Пшенко Артем ФИТ 3-4

Отчет по лабораторной работе № 9

Исследование потоковых шифров (алгоритмы BBS и RC4)

Информационная безопасность

Вариант 9

Цель данной лабораторной работы состоит в изучении работы алгоритмов поточного шифрования RC4 и BBS, их применении для шифрования и дешифрования сообщений. Необходимо продемонстрировать, как каждый из этих алгоритмов генерирует псевдослучайные последовательности и использовать их для шифрования и дешифрования сообщений.

Алгоритм RC4:

1. KSA (Key Scheduling Algorithm): создается массив S размером 256 байт и инициализируется значениями от 0 до 255. Затем, на основе ключа, происходит перемешивание элементов массива S.
2. PRGA (Pseudo-Random Generation Algorithm): используя массив S, генерируется псевдослучайная последовательность байт.
3. Шифрование/Дешифрование: путем применения операции XOR между байтами открытого текста и сгенерированным псевдослучайным потоком осуществляется шифрование и дешифрование.

Исходный код алгоритма RC4 приведен в листинге 1.

Листинг 1. Исходный код алгоритма RC4

|  |
| --- |
| import time  def KSA(key):  key\_length = len(key)  S = list(range(256))  j = 0  for i in range(256):  j = (j + S[i] + key[i % key\_length]) % 256  S[i], S[j] = S[j], S[i]  return S  def PRGA(S):  i = 0  j = 0  while True:  i = (i + 1) % 256  j = (j + S[i]) % 256  S[i], S[j] = S[j], S[i]  t = (S[i] + S[j]) % 256  K = S[t]  yield K  def RC4(key):  S = KSA(key)  return PRGA(S)  def generate\_keystream(key, n):  keystream = []  rc4 = RC4(key)  for \_ in range(n):  keystream.append(next(rc4))  return keystream  def evaluate\_speed(key, n, iterations=100):  start\_time = time.time()  for \_ in range(iterations):  generate\_keystream(key, n)  end\_time = time.time()  total\_time = end\_time - start\_time  avg\_time\_per\_iteration = total\_time / iterations  print("Total time for", iterations, "iterations:", total\_time, "seconds")  print("Average time per iteration:", avg\_time\_per\_iteration, "seconds")  key = [61, 60, 23, 22, 21, 20]  n = 8  keystream = generate\_keystream(key, n)  print("Generated keystream:", keystream)  evaluate\_speed(key, n)  # def encrypt\_decrypt(text, key):  # keystream = generate\_keystream(key, len(text))  # encrypted\_text = [a ^ b for a, b in zip(text, keystream)]  # decrypted\_text = [a ^ b for a, b in zip(encrypted\_text, keystream)]  # return encrypted\_text, decrypted\_text  # key = [61, 60, 23, 22, 21, 20]  # plaintext = [72, 69, 76, 76, 79] # "HELLO"  # encrypted\_text, decrypted\_text = encrypt\_decrypt(plaintext, key)  # print("Original Text:", plaintext)  # print("Encrypted Text:", encrypted\_text)  # print("Decrypted Text:", decrypted\_text) |

Результат работы алгоритма RC4 приведен на рисунке 1.

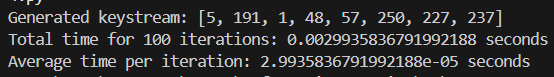


Рисунок 1. Результат работы программы

Алгоритм BBS:

1. Генерация простых чисел: генерируются два больших простых числа p и q, такие что p = 3 (mod4) и q = 3 (mod4).
2. Инициализация: вычисляется N=p\*q и случайное начальное значение x0, которое является взаимно простым с N.
3. Генерация последовательности: на каждом шаге значение x возводится в квадрат и берется по модулю N, при этом генерируется один бит последовательности в зависимости от четности результата.

Листинг 2. Исходный код алгоритма BBS

|  |
| --- |
| from sympy import \*  from random import randint  from math import gcd  def generate\_BBS\_sequence(p, q, x\_0, length):  N = p \* q  x = x\_0  result = []  for \_ in range(length):  x = pow(x, 2, N)  print('x = ', x)  result.append(x % 2)  print('result = ', result, '\n')  return result  p = randprime(10\*\*6, 10\*\*7)  q = randprime(10\*\*6, 10\*\*7)    while p % 4 != 3 and q % 4 != 3:  p = randprime(10\*\*6, 10\*\*7)  q = randprime(10\*\*6, 10\*\*7)    N = p \* q  x\_0 = randint(2, N - 1)  while gcd(x\_0, N) != 1:  x\_0 = randint(2, N - 1)    length = 10    bbs\_sequence = generate\_BBS\_sequence(p, q, x\_0, length)  print("BBS Sequence:", bbs\_sequence) |

Результат работы алгоритма BBS приведен на рисунке 2.

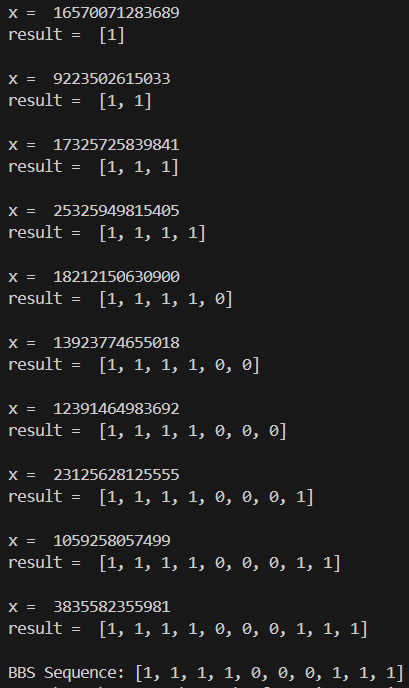


Рисунок 2. Результат работы программы

Таким образом, в ходе лабораторной работы были продемонстрированы работы алгоритмов поточного шифрования RC4 и BBS.