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 B свои атрибуты J. Стороне A необходимо убедить сторону B, 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 $n = p * q$;
2. T выбирает целое число e ($1 < e < \varphi(n)$), взаимно простое со значением функции $\varphi(n)$. Функция $\varphi(n)$ является функция Эйлера;
3. T вычисляет $s = e^{-1} \, mod \, \varphi(n)$ и секрет $x = J^{-s} mod \, n$;
4. T вычисляет $y = x^e \, mod \, n$;
5. Тройка {n, e, y} публикуется в качестве открытого ключа;
6. x играет роль закрытого ключа, и передается стороне A.

Схема подписи

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

Пусть A хочет подписать сообщение M.

1. A выбирает случайное целое r, находящееся в диапазоне от 1 до n – 1. A вычисляет $a = r^e \, mod \, n$;
2. A вычисляет $d = H(M, a)$, где M – подписываемое сообщение, а $H(x)$ – однонаправленная хеш-функция. Значение d, полученное с помощью хеш-функции, должно быть взято по модулю e;
3. A вычисляет $z = r * x^d \, mod \, n$. Подпись состоит из сообщения M, двух вычисленных значений d и z, и ее атрибутов J. A посылает подпись B.

Проверка подписи

Пусть B хочет проверить подпись.

1. B вычисляет $a' = z^e * J^d \ mod \ n$. Затем он вычисляет $d' = H(M, a')$. Если d' = $d \ mod \ e$, то A знает B, и подпись действительна.

3 Тестирование программы

```
Введите 1 для генерации ключей
Введите 2 для подписи документа
Введите 3 для проверки подписи
Введите режим работы: 1
Публичный ключ: (4523133449, 1739509973, 1874437499)
Приватный ключ: 2205446877
Введите 1 для генерации ключей
Введите 2 для подписи документа
Введите 3 для проверки подписи
Введите режим работы: 2
Подпись состоит из следующего:
Сообщения M: Привет мир!
Значения d: 1201226329
Значения z: 580311243
Атрибутов J: 133299819613694460644197938031451912208
Введите 1 для генерации ключей
Введите 2 для подписи документа
Введите 3 для проверки подписи
Введите режим работы: 3
Подпись действительна
```

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()
```