Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования

«Пермский национальный исследовательский политехнический университет»

Электротехнический факультет

Кафедра «Информационные технологии и автоматизированные системы»

Направление 09.03.01 – «Информатика и вычислительная техника»

Дисциплина: «Защита информации»

Профиль: «Программная инженерия»

Семестр 5

ОТЧЕТ

по лабораторной работе №3

Тема: «Шифрование методом Диффи-Хеллмана»

Вариант 7

Выполнил: студент группы РИС-23-3б

Гордеев В. А. \_\_\_\_\_\_\_\_\_\_\_

Проверил: доцент кафедры ИТАС

Шереметьев В. Г. \_\_\_\_\_\_\_\_\_\_\_

Дата \_\_\_\_\_\_

Пермь, 2025

**ЦЕЛЬ РАБОТЫ**

Получить практические навыки по использованию ассиметричных алгоритмов шифрования, на примере использования алгоритма RSA.

**ЗАДАНИЕ**

Выполнить шифрование текстового файла длиной не меньшей 256 символов методом Диффи-Хеллмана, используя в качестве x и y простые числа с разрядностью не меньшей двенадцати, используя в алгоритме шифрования функцию возведения в степень

**ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ**

Протокол Диффи-Хеллмана — это криптографический метод, который позволяет двум сторонам, не имеющим предварительных договоренностей, создать общий секретный ключ через незащищенный канал связи. Разработанный в 1976 году Уитфилдом Диффи и Мартином Хеллманом, этот алгоритм стал революционным решением проблемы распределения ключей в криптографии. Полученный общий секрет в дальнейшем может быть использован для шифрования сообщений с помощью алгоритмов симметричного шифрования.

Принцип работы и математическая основа

В основе протокола лежит сложность вычисления дискретных логарифмов. Проще говоря, существуют математические операции, которые легко выполнить в одном направлении, но чрезвычайно сложно обратить. Именно на этом принципе и строится безопасность алгоритма.

Алгоритм обмена ключами включает в себя следующие шаги:

1. Выбор общих параметров: Две стороны, условно назовем их Алиса и Боб, договариваются об использовании двух общедоступных чисел: большого простого числа *p* и целого числа *g*, которое является первообразным корнем по модулю *p*. Эти числа могут передаваться по открытому каналу, так как их знание не comprometeт безопасность протокола.
2. Генерация секретных ключей: Каждая сторона генерирует свое собственное секретное число (закрытый ключ), которое держится в тайне. Алиса выбирает секретное число *a*, а Боб — *b*.
3. Вычисление открытых ключей:
   * Алиса вычисляет свой открытый ключ *A* по формуле: A = g^a mod p.
   * Боб вычисляет свой открытый ключ *B* по формуле: B = g^b mod p.
4. Обмен открытыми ключами: Алиса отправляет свой открытый ключ *A* Бобу, а Боб отправляет свой открытый ключ *B* Алисе. Этими ключами они обмениваются по незащищенному каналу.
5. Вычисление общего секретного ключа:
   * Алиса, получив открытый ключ Боба *B*, вычисляет общий секретный ключ *K* по формуле: K = B^a mod p.
   * Боб, получив открытый ключ Алисы *A*, вычисляет общий секретный ключ *K* по формуле: K = A^b mod p.

В результате этих вычислений и Алиса, и Боб получают одно и то же значение *K*, которое и является их общим секретным ключом. Это происходит потому, что:  
K = (g^b mod p)^a mod p = g^(b*a) mod p  
K = (g^a mod p)^b mod p = g^(a*b) mod p

Таким образом, даже если злоумышленник перехватит общедоступные параметры *p* и *g*, а также открытые ключи *A* и *B*, он не сможет вычислить общий секретный ключ *K*, не зная секретных чисел *a* или *b*. Для этого ему пришлось бы решить задачу дискретного логарифмирования, что для больших чисел является вычислительно сложной задачей.

Уязвимости и применение

Несмотря на свою эффективность, "чистый" алгоритм Диффи-Хеллмана уязвим для атаки «человек посередине» (Man-in-the-Middle, MitM).[[1](https://www.google.com/url?sa=E&q=https%3A%2F%2Fvertexaisearch.cloud.google.com%2Fgrounding-api-redirect%2FAUZIYQGOGi1_-8cMu5KpmHv9v3b3jvJBr3o2MSO_-KSpV4IJCjc_tnWvBO-xuyeB-PdrkBjiARlRMTu2TZBfHFytjlGwYxwZz0xLh08O3J8AQOib_XZiLgoWlgehIqj_wdpsvIC694fki1nSh2gP0JCEgAUPvPpIf2bkM7nhiLAwsB2j8UauMLbNihHiP6zO4ydMC-V2_RJ6TP47QSkI7PYr0yK23cvMeA9GZUMhT4rHz9LvB2TCrjbj2h4b635cOhS5cmyA5CJfn-9tXSDGIE6nhZA2_JTEXFwNF18Vs2MgnH89feaSGX8%3D)] При такой атаке злоумышленник может вклиниться в канал связи, представившись каждой из сторон другой. Чтобы предотвратить подобные атаки, протокол Диффи-Хеллмана часто используется в сочетании с методами аутентификации, такими как цифровые подписи.

Протокол Диффи-Хеллмана широко применяется во многих криптографических стандартах и системах, включая:

* TLS/SSL: Протоколы, обеспечивающие безопасное соединение между клиентом и сервером в интернете.
* SSH: Протокол для безопасного удаленного управления компьютерами.
* IPSec: Набор протоколов для защиты данных, передаваемых по IP-сетям, например, в VPN-соединениях.

Таким образом, протокол Диффи-Хеллмана остается фундаментальным инструментом в современной криптографии, позволяя создавать безопасные каналы связи в недоверенной среде.

**ХОД РАБОТЫ**

Алгоритм обмена ключами Диффи-Хеллмана позволяет двум сторонам (назовем их Алиса и Боб) получить общий секретный ключ, обмениваясь информацией по незащищенному каналу. Этот ключ затем можно использовать для симметричного шифрования данных.

ВАЖНО: Сам по себе алгоритм Диффи-Хеллмана не шифрует данные, а только помогает безопасно выработать общий секрет. В этом приложении полученный секрет используется как ключ для надежного AES шифрования.

Процесс выработки общего ключа:

1. Публичные параметры:

   - Алиса и Боб договариваются о двух числах, которые не являются секретными:

     - p: очень большое простое число (модуль).

     - g: целое число, первообразный корень по модулю p (основание).

2. Генерация секретных ключей:

   - Алиса генерирует свой секретный ключ — случайное простое число 'x'. Она держит его в тайне.

   - Боб генерирует свой секретный ключ — случайное простое число 'y'. Он также держит его в тайне.

   (В этом приложении оба ключа 'x' и 'y' генерируются локально для демонстрации).

3. Вычисление и обмен открытыми ключами:

   - Алиса вычисляет свой открытый ключ 'A' по формуле: A = g^x mod p

   - Боб вычисляет свой открытый ключ 'B' по формуле: B = g^y mod p

   - Алиса и Боб обмениваются своими открытыми ключами (A и B) по сети. Злоумышленник может их перехватить.

4. Вычисление общего секрета:

   - Алиса, получив от Боба ключ B, вычисляет общий секрет 'S' по формуле: S = B^x mod p

   - Боб, получив от Алисы ключ A, вычисляет общий секрет 'S' по формуле: S = A^y mod p

Магия в том, что оба результата будут одинаковыми:

S = (g^y mod p)^x mod p = (g^x mod p)^y mod p = g^(xy) mod p

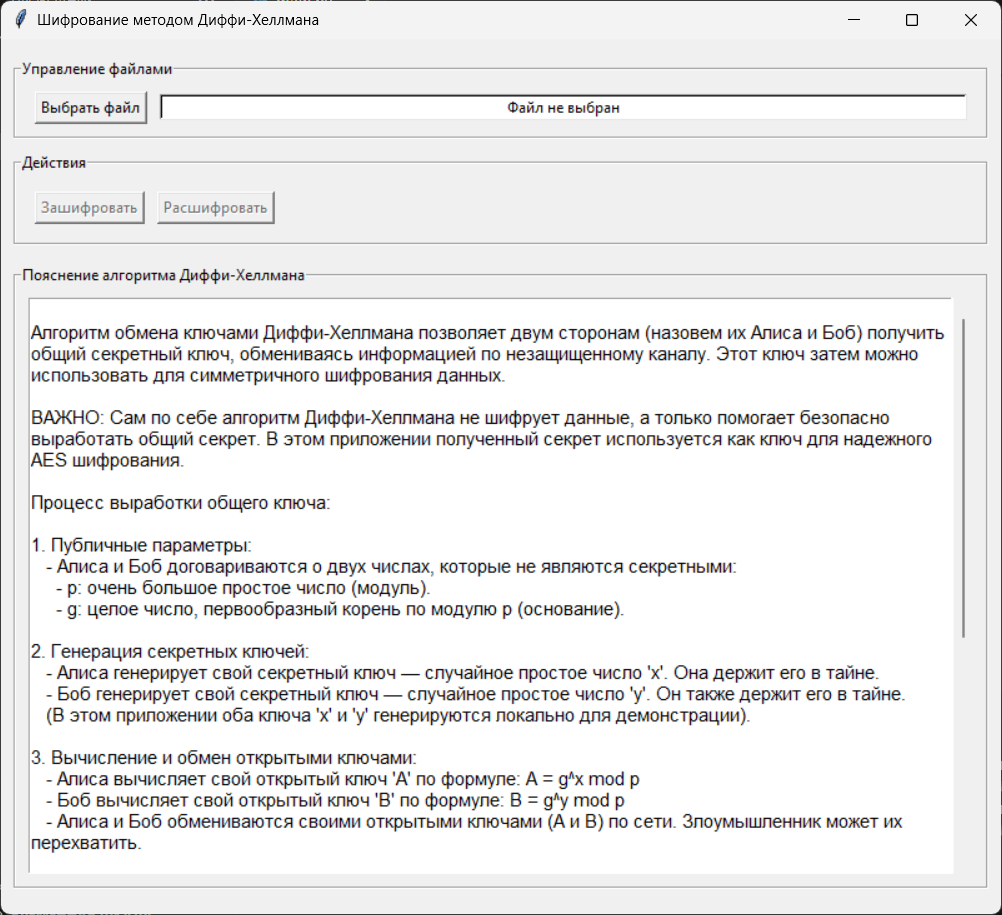
Теперь у Алисы и Боба есть идентичный секретный ключ 'S', который не передавался по сети в открытом виде. Злоумышленник, зная g, p, A и B, не может легко вычислить 'S', так как для этого ему нужно решить задачу дискретного логарифмирования (найти 'x' из A=g^x mod p), что для больших чисел является очень сложной задачей.

5. Шифрование файла:

   - Полученный общий секрет 'S' преобразуется в ключ фиксированной длины (например, с помощью хэш-функции SHA-256).

   - Этим ключом с помощью симметричного алгоритма (в данном случае AES) шифруется содержимое файла.

На Рисунке 1 представлен интерфейс разработанного приложения



*Рисунок 1. Интерфейс приложения*

**ПРИЛОЖЕНИЕ А**

**Листинг программы**

import tkinter as tk

from tkinter import filedialog, messagebox, scrolledtext

import os

import random

import base64

import hashlib

from cryptography.fernet import Fernet, InvalidToken

# --- Функции для генерации простых чисел ---

def is\_prime(n, k=5):

"""

Тест Миллера-Рабина на простоту числа.

n: число для проверки

k: количество раундов для повышения точности

"""

if n < 2:

return False

if n == 2 or n == 3:

return True

if n % 2 == 0:

return False

d, s = n - 1, 0

while d % 2 == 0:

d //= 2

s += 1

for \_ in range(k):

a = random.randint(2, n - 2)

x = pow(a, d, n)

if x == 1 or x == n - 1:

continue

for \_ in range(s - 1):

x = pow(x, 2, n)

if x == n - 1:

break

else:

return False

return True

def generate\_prime(bits):

"""Генерирует простое число с заданной разрядностью."""

while True:

p = random.getrandbits(bits)

# Убедимся, что число имеет нужную длину и нечетное

p |= (1 << bits - 1) | 1

if is\_prime(p):

return p

# --- Основной класс приложения ---

class CryptoApp:

def \_\_init\_\_(self, master):

self.master = master

self.master.title("Шифрование методом Диффи-Хеллмана (Демо)")

self.master.geometry("800x700")

self.filepath = None

# --- GUI элементы ---

main\_frame = tk.Frame(master, padx=10, pady=10)

main\_frame.pack(fill=tk.BOTH, expand=True)

# Фрейм для управления файлами

file\_frame = tk.LabelFrame(main\_frame, text="Управление файлами", padx=10, pady=10)

file\_frame.pack(fill=tk.X, pady=5)

self.btn\_select = tk.Button(file\_frame, text="Выбрать файл", command=self.select\_file)

self.btn\_select.pack(side=tk.LEFT, padx=5)

self.lbl\_filepath = tk.Label(file\_frame, text="Файл не выбран", bg="white", relief="sunken", width=60)

self.lbl\_filepath.pack(side=tk.LEFT, fill=tk.X, expand=True, padx=5)

# Фрейм для действий

actions\_frame = tk.LabelFrame(main\_frame, text="Действия", padx=10, pady=10)

actions\_frame.pack(fill=tk.X, pady=5)

self.btn\_encrypt = tk.Button(actions\_frame, text="Зашифровать", command=self.encrypt\_file, state=tk.DISABLED)

self.btn\_encrypt.pack(side=tk.LEFT, padx=5, pady=5)

self.btn\_decrypt = tk.Button(actions\_frame, text="Расшифровать", command=self.decrypt\_file, state=tk.DISABLED)

self.btn\_decrypt.pack(side=tk.LEFT, padx=5, pady=5)

# Фрейм для объяснения алгоритма

explanation\_frame = tk.LabelFrame(main\_frame, text="Пояснение алгоритма Диффи-Хеллмана", padx=10, pady=10)

explanation\_frame.pack(fill=tk.BOTH, expand=True, pady=10)

self.txt\_explanation = scrolledtext.ScrolledText(explanation\_frame, wrap=tk.WORD, height=15, font=("Arial", 11))

self.txt\_explanation.pack(fill=tk.BOTH, expand=True)

self.txt\_explanation.insert(tk.END, self.get\_explanation\_text())

self.txt\_explanation.config(state=tk.DISABLED)

def get\_explanation\_text(self):

return """

Алгоритм обмена ключами Диффи-Хеллмана позволяет двум сторонам (назовем их Алиса и Боб) получить общий секретный ключ, обмениваясь информацией по незащищенному каналу. Этот ключ затем можно использовать для симметричного шифрования данных.

ВАЖНО: Сам по себе алгоритм Диффи-Хеллмана не шифрует данные, а только помогает безопасно выработать общий секрет. В этом приложении полученный секрет используется как ключ для надежного AES шифрования.

Процесс выработки общего ключа:

1. Публичные параметры:

- Алиса и Боб договариваются о двух числах, которые не являются секретными:

- p: очень большое простое число (модуль).

- g: целое число, первообразный корень по модулю p (основание).

2. Генерация секретных ключей:

- Алиса генерирует свой секретный ключ — случайное простое число 'x'. Она держит его в тайне.

- Боб генерирует свой секретный ключ — случайное простое число 'y'. Он также держит его в тайне.

(В этом приложении оба ключа 'x' и 'y' генерируются локально для демонстрации).

3. Вычисление и обмен открытыми ключами:

- Алиса вычисляет свой открытый ключ 'A' по формуле: A = g^x mod p

- Боб вычисляет свой открытый ключ 'B' по формуле: B = g^y mod p

- Алиса и Боб обмениваются своими открытыми ключами (A и B) по сети. Злоумышленник может их перехватить.

4. Вычисление общего секрета:

- Алиса, получив от Боба ключ B, вычисляет общий секрет 'S' по формуле: S = B^x mod p

- Боб, получив от Алисы ключ A, вычисляет общий секрет 'S' по формуле: S = A^y mod p

Магия в том, что оба результата будут одинаковыми:

S = (g^y mod p)^x mod p = (g^x mod p)^y mod p = g^(xy) mod p

Теперь у Алисы и Боба есть идентичный секретный ключ 'S', который не передавался по сети в открытом виде. Злоумышленник, зная g, p, A и B, не может легко вычислить 'S', так как для этого ему нужно решить задачу дискретного логарифмирования (найти 'x' из A=g^x mod p), что для больших чисел является очень сложной задачей.

5. Шифрование файла:

- Полученный общий секрет 'S' преобразуется в ключ фиксированной длины (например, с помощью хэш-функции SHA-256).

- Этим ключом с помощью симметричного алгоритма (в данном случае AES) шифруется содержимое файла.

"""

def select\_file(self):

self.filepath = filedialog.askopenfilename(

title="Выберите текстовый файл",

filetypes=(("Текстовые файлы", "\*.txt"), ("Все файлы", "\*.\*"))

)

if self.filepath:

self.lbl\_filepath.config(text=self.filepath)

self.btn\_encrypt.config(state=tk.NORMAL)

if self.filepath.endswith(".enc"):

self.btn\_decrypt.config(state=tk.NORMAL)

else:

self.btn\_decrypt.config(state=tk.DISABLED)

else:

self.lbl\_filepath.config(text="Файл не выбран")

self.btn\_encrypt.config(state=tk.DISABLED)

self.btn\_decrypt.config(state=tk.DISABLED)

def derive\_key(self, shared\_secret):

"""Преобразует числовой секрет в ключ для Fernet."""

# Хэшируем числовой секрет, чтобы получить 32-байтный ключ

sha256 = hashlib.sha256()

sha256.update(str(shared\_secret).encode('utf-8'))

# Fernet требует ключ в кодировке base64

return base64.urlsafe\_b64encode(sha256.digest())

def encrypt\_file(self):

if not self.filepath:

messagebox.showerror("Ошибка", "Сначала выберите файл.")

return

try:

with open(self.filepath, 'r', encoding='utf-8') as f:

content = f.read()

if len(content) < 256:

messagebox.showerror("Ошибка", "Файл должен содержать не менее 256 символов.")

return

except Exception as e:

messagebox.showerror("Ошибка чтения файла", f"Не удалось прочитать файл: {e}")

return

# 1. Генерация публичных параметров

p = generate\_prime(16) # Модуль (для демо 16 бит достаточно)

g = 5 # Основание (часто используемое простое число)

# 2. Генерация секретных ключей x и y

x = generate\_prime(12) # Секретный ключ "Алисы"

y = generate\_prime(12) # Секретный ключ "Боба"

# 3. Вычисление открытых ключей

A = pow(g, x, p)

B = pow(g, y, p)

# 4. Вычисление общего секрета

shared\_secret = pow(B, x, p)

# 5. Шифрование файла

key = self.derive\_key(shared\_secret)

fernet = Fernet(key)

encrypted\_content = fernet.encrypt(content.encode('utf-8'))

# Сохранение зашифрованного файла и файла с ключами

encrypted\_filepath = self.filepath + ".enc"

key\_filepath = self.filepath + ".key"

with open(encrypted\_filepath, 'wb') as f:

f.write(encrypted\_content)

# Для расшифровки нам понадобятся p, g, A и секретный ключ Боба 'y'

# В реальной системе 'y' хранился бы у Боба, а Алиса отправила бы ему 'A'

with open(key\_filepath, 'w') as f:

f.write(f"{p}\n")

f.write(f"{g}\n")

f.write(f"{A}\n") # Открытый ключ Алисы

f.write(f"{y}\n") # Секретный ключ Боба

messagebox.showinfo("Успех", f"Файл успешно зашифрован!\nСохранен как: {encrypted\_filepath}\nКлюч сохранен в: {key\_filepath}")

def decrypt\_file(self):

if not self.filepath or not self.filepath.endswith(".enc"):

messagebox.showerror("Ошибка", "Выберите файл с расширением .enc для расшифровки.")

return

key\_filepath = self.filepath.replace(".enc", ".key")

if not os.path.exists(key\_filepath):

messagebox.showerror("Ошибка", f"Файл ключа не найден: {key\_filepath}")

return

try:

with open(key\_filepath, 'r') as f:

p = int(f.readline().strip())

g = int(f.readline().strip())

A = int(f.readline().strip())

y = int(f.readline().strip())

except Exception as e:

messagebox.showerror("Ошибка", f"Не удалось прочитать файл ключа: {e}")

return

# Пересчитываем общий секрет с точки зрения "Боба"

shared\_secret = pow(A, y, p)

key = self.derive\_key(shared\_secret)

fernet = Fernet(key)

try:

with open(self.filepath, 'rb') as f:

encrypted\_content = f.read()

decrypted\_content = fernet.decrypt(encrypted\_content)

decrypted\_filepath = self.filepath.replace(".enc", ".dec.txt")

with open(decrypted\_filepath, 'w', encoding='utf-8') as f:

f.write(decrypted\_content.decode('utf-8'))

messagebox.showinfo("Успех", f"Файл успешно расшифрован!\nСохранен как: {decrypted\_filepath}")

except InvalidToken:

messagebox.showerror("Ошибка", "Ошибка расшифровки. Ключ неверный или данные повреждены.")

except Exception as e:

messagebox.showerror("Ошибка", f"Произошла ошибка: {e}")

if \_\_name\_\_ == "\_\_main\_\_":

root = tk.Tk()

app = CryptoApp(root)

root.mainloop()