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

ЛАБОРАТОРНАЯ РАБОТА №4

Схема идентификации Гиллу-Кискате

студента 5 курса 531 группы
специальности 10.05.01 «Компьютерная безопасность»
факультета компьютерных наук и информационных технологий
Никитина Арсения Владимировича

Саратов 2023

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В данной лабораторной работе поставлена задача рассмотреть протокол идентификации с нулевым разглашением, на основе полученного материала реализовать схему идентификации Гиллу-Кискате.

1 Цель работы и порядок ее выполнения

Цель работы – изучение схемы идентификации Гиллу-Кискате и ее реализация.

Порядок выполнения работы:

1. Разобрать, что такое протокол идентификации с нулевым разглашением;
2. Разобрать протокол Гиллу-Кискате;
3. Произвести программную реализацию.

2 Теоретическая часть

Доказательство с нулевым разглашением в криптографии – это интерактивный криптографический протокол, позволяющий одной из взаимодействующих сторон убедиться в достоверности какого-либо утверждения (обычно математического), не имея при этом никакой другой информации от второй стороны.

Причём последнее условие является необходимым, так как обычно доказать, что сторона обладает определёнными сведениями в большинстве случаев тривиально, если она имеет право просто раскрыть информацию. Вся сложность состоит в том, чтобы доказать, что у одной из сторон есть информация, не раскрывая её содержание.

Протокол должен учитывать, что доказывающий сможет убедить проверяющего только в случае, если утверждение действительно доказано. В противном случае сделать это будет невозможно, или крайне маловероятно из-за вычислительной сложности.

Под интерактивностью протокола подразумевается непосредственный обмен информацией сторонами.

Данный криптографический протокол должен обладать тремя свойствами:

- Полнота: если утверждение действительно верно, то Доказывающий убедит в этом Проверяющего с любой наперед заданной точностью;

- Корректность: если утверждение неверно, то любой, даже «нечестный», Доказывающий не сможет убедить Проверяющего за исключением пренебрежимо малой вероятности;
- Нулевое разглашение: если утверждение верно, то любой, даже «нечестный», Проверяющий не узнает ничего кроме самого факта, что утверждение верно.

Протокол Гиллу-Кискате – это протокол идентификации с нулевым разглашением, расширение более раннего протокола Фиата-Шамира, разработанный Луи Гиллу, Жан-Жак Кискатр в 1988 году.

Протокол позволяет одному участнику (доказывающему А) доказать другому участнику (проверяющему В), что он обладает секретной информацией, не раскрывая ни единого бита этой информации.

Безопасность протокола основана на сложности извлечения квадратного корня по модулю достаточно большого составного числа по заданному модулю n.

В сравнении с протоколом Фиата-Шамира протокол Гиллу-Кискате имеет меньшее число сообщений, которыми необходимо поменяться сторонам для идентификации. Протокол требует только один раунд обмена сообщениями, имеет более низкие требования к памяти, используемой для хранения секретов пользователей, однако требует большего объёма вычислений.

Кроме того, схему идентификации Гиллу-Кискате можно легко преобразовать в схему подписи.

Схема идентификации

Вводные данные: Сторона А отправляет стороне В свои атрибуты *J*.

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

Алгоритм создания открытого и закрытого ключей

1. Центр доверия T выбирает два различных случайных простых числа p и q, после чего вычисляет их произведение $n = pq$;
2. T выбирает целое число $e\,(1 < e < \varphi(n))$, взаимно простое с $\varphi(n)$, где $\varphi(n)$ – функция Эйлера;
3. T вычисляет $s = e^{-1} \bmod \varphi(n)$ и секрет $x = J^{-s} \bmod n$;
4. T вычисляет $y = x^{e} \bmod n$;
5. Тройка $\{n, e, y\}$ публикуется в качестве открытого ключа;
6. x играет роль закрытого ключа, и передается стороне A.

Обмен сообщениями

1. **A** выбирает случайное целое r, находящееся в диапазоне от 1 до $n - 1$. **A** вычисляет $a = r^{e} \bmod n$ и отправляет его **B**;
2. **B** выбирает случайное целое c, находящееся в диапазоне от 0 до $e - 1$. **B** посылает c стороне **A**;
3. **A** вычисляет $z = rx^{c} \bmod n$ и посылает его **B**;
4. **B** проверяет: если $z^{e} = ay^{c} \bmod n$, то подлинность доказана.

3 Практическая часть

На рисунках 1 – 2 приведены примеры работы программы.

```
PS D:\lab> python crypto4.py
A отправляет стороне Б свои атрибуты: 675456789

Алгоритм создания открытого и закрытого ключей
Введите количество бит простого p: 10
Введите количество бит простого q: 10
Шаг 1
Параметр p: 971
Параметр q: 571
Результат шага - параметр n: 554441
Шаг 2
Параметр e: 78689
Шаг 3
Параметр s: 128309
Параметр x: 262155
Шаг 4
Параметр y: 161581

Открытый ключ: 554441 78689 161581
Закрытый ключ: 262155

Обмен сообщениями
1. Алиса отправляет Бобу вычисленное a: 88983
2. Боб отправляет Алисе выбранное c: 38970
3. Алиса отправляет Бобу вычисленное z: 489788
4. Боб проверяет z^e = ay^c mod n:  True
PS D:\lab>
```

Рисунок 1 – пример работы программы

```
PS D:\lab> python crypto4.py
A отправляет стороне Б свои атрибуты: 986755467890108

Алгоритм создания открытого и закрытого ключей
Введите количество бит простого p: 20
Введите количество бит простого q: 30
Шаг 1
Параметр p: 901861
Параметр q: 565091159
Результат шага - параметр n: 509633677746899
Шаг 2
Параметр e: 504139530610853
Шаг 3
Параметр s: 16012547991317
Параметр x: 289152072142505
Шаг 4
Параметр y: 4676774615560

Открытый ключ: 509633677746899 504139530610853 4676774615560
Закрытый ключ: 289152072142505

Обмен сообщениями
1. Алиса отправляет Бобу вычисленное a: 46721602996770
2. Боб отправляет Алисе выбранное c: 407433229707050
3. Алиса отправляет Бобу вычисленное z: 204608454478947
4. Боб проверяет z^e = ay^c mod n:  True
PS D:\lab>
```

Рисунок 2 – пример работы программы

ПРИЛОЖЕНИЕ А

Листинг программы

```
import random
import sympy
from Crypto.Util import number


def main():
    j = int(input("A отправляет стороне Б свои атрибуты: "))
    print("\nАлгоритм создания открытого и закрытого ключей")

    l1 = int(input("Введите количество бит простого p: "))
    p = number.getPrime(l1)
    l2 = int(input("Введите количество бит простого q: "))
    print("Шаг 1 \nПараметр p:", p)
    q = number.getPrime(l2)
    print("Параметр q:", q)
    n = p * q
    print("Результат шага - параметр n:", n)

    phi = (p - 1) * (q - 1)
    e = random.randint(2, phi)
    while sympy.gcd(e, phi) != 1:
        e = random.randint(1, phi)
    print("Шаг 2 \nПараметр e:", e)

    s = pow(e, -1, phi)
    x = pow(j, -s, n)
    print("Шаг 3 \nПараметр s:", s, "\nПараметр x:", x)

    y = pow(x, e, n)
    print("Шаг 4 \nПараметр y:", y, "\n")

    print("Открытый ключ:", n, e, y)
    print("Закрытый ключ:", x)

    print("\nОбмен сообщениями")
    r = random.randint(1, n)
    a = pow(r, e, n)
    print("1. Алиса отправляет Бобу вычисленное a:", a)

    c = random.randint(0, e)
    print("2. Боб отправляет Алисе выбранное c:", c)

    z = r * pow(x, c, n) % n
    print("3. Алиса отправляет Бобу вычисленное z:", z)

    print("4. Боб проверяет z^e = ay^c mod n: ", pow(z, e, n) == a *
pow(y, c, n) % n)


if __name__ == "__main__":
    main()
```