  
 s = 0  
 *for* i *in* range(len(*message*)):  
 *if message*[i] == '1':  
 s += *key*[i]  
 *return* s  
  
*def* encryption(*message\_for\_encryption* = '', *public\_key* = []):  
 l = len(*public\_key*)  
 message\_partials = [*message\_for\_encryption*[x:x + l] *for* x *in* range(0, len(*message\_for\_encryption*), l)]  
 encrypted\_value = []  
 *for* partial *in* message\_partials:  
 encrypted\_value.append(myltiplication(partial, *public\_key*))  
 *return* encrypted\_value  
  
*def* decryption(*message\_for\_decryption* = [], *private\_key* = []):  
  
 partial\_values = []  
 *for* part *in message\_for\_decryption*:  
 partial\_values.append(part \* mn % m)  
 result = ''  
 reversed\_private\_key = *private\_key*[::-1]  
 *for* partial *in* partial\_values:  
 r = ''  
 s = 0  
 *for* value *in* reversed\_private\_key:  
 *if* value + s <= partial:  
 s += value  
 r += '1'  
 *else*:  
 r += '0'  
 result += r[::-1]  
 print(result)  
  
public\_key = generate\_key\_based\_on\_private\_key(private\_key) # [31, 62, 14, 90, 70, 30]  
  
message\_for\_encryption = '100100111100101110'  
  
encrypted\_value = encryption(message\_for\_encryption, public\_key)  
  
decryption(encrypted\_value, private\_key)

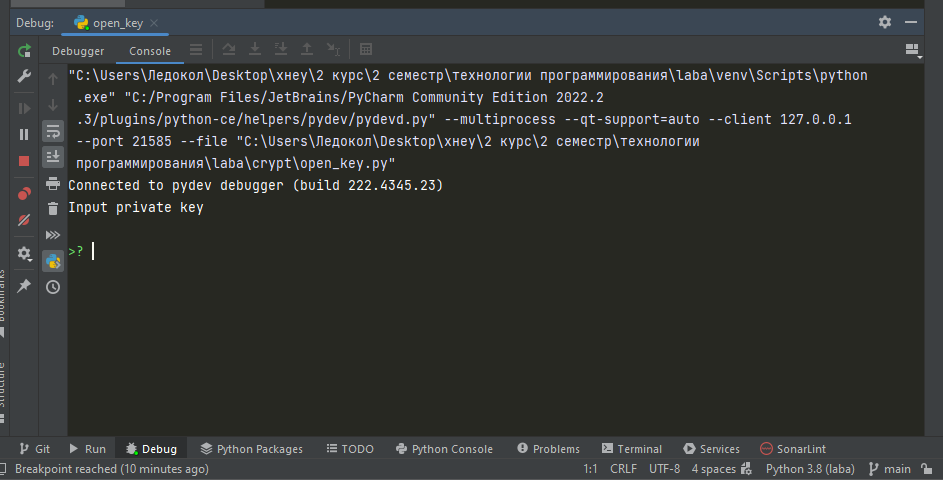
в результаті створюється дві пари ключів



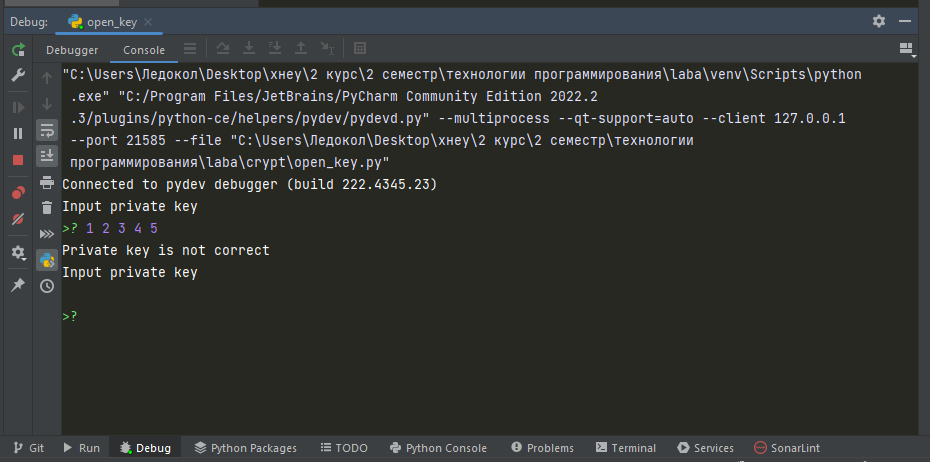
2, 3 також у цьому рішенні є шифрування і розшифрування

*def* input\_private\_key():  
 private\_key = list(map(int, input('Input private key\n').split()))  
 s = 0  
 *for* n *in* private\_key:  
 *if* n > s:  
 s += n  
 *else*:  
 *break  
 if* s < sum(private\_key):  
 print('Private key is not correct')  
 *return* input\_private\_key()  
 *else*:  
 *with* open('private\_key.txt', 'w') *as* file:  
 file.write(' '.join(map(str, private\_key)))  
 *return* private\_key  
  
  
private\_key = input\_private\_key()  
n = 7  
  
m = sum(private\_key) + 10  
  
*def* gcdExtended(*a*, *b*):  
 *if a* == 0 :  
 *return b*, 0, 1  
 gcd1, x1, y1 = gcdExtended(*b* % *a*, *a*)  
 x = y1 - (*b* // *a*) \* x1  
 y = x1  
 *return* gcd1, x, y  
  
gcd1, x, y = gcdExtended(n, m)  
  
mn = (x % m + m) % m  
  
*def* generate\_key\_based\_on\_private\_key(*private\_key* = []):  
 public\_key = []  
 *for* number *in private\_key*:  
 public\_key.append(number \* n % m)  
 *with* open('public\_key', 'w') *as* file:  
 file.write(' '.join(map(str, public\_key)))  
 *return* public\_key  
  
*def* myltiplication(*message* = '', *key* = []):  
 s = 0  
 *for* i *in* range(len(*message*)):  
 *if message*[i] == '1':  
 s += *key*[i]  
 *return* s  
  
*def* encryption(*message\_for\_encryption* = '', *public\_key* = []):  
 l = len(*public\_key*)  
 message\_partials = [*message\_for\_encryption*[x:x + l] *for* x *in* range(0, len(*message\_for\_encryption*), l)]  
 encrypted\_value = []  
 *for* partial *in* message\_partials:  
 encrypted\_value.append(myltiplication(partial, *public\_key*))  
 *return* encrypted\_value  
  
*def* decryption(*message\_for\_decryption* = [], *private\_key* = []):  
  
 partial\_values = []  
 *for* part *in message\_for\_decryption*:  
 partial\_values.append(part \* mn % m)  
 result = ''  
 reversed\_private\_key = *private\_key*[::-1]  
 *for* partial *in* partial\_values:  
 r = ''  
 s = 0  
 *for* value *in* reversed\_private\_key:  
 *if* value + s <= partial:  
 s += value  
 r += '1'  
 *else*:  
 r += '0'  
 result += r[::-1]  
 print(result)  
  
public\_key = generate\_key\_based\_on\_private\_key(private\_key)  
  
message\_for\_encryption = input('Input phrase for encryption')# 100100111100101110'  
  
encrypted\_value = encryption(message\_for\_encryption, public\_key)  
  
decryption(encrypted\_value, private\_key)

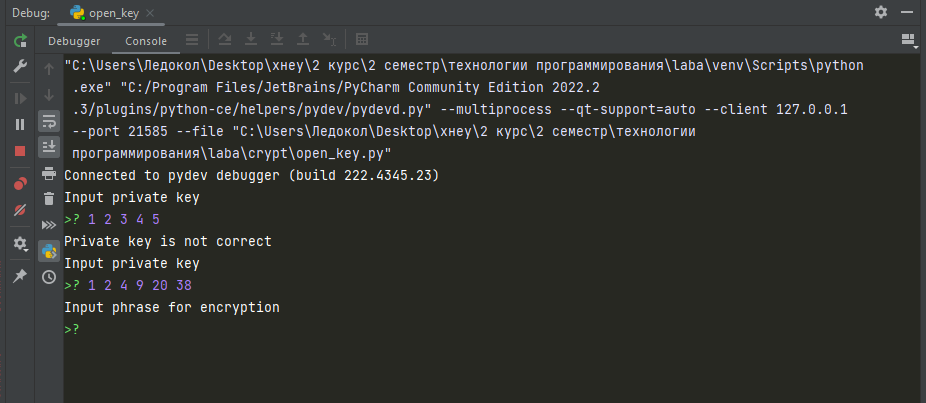
4. Приклад використання



Якщо ключ не правильний, програма видає, що ключ не правильний



Якщо ключ правильний, тоді програма запитує про ввод фрази для шифрування

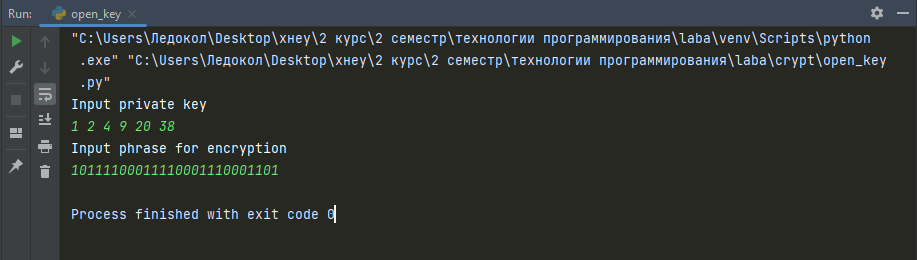


Потрібно ввести фразу для шифрування у вигляді 1 і 0

Трохи дописав код, щоб записувало дані у файли

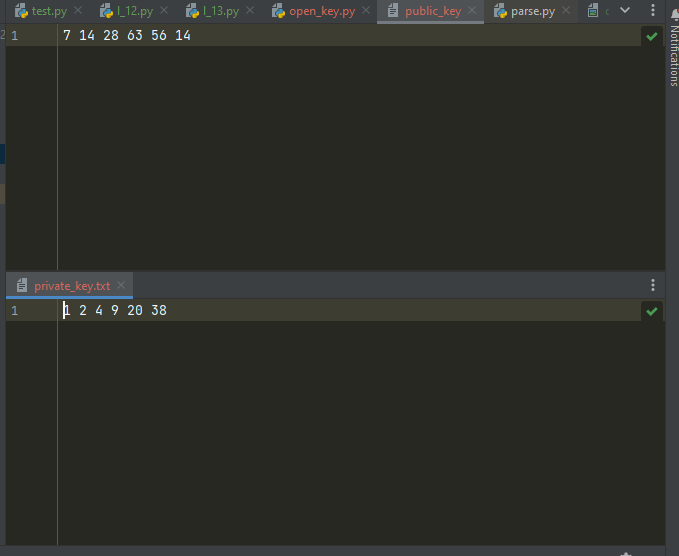
*def* input\_private\_key():  
 private\_key = list(map(int, input('Input private key\n').split()))  
 s = 0  
 *for* n *in* private\_key:  
 *if* n > s:  
 s += n  
 *else*:  
 *break  
 if* s < sum(private\_key):  
 print('Private key is not correct')  
 *return* input\_private\_key()  
 *else*:  
 *with* open('private\_key.txt', 'w') *as* file:  
 file.write(' '.join(map(str, private\_key)))  
 *return* private\_key  
  
  
private\_key = input\_private\_key()  
n = 7  
  
m = sum(private\_key) + 10  
  
*def* gcdExtended(*a*, *b*):  
 *if a* == 0 :  
 *return b*, 0, 1  
 gcd1, x1, y1 = gcdExtended(*b* % *a*, *a*)  
 x = y1 - (*b* // *a*) \* x1  
 y = x1  
 *return* gcd1, x, y  
  
gcd1, x, y = gcdExtended(n, m)  
  
mn = (x % m + m) % m  
  
*def* generate\_key\_based\_on\_private\_key(*private\_key* = []):  
 public\_key = []  
 *for* number *in private\_key*:  
 public\_key.append(number \* n % m)  
 *with* open('public\_key', 'w') *as* file:  
 file.write(' '.join(map(str, public\_key)))  
 *return* public\_key  
  
*def* myltiplication(*message* = '', *key* = []):  
 s = 0  
 *for* i *in* range(len(*message*)):  
 *if message*[i] == '1':  
 s += *key*[i]  
 *return* s  
  
*def* encryption(*message\_for\_encryption* = '', *public\_key* = []):  
 l = len(*public\_key*)  
 message\_partials = [*message\_for\_encryption*[x:x + l] *for* x *in* range(0, len(*message\_for\_encryption*), l)]  
 encrypted\_value = []  
 *for* partial *in* message\_partials:  
 encrypted\_value.append(myltiplication(partial, *public\_key*))  
 *return* encrypted\_value  
  
*def* decryption(*message\_for\_decryption* = [], *private\_key* = []):  
  
 partial\_values = []  
 *for* part *in message\_for\_decryption*:  
 partial\_values.append(part \* mn % m)  
 result = ''  
 reversed\_private\_key = *private\_key*[::-1]  
 *for* partial *in* partial\_values:  
 r = ''  
 s = 0  
 *for* value *in* reversed\_private\_key:  
 *if* value + s <= partial:  
 s += value  
 r += '1'  
 *else*:  
 r += '0'  
 result += r[::-1]  
 *return* result  
  
public\_key = generate\_key\_based\_on\_private\_key(private\_key)  
  
message\_for\_encryption = input('Input phrase for encryption\n')# 100100111100101110'  
  
encrypted\_value = encryption(message\_for\_encryption, public\_key)  
*with* open('encrypted\_value.txt', 'w') *as* file:  
 file.write(' '.join(map(str, encrypted\_value)))  
  
decrypted\_value = decryption(encrypted\_value, private\_key)  
*with* open('decrypted\_value.txt', 'w') *as* file:  
 file.write(decrypted\_value)

Перевірка



Та створені файли

Публічний та приватний ключ



Шифроване та розшифроване значення

