Пшенко Артем ФИТ 3-4

Отчет по лабораторной работе № 8

Исследование ассиметричных шифров (алгоритм Merkle-Hellman)

Информационная безопасность

Вариант 9

Цель данной лабораторной работы состоит в изучении работы алгоритма ассиметричного шифрования Merkle-Hellman, его применении для шифрования и дешифрования сообщений.

Алгоритм работы Merkle-Hellman

1. Генерация сверхвозрастающей последовательности: создается сверхвозрастающая последовательность, где каждый следующий элемент больше суммы всех предыдущих.
2. Генерация открытого ключа: на основе сверхвозрастающей последовательности и двух случайных чисел (n и a) генерируется открытый ключ.

Шифрование сообщения:

1. Сообщение преобразуется в двоичный формат.
2. Каждая часть двоичного сообщения умножается на соответствующий элемент открытого ключа и суммируется.

Расшифрование сообщения:

1. Используя модульный обратный элемент числа a, зашифрованное сообщение преобразуется обратно.
2. Преобразованное сообщение разлагается на исходные элементы сверхвозрастающей последовательности для восстановления исходного двоичного сообщения.

Исходный код программы приведен в листинге 1. Результат работы программы приведен на рисунке 1.

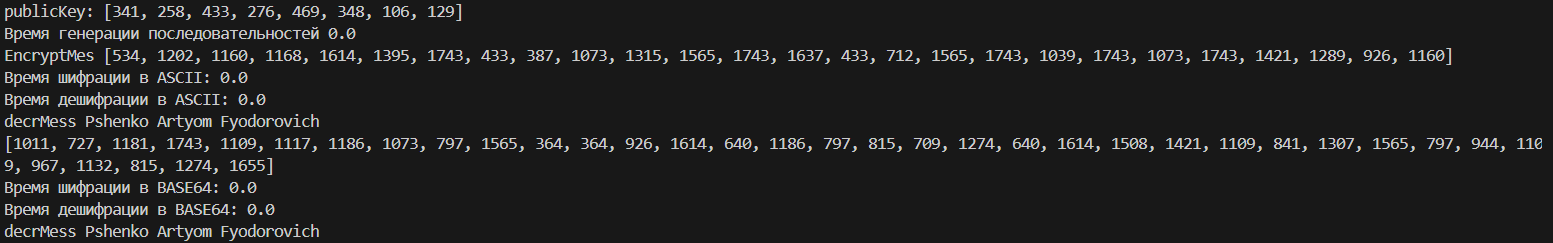


Рисунок 1. Результат работы программы

Таким образом, в ходе лабораторной работы была продемонстрирована работа алгоритма ассиметричного шифрования Merkle-Hellman.

Листинг 1. Исходный код алгоритма Merkle-Hellman

|  |
| --- |
| import random  import math  import base64  from time import time  # Function to generate superincreasing sequence  def generate\_superincreasing\_sequence(n, m):  max\_number = 2 \*\* m  sequence = []  sum = 0  next = 2  for i in range(n):  sequence.append(next)  sum += next + 1  next = sum  return sequence  # Function to generate a number more than the sum of sequence elements  def more\_than\_seq\_sum(seq):  return sum(seq) + 4  # Function to calculate gcd  def gcd(a, b):  while b != 0:  a, b = b, a % b  return a  # Function to generate a  def generate\_a(n):  while True:  a = random.randint(1, n - 1)  if gcd(a, n) == 1:  return a  # Function to generate open key  def generate\_open\_key(private\_key, n, a):  return [(elem \* a) % n for elem in private\_key]  # Function to convert message to ASCII to binary  def convert\_message\_to\_ascii\_to\_binary(characters):  return [format(ord(c), '08b') for c in characters]  # Function to convert message to Base64 to binary  def convert\_message\_to\_base64\_to\_binary(characters):  base64\_bytes = base64.b64encode(characters.encode('utf-8'))  base64\_message = base64\_bytes.decode('utf-8')  return [format(ord(c), '08b') for c in base64\_message]  # Function to encrypt message with private key  # умножаем каждый бит сообщения на соответствующий элемент открытого ключа и суммируя результаты  def encrypt(message, public\_key):  return [sum(int(bit) \* key for bit, key in zip(msg, public\_key)) for msg in message]  # Extended Euclidean Algorithm for finding modular inverse  def extended\_euclidean(a, b):  if a == 0:  return b, 0, 1  else:  g, x, y = extended\_euclidean(b % a, a)  return g, y - (b // a) \* x, x  # Function to find modular inverse  def mod\_inverse(a, n):  g, x, \_ = extended\_euclidean(a, n)  if g == 1:  return x % n  # Function to decrypt message with open key  def decrypt(encrypted\_message, private\_key, n, a):  k = mod\_inverse(a, n) # (a \* k) % n = 1  decrypted\_message = [(e \* k) % n for e in encrypted\_message]  result = []  # преобр числа в двоичн формат  for num in decrypted\_message:  binary = ''  for key in reversed(private\_key):  if num >= key:  num -= key  binary = '1' + binary  else:  binary = '0' + binary  result.append(binary)  return result  # Function to convert binary to ASCII to message  def convert\_binary\_to\_ascii\_to\_message(binary\_strings):  return ''.join(chr(int(binary, 2)) for binary in binary\_strings)  # Function to convert binary to Base64 to message  def convert\_binary\_to\_base64\_to\_message(binary\_strings):  base64\_message = ''.join(chr(int(binary, 2)) for binary in binary\_strings)  base64\_bytes = base64\_message.encode('utf-8')  message\_bytes = base64.b64decode(base64\_bytes)  return message\_bytes.decode('utf-8')  #Создание тайного ключа  start = time()  size = 8 # Размер последовательности  bits = 100 # Количество бит  private\_key = generate\_superincreasing\_sequence(size, bits)  #Генерация n и a  n = more\_than\_seq\_sum(private\_key)  a = generate\_a(n)  #Генерация открытого ключа (Нормальная последовательность чисел)  public\_key = generate\_open\_key(private\_key, n, a)  print('publicKey:', public\_key)  end = time()  print('Время генерации последовательностей', (end-start))  #Перевод сообщения в ASCII  start1 = time()  message = "Pshenko Artyom Fyodorovich"  conv\_mes = convert\_message\_to\_ascii\_to\_binary(message)  #Шифрование сообщения  start\_encr\_acsii = time()  encrypt\_mes = encrypt(conv\_mes, public\_key)  end\_encr\_acsii = time()  print("EncryptMes", encrypt\_mes)  print('Время шифрации в ASCII:', end\_encr\_acsii-start\_encr\_acsii)  #Расшифрование сообщения  start\_decr\_acsii = time()  decrypted\_message = decrypt(encrypt\_mes, private\_key, n, a)  end\_decr\_acsii = time()  print('Время дешифрации в ASCII:', end\_decr\_acsii-start\_decr\_acsii)  #Перевод из двоичной в char  decr\_mess = convert\_binary\_to\_ascii\_to\_message(decrypted\_message)  print("decrMess", decr\_mess)  #Перевод сообщения в Base64  start2 = time()  conv\_base64 = convert\_message\_to\_base64\_to\_binary(message)  #Шифрование сообщения  start\_encr\_base64 = time()  encrypt\_mes2 = encrypt(conv\_base64, public\_key)  end\_encr\_base64 = time()  print(encrypt\_mes2)  print('Время шифрации в BASE64:', end\_encr\_base64-start\_encr\_base64)  #Расшифрование сообщения  start\_decr\_base64 = time()  decrypted\_message2 = decrypt(encrypt\_mes2, private\_key, n, a)  end\_decr\_base64 = time()  print('Время дешифрации в BASE64:', end\_decr\_base64-start\_decr\_base64)  #Перевод из 2 в char  decr\_mess2 = convert\_binary\_to\_base64\_to\_message(decrypted\_message2)  print("decrMess", decr\_mess2) |

Результат работы программы приведен на рисунке 1