Учреждение образования

«Белорусский государственный технологический университет»

**Лабораторная работа №9**

Исследование ассиметричных шифров

Выполнил:

Студент 3 курса 5 группы ФИТ

Коршун Никита Игоревич

2024

1. **Криптоалгоритм на основе задачи о ранце**

Проблема укладки ранца формулируется просто. Дано множество предметов общим числом *z* различного веса. Спрашивается, можно ли положить некоторые из этих предметов в ранец так, чтобы его вес стал равен определенному значению *S*? Более формально задача формулируется так: дан набор значений *k1*, *k2*,…, *kz* и суммарное значение *S*. Требуется вычислить значения *bz* такие, что справедливо выражение, представленное на формуле 1.1.

 (1.1)

Суть метода для шифрования состоит в том, что существуют две различные задачи укладки ранца: одна из них решается легко и характеризуется линейным ростом трудоемкости, а другая решается трудно. Легкий для укладки ранец можно трансформировать в трудный. Трудный для укладки ранец применяется в качестве открытого ключа, который легко использовать для зашифрования, но невозможно – для расшифрования. В качестве закрытого ключа применяется легкий для укладки ранец, который предоставляет простой способ расшифрования сообщения.

В программном коде представленного приложения механизм генерации требует предварительного создания открытого и закрытого ключа. Данный сегмент кода представлен в листинге 1.1.

private\_key = []  
sequence = []  
num = 4  
z = 8  
buff = num  
  
for i in range(z):  
 sequence.append(buff)  
 buff = random.randint(buff \* 2, buff \* 3)  
private\_key = sequence  
  
print("Private Key")  
for i in private\_key:  
 print(i, end=" ")  
  
print("\n")  
public\_key = []  
n = sum(private\_key) + 2  
a = generate\_coprime(n)  
sequence1 = []  
for i in private\_key:  
 sequence1.append((a \* i) % n)  
public\_key = sequence1  
  
print("Public Key")  
for i in public\_key:  
 print(i, end=" ")  
print("\n")

def generate\_coprime(n):  
 while True:  
 number = random.randint(1, n)  
 if nod(number, n) == 1:  
 return number

Листинг 1.1 – Генерация открытого и закрытого ключа

В процессе зашифрования, поводится разбиение исходного текста на блоки меньшего размера в соответствии с используемым, пример представлен в листинге 1.2

def encrypt(public\_key, text):  
 start\_time = time.perf\_counter()  
 encrypted\_list = []  
  
 for b in text:  
 binary\_string = format(int(b), '08b')  
 sum\_val = 0  
 for i in range(8):  
 if binary\_string[i] == '1':  
 sum\_val += public\_key[i]  
 encrypted\_list.append(sum\_val)  
  
 elapsed\_time = (time.perf\_counter() - start\_time) \* 1000  
 print(f"Encrypt:\t{elapsed\_time:.2f} ms")  
 return encrypted\_list

Листинг 1.2 – Функция зашифрования

В результате генерируется битовая последовательность, являющаяся зашифрованным сообщением, она поступает входным параметром в функцию расшифрования, код функции представлен на листинге 1.3.

def decrypt(private\_key, encrypted\_text, a, n):  
 start\_time = time.perf\_counter()  
 decrypted\_bytes = []  
 inverse = 1  
  
 while inverse \* a % n != 1:  
 inverse += 1  
  
 for item in encrypted\_text:  
 decrypted\_value = (item \* inverse) % n  
 binary\_string = ''  
 for i in range(len(private\_key) - 1, -1, -1):  
 if decrypted\_value >= private\_key[i]:  
 binary\_string = '1' + binary\_string  
 decrypted\_value -= private\_key[i]  
 else:  
 binary\_string = '0' + binary\_string  
  
 decrypted\_byte = int(binary\_string, 2).to\_bytes(1, byteorder='big')  
 decrypted\_bytes.append(decrypted\_byte)  
  
 elapsed\_time = (time.perf\_counter() - start\_time) \* 1000  
 print(f"Decrypt:\t{elapsed\_time:.2f} ms")  
 return b''.join(decrypted\_bytes)

Листинг 1.3 – Функция расшифрования

В результате вычислений мы наблюдаем решение задачи о рюкзаке и последующем извлечении элементов последовательности на основе двух независимых ключей, результат представлен на рисунке 1.1.

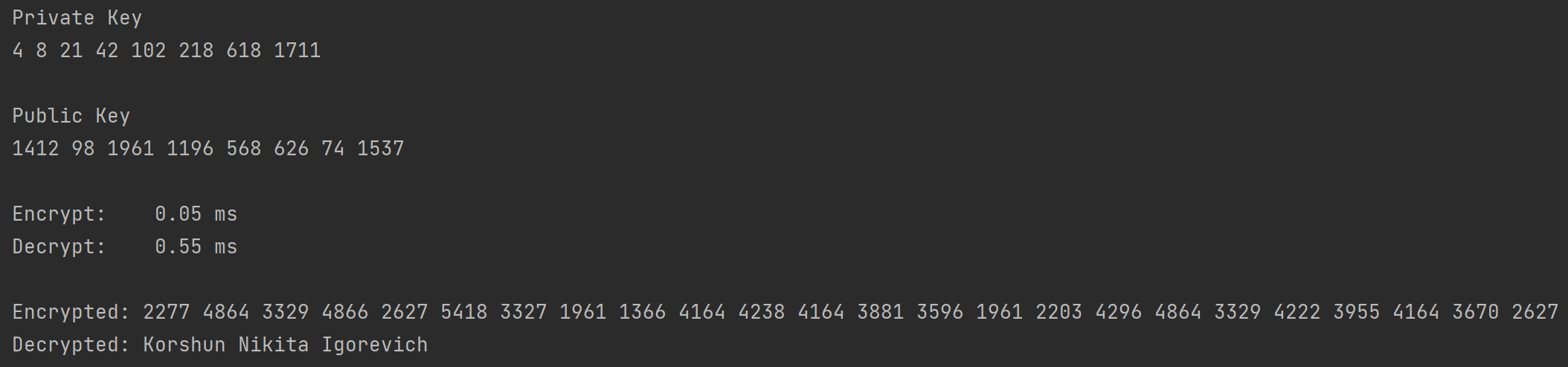


Рисунок 1.1 – Результат вывода программы

График изменения времени работы алгоритма в зависимости от длинны сообщения отображен на рисунке 1.2.

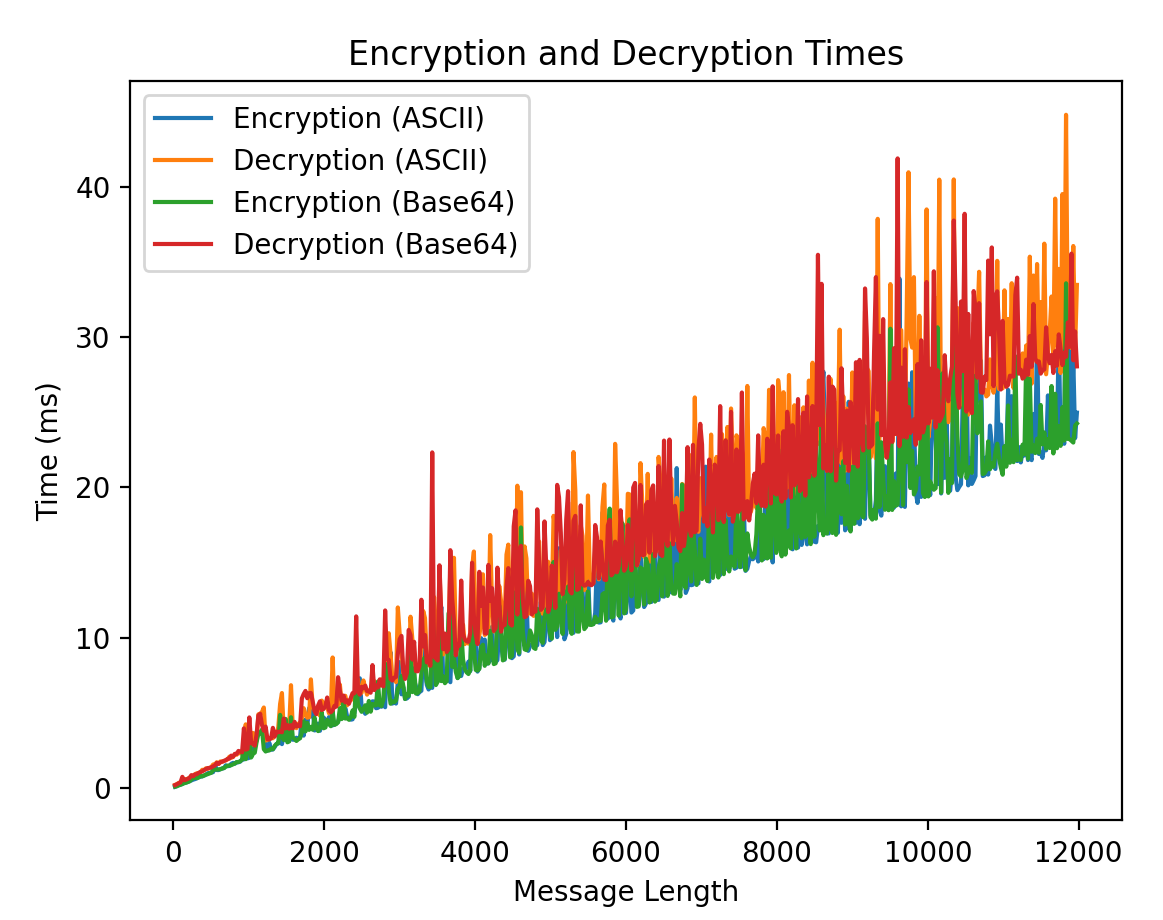


Рисунок 1.2 – График времени выполнения

**Вывод:** Важно отметить, что хотя ранцевые шифры обладают высокой степенью защищенности и низким уровнем ресурсозатратности, существует известная уязвимость в алгоритме, которая позволяет восстановить секретный ключ, зашифрованное сообщение и открытую последовательность.