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 Гиллу – Кискате.**

ОТЧЁТ

ПО ДИСЦИПЛИНЕ

«КРИПТОГРАФИЧЕСКИЕ ПРОТОКОЛЫ»

студента 5 курса 531 группы

специальности 10.05.01 Компьютерная безопасность

факультета компьютерных наук и информационных технологий

Мызникова Сергея Анатольевича

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

Саратов 2024

**1 Постановка задачи**

Цель работы: изучить алгоритм подписи Гиллу-Кискате и привести его программную реализацию

**2 Теоретические сведения**

Схема Фейге–Фиата–Шамира была первым практическим протоколом идентификации. Этот протокол минимизировал вычисления, увеличивая количество итераций и аккредитаций на итерацию. Для ряда реализаций, например, для интеллектуальных карточек, это неприемлемо. Обмены информацией с внешним миром требуют времени, а хранение данных для каждой аккредитации может быстро исчерпать ограниченные возможности карточки.

Луи Гиллу (Louis Guillou) и Жан-Жак Кискате (Jean-Jacques Quisquater) разработали алгоритм идентификации с нулевым разглашением. Обмены информацией между Пегги и Виктором, а также параллельные аккредитации в каждом обмене сведены к абсолютному минимуму: для каждого доказательства существует только один обмен, в котором предусмотрена только одна аккредитация. Для достижения того же уровня безопасности при использовании схемы Гиллу–Кискате потребуется выполнить в три раза больше вычислений, чем в схеме Фейге–Фиата–Шамира. Как и схему Фейге–Фиата–Шамира, этот алгоритм идентификации можно превратить в алгоритм цифровой подписи.

**Алгоритм генерации открытого и закрытого ключей**

Предположим, что роль A играет интеллектуальная карточка, которая собирается доказать свою подлинность B. Идентификация A проводится по ряду атрибутов, представляющих собой строку данных, содержащих название карточки, период ее действия, номер банковского счета и другие данные, подтверждающие ее правомочность. Эта битовая строка называется J. (В реальности строка атрибутов может быть очень длинной, а в качестве строки J используется ее хеш-значение. Такое усложнение никак не влияет на протокол.) Эта строка аналогична открытому ключу.

Алгоритм:

Сторона A отправляет стороне В свои атрибуты J. Стороне A необходимо убедить сторону В, что это именно её атрибуты. Для этого сторона A доказывает своё знание секрета x стороне B, не раскрывая при этом ни одного бита самого секрета x. Для этого сторонам потребуется всего 1 раунд.

1. Центр доверия T выбирает два различных случайных простых числа p и q, после чего вычисляет их произведение ;
2. T выбирает целое число e (1 < e < ), взаимно простое со значением функции . Функция является функция Эйлера;
3. T вычисляет и секрет ;
4. T вычисляет ;
5. Тройка {n, e, y} публикуется в качестве открытого ключа;
6. x играет роль закрытого ключа, и передается стороне А.

**Схема подписи**

Схема строится на основе схемы идентификации, открытый и закрытый ключи не меняются.

Пусть A хочет подписать сообщение *.*

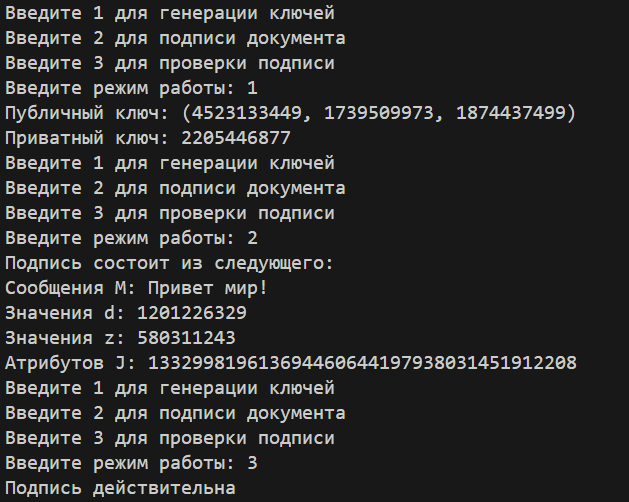
1. A выбирает случайное целое r, находящееся в диапазоне от 1 до n – 1. A вычисляет ;
2. A вычисляет , где – подписываемое сообщение, а – однонаправленная хеш-функция. Значение , полученное с помощью хеш-функции, должно быть взято по модулю ;
3. A вычисляет . Подпись состоит из сообщения , двух вычисленных значений и , и ее атрибутов . A посылает подпись B.

**Проверка подписи**

Пусть В хочет проверить подпись.

1. B вычисляет . Затем он вычисляет . Если = , то A знает B, и подпись действительна.

**3 Тестирование программы**



**ПРИЛОЖЕНИЕ А**

import random

import math

import hashlib

def is\_prime(n):

if n <= 1:

return False

for i in range(2, int(n \*\* 0.5) + 1):

if n % i == 0:

return False

return True

def hash\_file(filename):

hash = hashlib.md5()

with open(filename, "rb") as f:

for byte\_block in iter(lambda: f.read(4096), b""):

hash.update(byte\_block)

return hash.hexdigest()

def generate\_prime(limit):

prime\_candidate = random.randint(1000, limit)

while not is\_prime(prime\_candidate):

prime\_candidate = random.randint(1000, limit)

return prime\_candidate

def generate\_keys():

p = generate\_prime(100000)

q = generate\_prime(100000)

while (p == q):

p = generate\_prime(100000)

q = generate\_prime(100000)

n = p \* q

phi\_n = (p - 1) \* (q - 1)

e = random.randint(2, phi\_n - 1)

while math.gcd(e, phi\_n) != 1:

e = random.randint(2, phi\_n - 1)

s = pow(e, -1, phi\_n)

J = int(hash\_file('Alice.txt'), 16)

J\_s = pow(int(J), s, n)

x = pow(J\_s, -1, n)

y = pow(x, e, n)

public\_key = (n, e, y)

private\_key = (x,)

return public\_key, private\_key

def message\_signature(public, private, M):

n, e, y = public

x = private[0]

e, n, x = int(e), int(n), int(x)

r = random.randint(1, n - 1)

a = pow(r, e, n)

new\_message = f"{M}{a}"

d = hashlib.md5()

d.update(new\_message.encode('utf-8'))

d = int(d.hexdigest(), 16) % e

z = r \* pow(x, d, n) % n

J = int(hash\_file('Alice.txt'), 16)

return (M, d, z, J)

def write\_to\_file(files, message):

with open(files, 'w') as file:

file.write(','.join(map(str, message)))

def open\_file(files):

with open(files, 'r') as file:

data = tuple(map(str, file.read().split(',')))

return data

def signature\_verification(sign, public):

n, e, y = public

e, n = int(e), int(n)

M, d, z, J = sign

d, z, J = int(d), int(z), int(J)

a\_ = pow(z, e, n) \* pow(J, d, n) % n

d\_ = hashlib.md5()

new\_message = f'{M}{a\_}'

d\_.update(new\_message.encode('utf-8'))

d\_ = int(d\_.hexdigest(), 16) % e

if d\_ == d:

print('Подпись действительна')

else:

print('Подпись недействительна')

return

def main():

print("Введите 1 для генерации ключей")

print("Введите 2 для подписи документа")

print("Введите 3 для проверки подписи")

a = input("Введите режим работы: ")

if a == "1":

public\_key, private\_key = generate\_keys()

write\_to\_file('public\_key.txt', public\_key)

write\_to\_file('private\_key.txt', private\_key)

print(f"Публичный ключ: {public\_key}\nПриватный ключ: {int(private\_key[0])}")

elif a == "2":

try:

with open('input.txt', 'r', encoding='utf-8') as file:

plaintext = file.read()

public\_key = open\_file('public\_key.txt')

private\_key = open\_file('private\_key.txt')

if len(public\_key) == 0 or len(private\_key) == 0 or len(plaintext) == 0:

print("Error reading")

return

sign = message\_signature(public\_key, private\_key, plaintext)

with open('signature.txt', 'w', encoding='utf-8') as file:

file.write(str(sign))

print(f"Подпись состоит из следующего: \nСообщения M: {sign[0]}\nЗначения d: {sign[1]}\

\nЗначения z: {sign[2]}\nАтрибутов J: {sign[3]}")

except Exception as e:

print("Error: ", e)

elif a == "3":

try:

public\_key = open\_file('public\_key.txt')

with open('signature.txt', 'r', encoding='utf-8') as file:

sign = eval(file.read())

if len(sign) == 0 or len(public\_key) == 0:

print("Error reading")

return

signature\_verification(sign, public\_key)

except Exception as e:

print("Error: ", e)

else:

print("Неверный выбор")

while True:

main()