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 Шамира**

ОТЧЁТ

ПО ДИСЦИПЛИНЕ

«КРИПТОГРАФИЧЕСКИЕ ПРОТОКОЛЫ»

студента 5 курса 531 группы

специальности 10.05.01 Компьютерная безопасность

факультета компьютерных наук и информационных технологий

Таранова Алексея Вадимовича

|  |  |  |
| --- | --- | --- |
| Преподаватель  аспирант | \_\_\_\_\_\_\_\_\_\_ | Р. А. Фарахутдинов |
|  | подпись, дата |  |

Саратов 2023

**1. Теоретическая часть**

Цель работы – изучение алгоритма разделения секрета – схемы Шамира (интерполяционные полиномы Лагранжа) и его программная реализация.

Для интерполяции многочлена степени требуется точек. К примеру, для задания прямой достаточно двух точек, для задания параболы — трех точек, и так далее.

Основная идея данной схемы состоит в том, что интерполяция невозможна, если известно меньшее число точек.

Если мы хотим разделить секрет между людьми таким образом, чтобы восстановить его могли только человек , мы «прячем» его в формулу многочлена степени

Восстановить этот многочлен и исходный секрет можно только по точкам. Количество же различных точек многочлена ограничивается размером числового поля, в котором ведутся расчёты.

Алгоритм:

* Подготовительная фаза:

Пусть нужно разделить секрет *M* между *n* сторонами таким образом, чтобы любые участников могли бы восстановить секрет (то есть нужно реализовать

пороговую схему).

Выберем некоторое простое число *p > M.* Это число можно открыто сообщать всем участникам. Оно задаёт конечное поле размера . Над этим полем построим многочлен степени (то есть случайно выберем все коэффициенты многочлена, кроме )

В этом многочлене — это разделяемый секрет, а остальные коэффициенты .— некоторые случайные числа, которые нужно будет «забыть» после того, как процедура разделения секрета будет завершена.

* Генерация долей секрета

Теперь вычисляем «доли» — значения построенного выше многочлена, в

различных точках, причём .

Аргументы многочлена (номера секретов) не обязательно должны идти по порядку, главное — чтобы все они были различны по модулю .

После этого каждой стороне, участвующей в разделении секрета, выдаётся доля секрета вместе с номером .

Помимо этого, всем сторонам сообщается степень многочлена и размер поля Случайные коэффициенты и сам секрет M «забываются».

* Восстановление секрета

Теперь любые участников, зная координаты различных точек многочлена, смогут восстановить многочлен и все его коэффициенты, включая последний из них — разделяемый секрет.

Особенностью схемы является то, что вероятность раскрытия секрета в случае произвольных долей оценивается как .

То есть в результате интерполяции по точке секретом может быть любой элемент поля с равной вероятностью. При этом попытка полного перебора всех возможных теней не позволит злоумышленникам получить дополнительную информацию о секрете.

Прямолинейное восстановление коэффициентов многочлена через решение системы уравнений можно заменить на вычисление интерполяционного многочлена Лагранжа (отсюда одно из названий метода). Формула многочлена будет выглядеть следующим образом:

где — координаты точек многочлена. Все операции выполняются также в конечном поле p.

**2. Описание программы**

Реализация алгоритма состоит из основной программы (main.py), дилера dealer.py для генерации секрета, создания полинома и генерации частей секрета для хранителей и хранителей (keeper.py) где находятся части секрета.

Сначала вызывается InputSecret(self) из deeler.py: метод для ввода секрета с проверкой на то, что введено число. Затем вводится количество сторон, генерируется простое число и вводится минимальное число сторон для восстановления секрета. Вызывается GenerateCoefs(self, polynom\_degree, p) из deeler.py: метод для генерации случайных коэффициентов полинома от 1 до p.

Затем вычисляются значения полинома в точке, создаются доли секрета для конкретного пользователя используя ShareOfThePartSecret(self, keeper\_number, p) ) из deeler.py и сохраняются в классе для представления объекта, который является хранителем части секрета – PartOfSecret из keeper.py. Случайные коэффициенты забываются (Dealer1.ForgetCoefs()).

Теперь выводятся части секрета, используя CreateGroup(ListOfParticipants) генерируется группа участников, которые будут восстанавливать секрет (у пользователя спрашивается кто будет в этой группе).

После этого секрет восстанавливается с использованием RecoverSecret(Group, p), где реализован интерполяционный полином Лагранжа. Если восстановленный секрет и изначальный (который хранился только у дилера) равны, то проверка пройдена. Если же не равны, то выводится сообщение о том, что проверка не пройдена.

**3. Пример запуска программы**

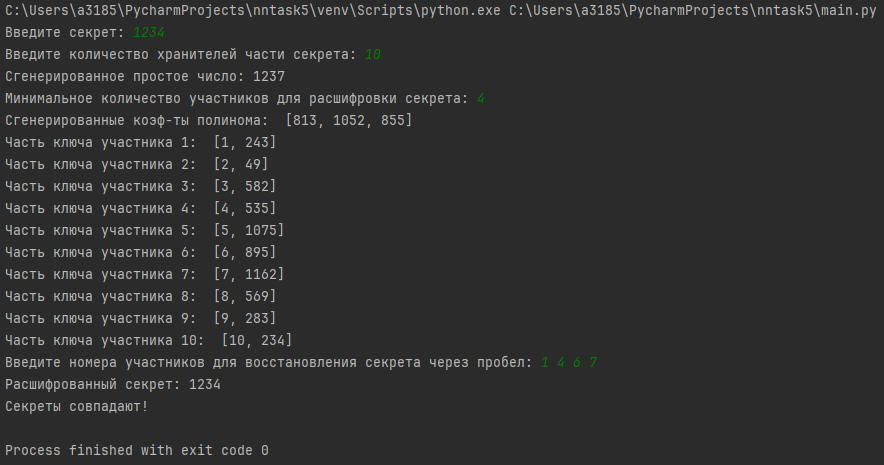
****

Рисунок 1 – Результат работы программы

**ПРИЛОЖЕНИЕ А**

Листинг 1 – main.py

from dealer import Dealer

from keeper import PartOfSecret

import sympy.ntheory as nt

from sympy import mod\_inverse

# Проверка на то, что было введено число.

def CheckInput(Message):

while True:

try:

Number = int(input(Message))

return Number

except ValueError:

print('Неверный ввод. Введите число!')

continue

def RecoverSecret(Group, p):

# Номера хранителей (точки) отделяются от долей секрета (значений полинома в точке).

ListOfX = []

ListOfY = []

for Keeper in Group:

ListOfX.append(Keeper.PairKey[0])

ListOfY.append(Keeper.PairKey[1])

# Интерполяционный полином Лагранжа.

LenX = len(ListOfX)

InterPoly = 0

# Вычисление li(x)

for i in range(LenX):

Yi = ListOfY[i]

Part = 1

for j in range(LenX):

if j == i:

continue

InverseDenom = mod\_inverse(ListOfX[j] - ListOfX[i], p)

Part \*= (ListOfX[j] % p) \* InverseDenom

# Вычисление значения полинома. Сумма li(x) \* yi

InterPoly += Yi \* Part

# После вычисления суммы - mod p

Secret = InterPoly % p

return Secret

# Создание группы участвующей в восстановлении секрета. Ввод через пробел.

# Проводится также проверка на то, что такие участники есть.

def CreateGroup(ListOfParticipants):

while True:

GroupOfParticipants = []

try:

input\_message = 'Введите номера участников для восстановления секрета через пробел: '

NumberOfParticipants = list(map(int, input(input\_message).split()))

for i in NumberOfParticipants:

FindParticipant = next(j for j in ListOfParticipants if j.PairKey[0] == i)

GroupOfParticipants.append(FindParticipant)

if len(GroupOfParticipants) == len(NumberOfParticipants):

return GroupOfParticipants

except StopIteration:

print('Один из участников не был найден. Повторите ввод!')

continue

# Основной код.

def main():

# Ввод секрета (число).

Dealer1 = Dealer()

Dealer1.InputSecret()

# Ввод количества сторон.

while 1:

n = CheckInput('Введите количество хранителей части секрета: ')

if n <= 1:

print('Количество участников должно быть строго больше 1')

else:

break

# Генерация простого числа p большего секрета и количества участников.

if n > Dealer1.Secret:

# Возвращает первое простое число, которое идет после переданного в функцию числа.

p = nt.nextprime(n)

else:

p = nt.nextprime(Dealer1.Secret)

print('Сгенерированное простое число: ' + str(p))

# Ввод минимального числа сторон для восстановления секрета.

while 1:

k = CheckInput('Минимальное количество участников для расшифровки секрета: ')

if k > n or k <= 1:

print('Минимальное количество участников должно быть меньше или равно общему количеству участников (от '

'2-ух)')

else:

break

# Генерация коэффициентов полинома и их вывод на экран.

Dealer1.GenerateCoefs(k - 1, p)

print('Сгенерированные коэф-ты полинома: ', Dealer1.Coefs)

# Список хранителей частей секрета.

ListOfKeepers = []

# Вычисление координат различных точек.

for i in range(1, n + 1):

Keeper = PartOfSecret(Dealer1.ShareOfThePartSecret(i, p))

ListOfKeepers.append(Keeper)

# Стираем случайные коэффициенты.

Dealer1.ForgetCoefs()

# Вывод частей ключей.

for KeeperKey in ListOfKeepers:

print('Часть ключа участника ' + str(KeeperKey.PairKey[0]) + ': ', KeeperKey.PairKey)

# Группа участников, которые будут восстанавливать секрет.

GroupOfKeepers = CreateGroup(ListOfKeepers)

# Восстановление секрета.

GroupSecret = RecoverSecret(GroupOfKeepers, p)

# Проверка на то, что изначальный секрет и восстановленный совпадают.

print('Расшифрованный секрет: ' + str(GroupSecret))

if Dealer1.Secret == GroupSecret:

print('Секреты совпадают!')

else:

print('Секреты отличаются!')

return 0

main()

**ПРИЛОЖЕНИЕ Б**

Листинг 2 – dealer.py

import random

class Dealer:

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

self.Secret = 0

self.Coefs = []

# Ввод секрета с проверкой на то, что введено число.

def InputSecret(self):

while True:

try:

self.Secret = int(input('Введите секрет: '))

break

except ValueError:

print('Неверный ввод. Введите число!')

# Генерация случайных коэффициентов полинома от 1 до p.

def GenerateCoefs(self, polynom\_degree, p):

self.Coefs = [random.randint(1, p) for \_ in range(polynom\_degree)]

# Обнуление коэффициентов полинома

def ForgetCoefs(self):

self.Coefs = []

# Вычисление значения полинома в точке. Создание доли секрета для конкретного пользователя.

def ShareOfThePartSecret(self, keeper\_number, p):

coefs\_with\_secret = self.Coefs + [self.Secret]

degree = len(coefs\_with\_secret)

# Значение полинома.

f = sum(coef \* keeper\_number \*\* (degree - i - 1) for i, coef in enumerate(coefs\_with\_secret))

return [keeper\_number, f % p]

**ПРИЛОЖЕНИЕ B**

Листинг 3 – keeper.py

class PartOfSecret:

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

self.PairKey = PairKey