МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение

высшего образования

«Вятский государственный университет»

Факультет автоматики и вычислительной техники

Кафедра ЭВМ

Отчёт

Лабораторная работа № 3 по дисциплине

«Защита информации»

Выполнил студент группы ИВТб-4301\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ / Птахова А. М./

Проверил доцента кафедры ЭВМ\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/ Караваева О. В./

Киров 2024

1. Цель

Изучить алгоритмы шифрования и програмно реализовать их

1. Ход работы

2.1 Алгоритм Вижинера

**Задание:**

Напишите дешифратор для <<ШИФРА ВИЖЕНЕРА>> и расшифруйте

предоставленную строку: < ~-, 3&( №83|#( №-=:3 ~=(-'?! ; ~;33^ .3;%( ,~')

**Листинг кода:**

def vigenere\_decrypt(ciphertext):

n = len(ciphertext)

best\_period = 1

best\_index = 0.0

for period in range(1, n//2 + 1):

groups = [''.join(ciphertext[i] for i in range(j, n, period)) for j in range(period)]

group\_index = sum(get\_coincidence\_index(group) for group in groups) / period

if abs(0.066 - group\_index) < abs(0.066 - best\_index):

best\_index = group\_index

best\_period = period

decrypted\_text = ''

key\_word=""

for i in range(best\_period):

group = ''.join(ciphertext[j] for j in range(i, n, best\_period))

#most\_common\_char = max(set(group), key = group.count)

print(group)

ind = get\_coincidence\_index(group)

print(ind)

for z in range(len(freq\_symb)):

#print(abs(freq\_symb[z]-ind))

if (abs(freq\_symb[z]-ind)<=0.006):

#print(z)

key\_word+=al[z]

break;

print(key\_word)

for i in range(0,len(ciphertext)):

str\_1 = dic[key\_word[i%best\_period]]

#print(str\_1)

ind1 = str\_1.index(ciphertext[i])

#print(ind1)

decrypted\_text += al[ind1]

#print(al[ind1])

return decrypted\_text

2.2 Алгоритм кузнечик

**Задание:**

Зашифруйте строку: <<Семья — это если по звуку угадываешь, кто именно моется в душе.>> Алгоритмом шифрования Кузнечик в режиме гаммирование c обратной связью.

Ключ для шифрования: 4501949630601736750961088499860450650604377412073095927180569493

**Листинг кода:**

def encrypt\_text(input\_entry, output\_entry, key\_entry):

input\_text = input\_entry.get("1.0", "end-1c")

#key = generate\_key(len(input\_text))

key\_entry.insert("1.0", key)

encrypted\_text = encrypt( input\_text, key)

output\_entry.delete("1.0", tk.END)

output\_entry.insert("1.0", encrypted\_text)

def decrypt\_text(input\_entry, output\_entry, key\_entry):

input\_text = input\_entry.get("1.0", "end-1c")

#key = key\_entry.get("1.0", "end-1c")

decrypted\_text = decrypt(input\_text, key)

output\_entry.delete("1.0", tk.END)

output\_entry.insert("1.0", decrypted\_text)

* 1. Алгоритм rsa

**Задание:** Сгенерируйте открытый и закрытый ключи в алгоритме шифрования RSA, выбрав простые числа p = 139 и q = 223.

Зашифруйте сообщение: Кто всё поймет, тот всё и простит.

**Листинг кода:**

def gcd(a, b):

while b != 0:

a, b = b, a % b

return a

def modinv(a, m):

m0, x0, x1 = m, 0, 1

while a > 1:

q = a // m

m, a = a % m, m

x0, x1 = x1 - q \* x0, x0

return x1 + m0 if x1 < 0 else x1

def generate\_keys(p, q):

n = p \* q

phi\_n = (p - 1) \* (q - 1)

# Выбираем открытую экспоненту e

e = 65537

# Находим закрытую экспоненту d

d = modinv(e, phi\_n)

return (e, n), (d, n)

def encrypt(message, public\_key):

e, n = public\_key

encrypted\_message = [pow(ord(char), e, n) for char in message]

return encrypted\_message

def decrypt(encrypted\_message, private\_key):

d, n = private\_key

decrypted\_message = ''.join([chr(pow(char, d, n)) for char in encrypted\_message])

return decrypted\_message

def encrypt\_sig(message, private\_key):

d, n = private\_key

encrypted\_message = [pow(byte, d, n) for byte in message]

return encrypted\_message

def decrypt\_sig(encrypted\_message, public\_key):

e, n = public\_key

decrypted\_message = [pow(byte, e, n) for byte in encrypted\_message]

return decrypted\_message

p = 139

q = 223

2.4 Функция хэширования

**Задание:** Реализовать алгоритм криптографической функции – SHA1.

Зашифруйте сообщение: И вот она сперва долго плакала, а потом стала злая.

**Листинг кода:**

def sha1(data):

# Инициализация переменных

h0 = 0x67452301

h1 = 0xEFCDAB89

h2 = 0x98BADCFE

h3 = 0x10325476

h4 = 0xC3D2E1F0

# Дополнение данных до кратного 512-битному блоку

data = bytearray(data, 'utf-8')

data\_len = len(data) \* 8

data += b'\x80'

data += b'\x00' \* ((56 - len(data) % 64) % 64)

data += struct.pack('>Q', data\_len)

# Функции для циклической логики SHA-1

def left\_rotate(n, b):

return ((n << b) | (n >> (32 - b))) & 0xFFFFFFFF

for i in range(0, len(data), 64):

chunk = data[i:i+64]

w = list(struct.unpack('>16I', chunk)) + [0] \* 64

for j in range(16, 80):

w[j] = left\_rotate(w[j-3] ^ w[j-8] ^ w[j-14] ^ w[j-16], 1)

a, b, c, d, e = h0, h1, h2, h3, h4

for j in range(80):

if j <= 19:

f = (b & c) | ((~b) & d)

k = 0x5A827999

elif j <= 39:

f = b ^ c ^ d

k = 0x6ED9EBA1

elif j <= 59:

f = (b & c) | (b & d) | (c & d)

k = 0x8F1BBCDC

else:

f = b ^ c ^ d

k = 0xCA62C1D6

temp = left\_rotate(a, 5) + f + e + k + w[j] & 0xFFFFFFFF

e = d

d = c

c = left\_rotate(b, 30)

b = a

a = temp

h0 = (h0 + a) & 0xFFFFFFFF

h1 = (h1 + b) & 0xFFFFFFFF

h2 = (h2 + c) & 0xFFFFFFFF

h3 = (h3 + d) & 0xFFFFFFFF

h4 = (h4 + e) & 0xFFFFFFFF

return '%08x%08x%08x%08x%08x' % (h0, h1, h2, h3, h4)

2.5 ЭЦП

**Задание:** Используя хеш-образ строки: << Злых людей нет на свете, есть только люди несчастливые.>>

И вычислите электронную цифровую подпись по схеме RSA.

**Листинг кода:**

def encrypt\_text(input\_entry, output\_entry1, output\_entry11, private\_entry):

input\_text = input\_entry.get("1.0", "end-1c")

byte\_message = input\_text.encode('utf-8')

hash\_value = fourth\_task.encrypt\_adler32(byte\_message)

hash\_bytes = split\_hash(hash\_value)

private\_key = tuple(map(int, private\_entry.get("1.0", "end-1c").split()))

signature = third\_task.encrypt\_sig(hash\_bytes, private\_key)

output\_entry1.delete("1.0", tk.END)

output\_entry1.insert("1.0", signature)

output\_entry11.delete("1.0", tk.END)

output\_entry11.insert("1.0", hash\_value)

def decrypt\_text(input\_entry, output\_entry2, output\_entry22, output\_entry222, public\_entry):

input\_text = input\_entry.get("1.0", "end-1c")

byte\_message = input\_text.encode('utf-8')

hash\_value = fourth\_task.encrypt\_adler32(byte\_message)

input\_numbers = output\_entry22.get("1.0", "end-1c")

numbers\_list = list(map(int, input\_numbers.split()))

public\_key = tuple(map(int, public\_entry.get("1.0", "end-1c").split()))

decode\_hash\_bytes = third\_task.decrypt\_sig(numbers\_list, public\_key)

decode\_hash = bytes\_to\_hash(decode\_hash\_bytes)

output\_entry2.delete("1.0", tk.END)

output\_entry2.insert("1.0", hash\_value)

output\_entry222.delete("1.0", tk.END)

output\_entry222.insert("1.0", decode\_hash)

3. Вывод

В ходе выполнения лабораторной работы были изучены и реализованы алгоритмы шифр Вижинера, кузнечика, rsa, SHA-1, ЭЦП.