Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение

высшего образования

«Ульяновский государственный технический университет»

Кафедра «Вычислительная техника»

Дисциплина «Высокопроизводительные вычисления»

**Лабораторная работа № 5**

**Вариант 10**

**Блочные шифры, режимы использования блочных шифров**

Выполнил:

Студент группы ИВТАСбд-42

Сулейманов М.З.

Проверил:

Мартынов А.И.

Ульяновск

2024

**Цель работы**

Научиться реализовывать методы ассиметричной криптографии: генерацию пары ключей (открытый ключ, закрытый ключ), шифрование сообщения с использованием открытого ключа, дешифрование с использованием закрытого ключа.

**Задание**

Реализовать приложение с графическим интерфейсом, позволяющее выполнять следующие действия:

1. Генерировать простые числа от заданной границы вправо

2. Создавать пару: {Открытый ключ, Закрытый ключ}

3. Шифровать текстовые и двоичные файлы Открытым ключом с помощью ассиметричного алгоритма, указанного в варианте задания

4. Дешифровать текстовые и двоичные файлы Закрытым ключом с помощью ассиметричного алгоритма, указанного в варианте задания

5. Сохранять зашифрованные/дешифрованные данные в файл

6. Загружать зашифрованные/дешифрованные данные из файла

**Теоретическая часть**

**Криптосистема Эль-Гамаля**

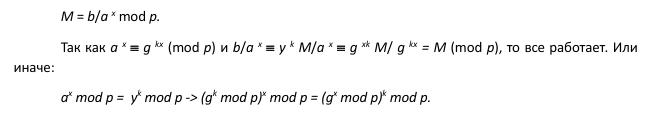
Данная система является альтернативой алгоритму RSA и при равном значении ключа обеспечивает ту же криптостойкость. Метод Эль-Гамаля основан на проблеме дискретного логарифма. Если возводить число в степень в конечном поле достаточно легко, то восстановить аргумент по значению (то есть найти логарифм) довольно трудно. Для генерации пары ключей сначала выбирается простое число p и два случайных числа, g и x, оба меньше p. Затем вычисляется



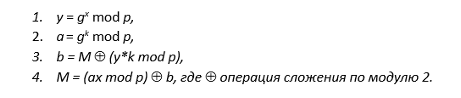
Открытым ключом являются y, g и p. И g, и p можно сделать общими для группы пользователей. Закрытым ключом является x. Для шифрования сообщения M сначала выбирается случайное число k, взаимно простое с p - 1. Затем вычисляются



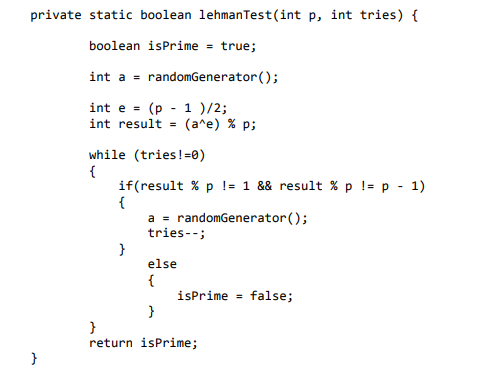
Пара (a,b) является шифротекстом. Получаемый шифротекст в два раза длиннее открытого текста. Для дешифрирования (a,b) вычисляется



Каждая подпись или шифрование EIGamal требует нового значения k, и это значение должно быть выбрано случайным образом. Если когда-нибудь Злоумышленник раскроет k, он сможет раскрыть закрытый ключ x. Если Злоумышленник когда-нибудь сможет получить два сообщения, подписанные или зашифрованные с помощью одного и того же k, то он сможет раскрыть x, даже не зная значение k. Алгоритм генерации ключей и шифрования/дешифрования Открытый ключ: 1. Выбрать простое число p (может быть общим для группы пользователей) 2. Выбрать g

< p. Шифрование: 1. k выбирается случайным образом, взаимно простое с p–1 2. a (шифротекст) =g k mod p, 3. b (шифротекст)= y k M mod p. Дешифрирование: 

**Алгоритм Рабина-Миллера**

1. Выберите случайное число a, меньшее p 2. Вычислите a (p-1)/2 mod p 3. Если a (p-1)/2 mod p ≠1 или -1, то p не является простым 4. Если a (p-1)/2 mod p = 1 или -1, то вероятность того, что число p не является простым, не больше 50 процентов. И снова, вероятность того, что случайное число a будет свидетелем составной природы числа p, не меньше 50 процентов. Повторите эту проверку t раз. Если результат вычислений равен 1 или -1, но не всегда равен 1, то p является простым числом с вероятностью ошибки (1/2)t . 

**Листинг**

|  |
| --- |
| import tkinter as tk  from tkinter import filedialog, messagebox  import random  # Алгоритм Лемана  def lehmanTest(p, tries=1000):      if p < 2:          return False      if p in (2, 3):          return True      if p % 2 == 0:          return False      isPrime = True      for \_ in range(tries):          a = random.randint(2, p - 1) # Изменил диапазон на 2,p-1          e = (p - 1) // 2 # Используем целочисленное деление //          result = pow(a, e, p)  # Модульная арифметика!          if result % p != 1 and result % p != p - 1:              isPrime = False # сразу меняем значение isPrime, если тест не прошел              break  # Выходим из цикла, тест провален      return isPrime    def generate\_prime(start):      while True:          if lehmanTest(start):              return start          start += 1          print(str(start))  def generate\_keys():      start = 2 \*\* 127      p = generate\_prime(start)      g = random.randint(2, p - 2)      x = random.randint(1, p - 2)  # Закрытый ключ      y = pow(g, x, p)  # Открытый ключ      return (p, g, y), x  # (открытый ключ, закрытый ключ)  def encrypt(message, public\_key):      p, g, y = public\_key      k = random.randint(1, p - 2)      a = pow(g, k, p)      b = (pow(y, k, p) \* message) % p      return a, b  def decrypt(encrypted\_message, private\_key, public\_key):      a, b = encrypted\_message      p, \_, \_ = public\_key      s = pow(a, private\_key, p)      message = (b \* pow(s, p - 2, p)) % p      return message  # --- GUI приложение ---  class ElGamalApp:      def \_\_init\_\_(self, root):          self.root = root          self.root.title("Эль-Гамаль Шифрование")          # Переменные          self.public\_key = None          self.private\_key = None          self.encrypted\_data = None          self.decrypted\_data = None          # Элементы интерфейса          self.label1 = tk.Label(root, text="Введите стартовое число для генерации простого числа:")          self.label1.pack()          self.entry\_start = tk.Entry(root)          self.entry\_start.pack()          self.button\_generate\_prime = tk.Button(root, text="Сгенерировать простое число",                                                 command=self.generate\_prime\_number)          self.button\_generate\_prime.pack()          self.label\_prime = tk.Label(root, text="Простое число:")          self.label\_prime.pack()          self.button\_generate\_keys = tk.Button(root, text="Сгенерировать ключи", command=self.generate\_keys)          self.button\_generate\_keys.pack()          self.label\_keys = tk.Label(root, text="Открытый и закрытый ключи:")          self.label\_keys.pack()          self.button\_encrypt = tk.Button(root, text="Зашифровать файл", command=self.encrypt\_file)          self.button\_encrypt.pack()          self.button\_decrypt = tk.Button(root, text="Дешифровать файл", command=self.decrypt\_file)          self.button\_decrypt.pack()      def generate\_prime\_number(self):          try:              start = int(self.entry\_start.get())              prime = generate\_prime(start)              self.label\_prime.config(text=f"Простое число: {prime}")          except ValueError:              messagebox.showerror("Ошибка", "Введите корректное стартовое число.")      def generate\_keys(self):          self.public\_key, self.private\_key = generate\_keys()          self.label\_keys.config(text=f"Открытый ключ: {self.public\_key}\nЗакрытый ключ: {self.private\_key}")      def encrypt\_file(self):          if not self.public\_key:              messagebox.showerror("Ошибка", "Сначала сгенерируйте ключи!")              return          file\_path = filedialog.askopenfilename()          if file\_path:              with open(file\_path, "rb") as file:                  data = file.read()              encrypted\_data = [encrypt(byte, self.public\_key) for byte in data]              save\_path = filedialog.asksaveasfilename(defaultextension=".txt")              with open(save\_path, "wb") as file:                  for a, b in encrypted\_data:                      # Запись длины a и b и их значений                      a\_bytes = a.to\_bytes((a.bit\_length() + 7) // 8, 'big')                      b\_bytes = b.to\_bytes((b.bit\_length() + 7) // 8, 'big')                      file.write(len(a\_bytes).to\_bytes(4, 'big') + a\_bytes)                      file.write(len(b\_bytes).to\_bytes(4, 'big') + b\_bytes)              messagebox.showinfo("Успех", "Файл успешно зашифрован")      def decrypt\_file(self):          if not self.private\_key or not self.public\_key:              messagebox.showerror("Ошибка", "Сначала сгенерируйте ключи!")              return          file\_path = filedialog.askopenfilename()          if file\_path:              with open(file\_path, "rb") as file:                  encrypted\_data = []                  while True:                      len\_a\_bytes = file.read(4)                      if not len\_a\_bytes:                          break                      len\_a = int.from\_bytes(len\_a\_bytes, 'big')                      a = int.from\_bytes(file.read(len\_a), 'big')                      len\_b\_bytes = file.read(4)                      len\_b = int.from\_bytes(len\_b\_bytes, 'big')                      b = int.from\_bytes(file.read(len\_b), 'big')                      encrypted\_data.append((a, b))              decrypted\_data = bytes([decrypt(pair, self.private\_key, self.public\_key) for pair in encrypted\_data])              save\_path = filedialog.asksaveasfilename(defaultextension=".txt")              with open(save\_path, "wb") as file:                  file.write(decrypted\_data)              messagebox.showinfo("Успех", "Файл успешно расшифрован")  root = tk.Tk()  app = ElGamalApp(root)  root.mainloop() |