МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

**«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИМЕНИ Н.Г. ЧЕРНЫШЕВСКОГО»**

Кафедра теоретических основ компьютерной безопасности и криптографии

**Схема подписи Эль – Гамаля**

ОТЧЁТ

ПО ДИСЦИПЛИНЕ

«КРИПТОГРАФИЧЕСКИЕ ПРОТОКОЛЫ»

студента 5 курса 531 группы

специальности 10.05.01 Компьютерная безопасность

факультета компьютерных наук и информационных технологий

Сенокосова Владислава Владимировича

|  |  |  |
| --- | --- | --- |
| Преподаватель, доцент | \_\_\_\_\_\_\_\_\_\_ | В.E. Новиков |
|  | подпись, дата |  |

Саратов 2024

**Содержание**

[1 Цель работы и порядок выполнения 3](#_Toc178281394)

[2 Цифровая подпись 4](#_Toc178281395)

[3 Схема подписи Эль-Гамаля 5](#_Toc178281396)

[4 Программная реализация подписи Эль-Гамаля 7](#_Toc178281397)

[5 Тестирование 10](#_Toc178281398)

[ЗАКЛЮЧЕНИЕ 12](#_Toc178281399)

[ПРИЛОЖЕНИЕ А 13](#_Toc178281400)

# **1 Цель работы и порядок выполнения**

**Цель работы** – изучение схемы подписи Эль-Гамаля и ее программная реализация

**Порядок выполнения:**

1 Изучить необходимую теорию для реализации схемы подписи Эль-Гамаля

2 Реализовать на языке Python программу, с функционалом подписи Эль-Гамаля, с пояснениями функций, использующихся в программе.

3 Протестировать работоспособность программы с возможными параметрами в том числе с попытками неправомерного изменения информации в числовых значениях и передаваемом сообщении.

# **2 Цифровая подпись**

Цифровая подпись представляет собой криптографический механизм, обеспечивающий целостность и подлинность сообщений или данных. Это важный элемент в современных системах безопасности, так как позволяет гарантировать, что сообщение было отправлено конкретным пользователем и не было изменено в процессе передачи. Цифровая подпись также предоставляет возможность проверки авторства сообщения с использованием открытого ключа подписанта.

**Основные свойства цифровой подписи:**

1. **Аутентификация** — цифровая подпись позволяет убедиться в том, что сообщение отправлено конкретным пользователем, известным как отправитель или подписант.
2. **Целостность** — если сообщение было изменено после подписания, цифровая подпись станет недействительной при проверке.
3. **Неотказуемость** — отправитель не может отказаться от отправки сообщения, так как его подпись уникально связана с его закрытым ключом.

**Как работает цифровая подпись:**

Процесс цифровой подписи включает два ключевых этапа:

1. **Создание подписи**: отправитель генерирует цифровую подпись, используя свой закрытый ключ и сообщение, которое нужно подписать.
2. **Проверка подписи**: получатель использует открытый ключ отправителя для проверки подписи и убедиться в подлинности сообщения.

Цифровая подпись основана на использовании асимметричной криптографии, где каждый пользователь имеет два ключа: **закрытый** (приватный) ключ, известный только подписанту, и **открытый** ключ, доступный всем для проверки подписи.

# **3 Схема подписи Эль-Гамаля**

Схема цифровой подписи Эль-Гамаля является одной из распространённых схем асимметричной криптографии, основанных на вычислительных сложностях дискретного логарифма. Этот алгоритм был предложен Тахером Эль-Гамалем в 1985 году и представляет собой расширение системы шифрования Эль-Гамаля для создания цифровых подписей.

**Параметры схемы:**

— большое простое число.

— первообразный корень

— закрытый ключ, случайно выбранное целое

=

Открытым ключом является  (y,g,p), закрытым ключом — число x

**Процесс создания подписи:**

1. Вычисляем хэш m от нашего сообщения (в текущей реализации будет использоваться )
2. Отправитель выбирает случайное число , взаимно простое с , которое будет разовым ключом для подписи сообщения.
3. Вычисляется значение = , которое используется в подписи.
4. Вычисляется значение где это мультипликативное обратное по модулю , которое находится с помощью расширенного алгоритма Евклида.
5. Цифровая подпись сообщения M представляется в виде пары .

**Процесс проверки подписи:**

Чтобы проверить подпись, получатель использует открытый ключ и подпись сообщения и проверяет достоверность следующим образом:

1. Проверяется выполнимость условий: и . Если хотя бы одно из них не выполняется, то подпись считается неверной.
2. Вычисляется хэш функция
3. Подпись считается верной, если выполняется сравнение:

Рассматриваемый алгоритм корректен в том смысле, что подпись, вычисленная по указанным выше правилам, будет принята при её проверке.

Преобразуя определение s, имеем:

Из малой теоремы ферма следует:

Схема подписи Эль-Гамаля позволяет обеспечить аутентификацию и защиту от подделки, основываясь на вычислительной сложности дискретного логарифма.

# **4 Программная реализация подписи Эль-Гамаля**

Программная реализация состоит из набора функций, рассмотрим каждую функцию отдельно.

1. Расширенный алгоритм Евклида . См. Рис. 1

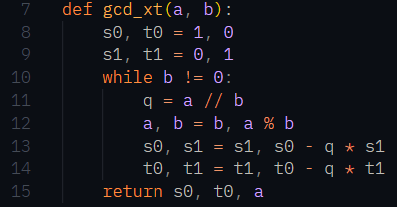


Рисунок 1 – Программная реализация расширенного алгоритма Евклида

Этот метод реализует расширенный алгоритм Евклида, который не только находит (наибольший общий делитель) двух чисел и , но и вычисляет коэффициенты и , такие что . Это необходимо для нахождения обратного элемента по модулю, который используется в процессе подписи сообщения.

1. Вычисление хэша файла . См. Рис. 2

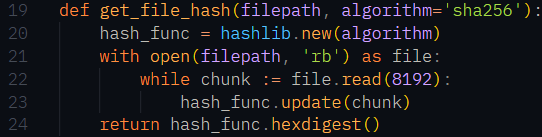


Рисунок 2 – Программная реализация вычисления хэша файла

Эта функция вычисляет хэш файла с использованием алгоритма хэширования, заданного параметром (по умолчанию используется ). Хэш-функция служит для того, чтобы представить сообщение как фиксированную строку определённой длины, которую можно подписывать. Файл считывается по частям, что позволяет обрабатывать даже большие файлы.

1. Генерация простого числа с заданной битностью См. Рис. 3



Рисунок 3 – Программная реализация генерации простого числа

Функция генерирует случайное простое число длиной в заданное количество бит с использованием библиотеки SymPy. Простое число требуется для параметра в схеме Эль-Гамаля.

1. Генерация числа , взаимно простого с . См. Рис. 4

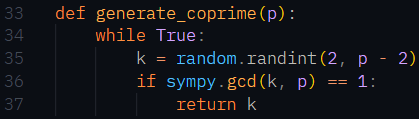


Рисунок 4 – Программная реализация генерации числа взаимно простого с числом p - 1

Эта функция генерирует случайное число , которое является взаимно простым с . Число используется как временный ключ в процессе подписания сообщения, и его взаимная простота с гарантирует, что можно будет найти его обратное по модулю.

1. Подпись сообщения . См. Рис. 5

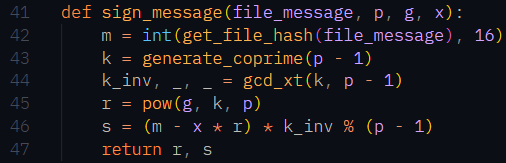


Рисунок 5 – Программная реализация подписи сообщения

Эта функция реализует процесс подписания сообщения по алгоритму Эль-Гамаля, который рассматривался выше

1. Генерация параметров . См. Рис. 6

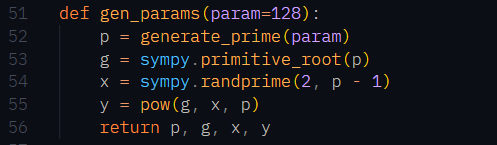


Рисунок 6 – Программная реализация генерации открытого и закрытого ключей

Функция генерирует параметры для открытого и закрытого ключей

1. Проверка подписи . См. Рис. 7

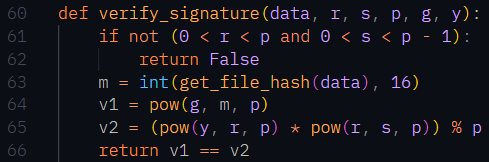


Рисунок 7 – Программная реализация подписи передаваемого сообщения

Эта функция проверяет цифровую подпись согласно схеме, представленной при описании схемы подписи Эль-Гамаля

**Основной блок программы**

В основном блоке программы выполняются следующие шаги:

1. Генерация или загрузка ключей.
2. Подпись файла.
3. Проверка подписи.

Программа выводит результат, показывая, изменялся ли файл в процессе передачи, что помогает подтвердить целостность и подлинность сообщения.

# **5 Тестирование**

Тестирование будем производить в несколько этапов:

1. Генерируем ключи и пробуем осуществить подпись с последующей проверкой достоверности данных. См.Рис. 8

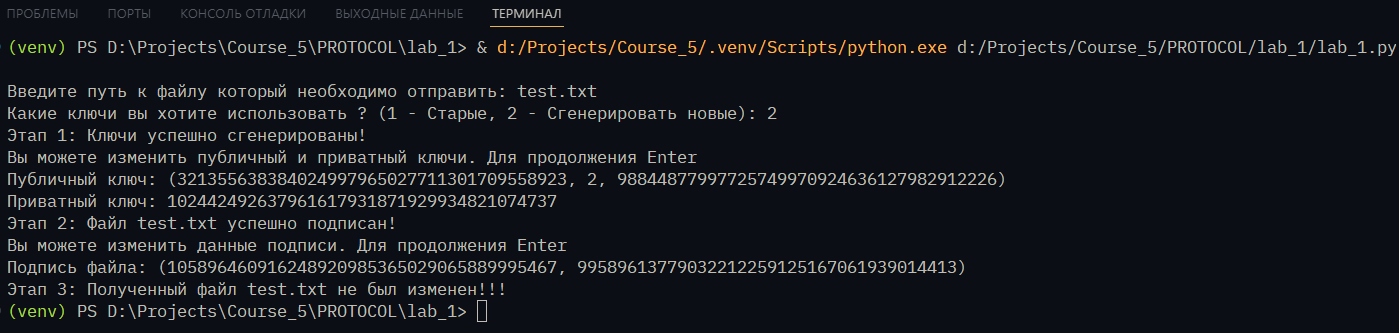


Рисунок 8 – Проверка подписи без вмешательства в функционал

Текст, который отсылается в файле test.txt:



Так как мы не меняли никакие параметры, то подпись была подтверждена.

1. Изменяем параметры открытого и закрытого ключей и смотрим на результат

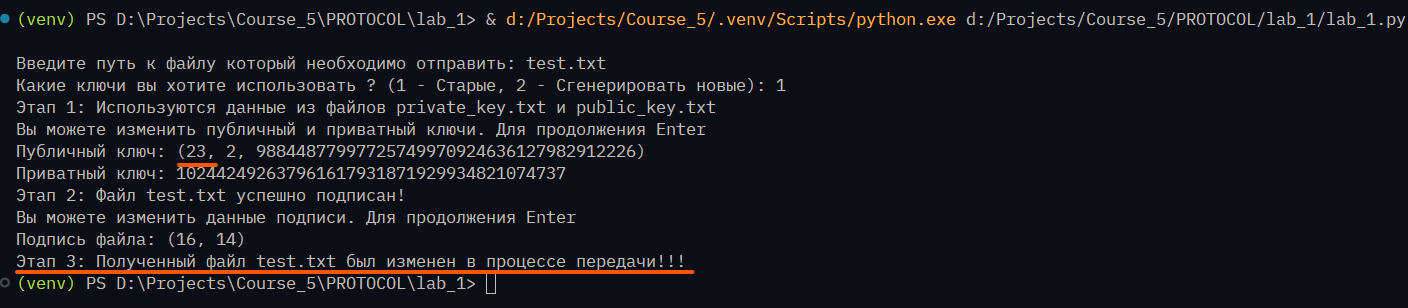


Рисунок 9 – Проверка подписи c вмешательством в открытый ключ

В результате мы видим, что получилось отследить изменение.

1. Изменяем параметры подписи и смотрим на результат

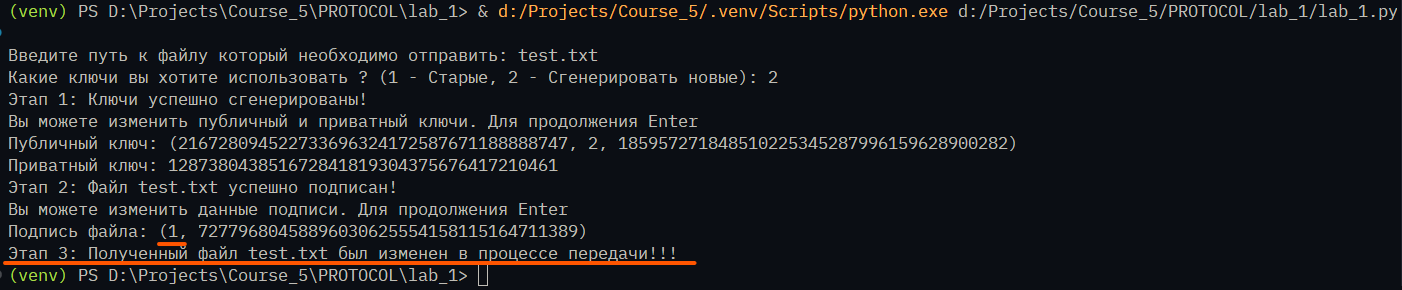


Рисунок 10 – Проверка подписи c вмешательством в подпись

В результате мы видим, что получилось отследить изменение.

1. Изменяем содержимое передаваемого файла.



Как видим, мы удалили часть передаваемого сообщения, что должно обнаружиться и это подтверждается:

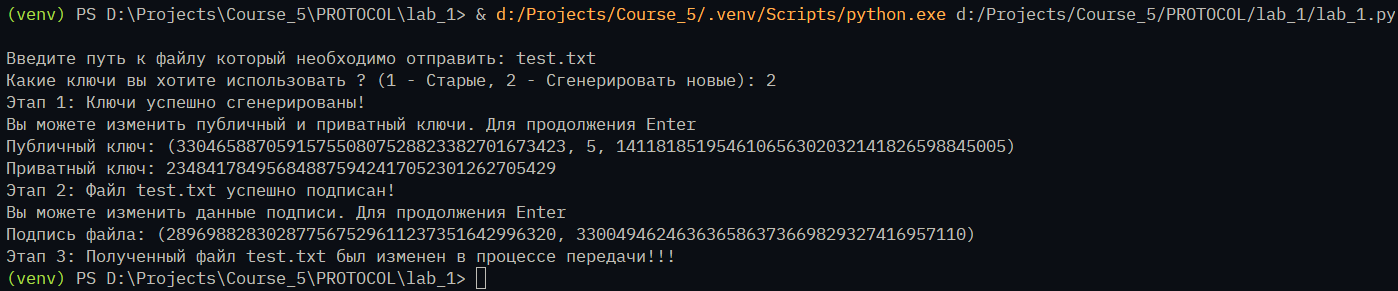


Рисунок 11 – Проверка подписи c вмешательством в текст передаваемого сообщения

# **ЗАКЛЮЧЕНИЕ**

В ходе выполнения данной работы была изучена теория, необходимая для реализации схемы цифровой подписи Эль-Гамаля, а также успешно реализована программа на языке Python, выполняющая этот алгоритм. В процессе разработки были созданы функции, которые обеспечивают генерацию ключей, создание цифровой подписи и проверку её подлинности. Каждый этап программы был детально пояснён.

Программа прошла тестирование с различными параметрами, включая тесты на целостность и защиту от изменения данных. Экспериментально было подтверждено, что при любом неправомерном изменении содержимого сообщения или параметров схемы, программа корректно обнаруживает изменения, что доказывает её работоспособность и устойчивость к подделке данных.

Таким образом, поставленная цель работы — изучение схемы подписи Эль-Гамаля и её программная реализация — была успешно достигнута.

# **ПРИЛОЖЕНИЕ А**

**Алгоритм подписи Эль-Гамаля**

import sympy

import hashlib

import random

*# Расширенный алгоритм Евклида*

def gcd\_xt(a, b):

    s0, t0 = 1, 0

    s1, t1 = 0, 1

    while b != 0:

        q = a // b

        a, b = b, a % b

        s0, s1 = s1, s0 - q \* s1

        t0, t1 = t1, t0 - q \* t1

    return s0, t0, a

*# Вычисление хэша файла*

def get\_file\_hash(filepath, algorithm='sha256'):

    hash\_func = hashlib.new(algorithm)

    with open(filepath, 'rb') as file:

        while chunk := file.read(8192):

            hash\_func.update(chunk)

    return hash\_func.hexdigest()

*# Генерирование простого числа с заданной битностью*

def generate\_prime(bits):

    return sympy.randprime(2\*\*(bits-1), 2\*\*bits)

*# Поиск числа взаимно простого с p*

def generate\_coprime(p):

    while True:

        k = random.randint(2, p - 2)

        if sympy.gcd(k, p) == 1:

            return k

*# Подпись сообщения*

def sign\_message(file\_message, p, g, x):

    m = int(get\_file\_hash(file\_message), 16)

    k = generate\_coprime(p - 1)

    k\_inv, \_, \_ = gcd\_xt(k, p - 1)

    r = pow(g, k, p)

    s = (m - x \* r) \* k\_inv % (p - 1)

    return r, s

*# Генерирование открытого и закрытого ключей*

def gen\_params(param=128):

    p = generate\_prime(param)

    g = sympy.primitive\_root(p)

    x = sympy.randprime(2, p - 1)

    y = pow(g, x, p)

    return p, g, x, y

*# Проверка подписи другой стороной*

def verify\_signature(data, r, s, p, g, y):

    if not (0 < r < p and 0 < s < p - 1):

        return False

    m = int(get\_file\_hash(data), 16)

    v1 = pow(g, m, p)

    v2 = (pow(y, r, p) \* pow(r, s, p)) % p

    return v1 == v2

*# Чтение публичного ключа*

def read\_public\_key(name\_file="public\_key.txt"):

    with open(name\_file, "r") as f:

            line = f.readline()

            lst = list(map(lambda x: int(x), line[1:-1].split(",")))

            p, g, y = lst[0], lst[1], lst[2]

    return p, g, y

*# Чтение приватного ключа*

def read\_private\_key(name\_file="private\_key.txt"):

    with open(name\_file, "r") as f:

            x = int(f.readline())

    return x

*# Подписание данных*

def reaa\_signed\_data(name\_file="signed\_data.txt"):

    with open(name\_file, "r") as f:

            line = f.readline()

            lst = list(map(lambda x: int(x), line[1:-1].split(",")))

            r, s = lst[0], lst[1]

    return r, s

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

    param = input("Введите режим работы (1 - Генерация ключей, 2 - Подпись документа, 3 - Проверка подписи): ")

    try:

        if param == "1":

*# Step 1 - Генерирование параметров открытого и закрытого ключа*

            param1 = input("1 - Использовать старые ключи, 2 - сгенерировать новые: ")

            if param1 == "2":

                p, g, x, y = gen\_params()

                with open("private\_key.txt", "w") as f:

                    f.write(str(x))

                with open("public\_key.txt", "w") as f:

                    f.write(f"({p}, {g}, {y})")

                print("Проход 1: Ключи успешно сгенерированы!")

                x1, x2, x3 = read\_public\_key()

                x4 = read\_private\_key()

                print(f"Публичный ключ: ({x1}, {x2}, {x3})")

                print(f"Приватный ключ: {x4}")

            elif param1 == "1":

                print("Проход 1: Используются данные из файлов private\_key.txt и public\_key.txt")

                x1, x2, x3 = read\_public\_key()

                x4 = read\_private\_key()

                print(f"Публичный ключ: ({x1}, {x2}, {x3})")

                print(f"Приватный ключ: {x4}")

        elif param == "2":

            file\_path = input("Введите путь к файлу который необходимо отправить: ")

*# Step 2 - Подпись тектого файла*

            x = read\_private\_key()

            p, g, y = read\_public\_key()

            r, s = sign\_message(file\_path, p, g, x)

            with open("signed\_data.txt", "w") as f:

                f.write(f"({r}, {s})")

            print(f"Проход 2: Файл {file\_path} успешно подписан!")

            print(f"Полученная подпись: ({r}, {s})")

        elif param == "3":

            file\_path = input("Введите путь к файлу, у которого надо проверть подпись: ")

*# Step 3 - Проверка полученного письма*

            r, s = reaa\_signed\_data()

            print(f"Подпись файла: ({r}, {s})")

            p, g, y = read\_public\_key()

            flag = verify\_signature(file\_path, r, s, p, g, y)

            if flag:

                print(f"Проход 3: Полученный файл {file\_path} не был изменен!!!")

            else:

                print(f"Проход 3: Полученный файл {file\_path} был изменен в процессе передачи!!!")

    except FileNotFoundError:

        print("Указан неправильный файл!")