Учреждение образования

«Белорусский государственный технологический университет»

**Кафедра информационных систем и технологий**

**«Отчёт по лабораторной работе №8»**

«Исследование потоковых шифров»

**Выполнил:** студент 3 курса

4 группы специальности ПОИТ

Супрунюк Евгений Андреевич

**Проверил:** преподаватель

Сазонова Дарья Владимировна

Минск 2023

1. **Описание приложения**

Приложение написано на языке программирования Python и позволяет провести шифрование поточным шифром.

1. **Методика выполнения расчетов**

В данной лабораторной работе была поставлена цель создания приложения, которое реализовывает алгоритм BBS ПСП и потоковый шифр RC4. На листинге 2.1 представлены класс с функциями, реализующий алгоритм BBS ПСП, а на листинге 2.2 потоквый шифр RC4.

|  |
| --- |
| class BBS:      """      p is prime number      q is a prime number      seed is the x0      mfactor is compute by p \* q      """      def \_\_init\_\_(self, p, q, seed):          self.p = p          self.q = q          self.seed = seed          self.xn = seed          self.m\_factor = p \* q      """      generation of bits string, n is the lenght of the number to generate in bits      """      def get\_random(self, n):          out = ''          for i in range(n):              self.xn = pow(self.xn, 2, self.m\_factor)              out += str(self.xn % 2)          return out      """      The same as above but return the value as an integer      """      def get\_random\_int(self, n):          return int(self.get\_random(n), 2) |

Листинг 2.1 –код программы, реализующий алгоритм BBS ПСП

|  |
| --- |
| import random as rd  import sys  import time  # BASIC FUNCTIONS used for conversions from one notation to another  def binary(a):      '''      a should be a positive integer smaller than 256      returns the 8-bit binary writing of a      '''      assert a < 256      res = bin(a)[2:]      while len(res) != 8:          res = "0"+res      return res  def toInts(text):      '''      @input: text: type(text) = string      @output: list of the ASCII number of each character of text      '''      return [ord(letter) for letter in text]  def toBytes(intList):      '''      @intput: intList: type(intList) = list of integers      @output: string with the different integers written in hexadecimal notation      '''      res = "0x"      for integer in intList:          hexa = hex(integer)[2:]          if len(hexa) == 1:              hexa = '0'+hexa          res = res + hexa      return res  def toChar(intList):      '''      @intput: intList: type(intList) = list of integers      @output: list of characters corresponding to the integers it got as inputs (using ASCII)      '''      return [chr(integer) for integer in intList]  def toStr(charList):      '''      charList should be a list of characters      concatenates all the characters from the list and returns the final string      '''      res = ""      for letter in charList:          res = res+letter      return res  # ENCRYPT/DECRYPT FUNCTIONS  def xor(a, b):      '''      @signature: int xor(int a, int b) - all in base 10      returns the integer corresponding to the binary XOR operation between a and b      '''      a = binary(a)      b = binary(b)      assert (len(a) == 8 and len(b) == 8)      output = ""      for i in range(len(a)):          # print(i, a, b)          inter = int(a[i])+int(b[i])          if inter == 2:              inter = 0          output = output+str(inter)      return int(output, 2)  def schedule(key):      '''      KSA: Key Schedule Algorithm      key should be a list of positive integers (<256)      creates S as the list of all possible bytes (base 10 integers from 0 to 255)      shuffles S with the help of the key, using the main method from the Pseudo-Random Generation Algo (PRGA)      '''      S = [i for i in range(256)]      j = 0      for i in range(256):          j = (j + S[i] + key[i % len(key)]) % 256          S[i], S[j] = S[j], S[i]      return S  def cipher(inputText, S):      '''      encrypts plainText with S, using the RC4 algorithm      '''      inputInts = toInts(inputText)      i = 0      j = 0      outputInts = []      for elem in inputInts:          i = (i+1) % 256          j = (j+S[i]) % 256          S[i], S[j] = S[j], S[i]          t = (S[i]+S[j]) % 256          outputInts.append(xor(elem, S[t]))      return toChar(outputInts)  def genKey(l=8):      '''      generates a random key of length l bytes (8 by default)      '''      return [rd.randint(0, 255) for i in range(l)]  # TEST SCRIPT  if \_\_name\_\_ == "\_\_main\_\_":      if len(sys.argv) > 3:          print('[ERROR] too many arguments! If you want to encrypt a message with spaces, you will need to put quotes around it.')          print('Example: "Hello World!" "MySecretKey"')          exit()      # get plainText      try:          plainText = sys.argv[1]      except:          print("[ERROR] missing argument! Please add a message (and your key if you have one) to encrypt.\nExample: Hello! secretKey")          exit()      # get secret key      try:          myKey = toInts(sys.argv[2])          print("[INFO] using your key: {}".format(sys.argv[2]))      except:          myKey = genKey()  # or use myKey=toInts("YourOwnSecretString")          print("[INFO] using a randomly-generated key")      print('Plain text: {}'.format(plainText))      print('Secret Key: {}'.format(toBytes(myKey)))      crypt\_time = time.time()      cipherText = cipher(plainText, schedule(myKey))      crypt\_time = time.time() - crypt\_time      print(f'Cryption time: {crypt\_time}')      print('Cipher bytes: {}'.format(toBytes(toInts(cipherText))))      decrypt\_time = time.time()      print('Result after decrypting: {}'.format(          toStr(cipher(cipherText, schedule(myKey)))))      decrypt\_time = time.time() - decrypt\_time      print(f'Decryption time: {decrypt\_time}') |

Листинг 2.2 –код программы, реализующий потоковый шифр RC4

**3. Результаты работы приложения**

Для выполнения расчетов достаточно необходимо запустить приложение и ввести необходимые параметры. Рисунок 3.1 и 3.2 показывают требуемые в данной лабораторной работе результаты.

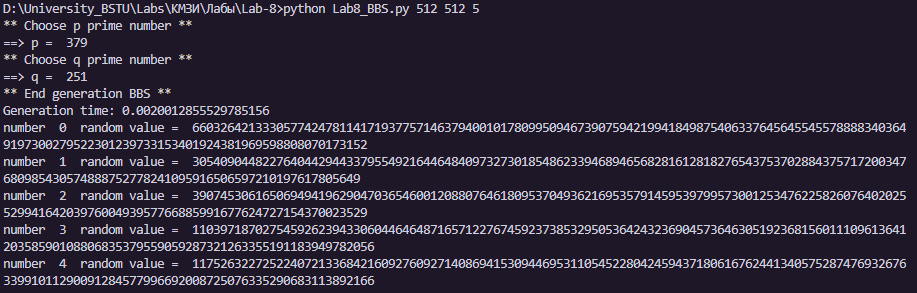


Рисунок 3.1 – Результат работы по алгоритму BBS ПСП

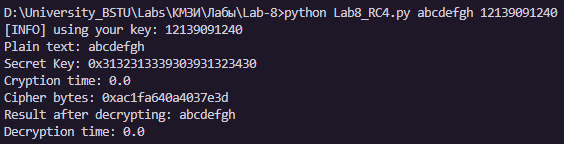


Рисунок 3.2 – Результат работы по потоковому шифру RC4

**Вывод**

В ходе лабораторной работы было разработано приложение для генерации ПСП по алгоритму BBS и приложение для потокового шифра RC4.