Федеральное государственное автономное образовательное учреждение высшего образования «Санкт-Петербургский национальный исследовательский университет информационных технологий, механики и оптики»

Факультет программной инженерии и компьютерных технологий

Отчет по лабораторной работе № 9

“Получение ЭЦП на основе эллиптических кривых”

по дисциплине Информационная безопасность

Вариант 10

|  |  |
| --- | --- |
| Студент группы № P34151 | Шипулин Павел Андреевич |
| Преподаватель | Маркина Татьяна Анатольевна |

Санкт-Петербург

2024

# Цель работы

Сгенерировать ЭЦП для сообщения с известным значением хэш-свертки , зная секретный ключ подписи при данном значении выбираемого случайным образом числа . Используется эллиптическая кривая и генерирующая точка порядка .

# Вариант задания

|  |  |  |  |
| --- | --- | --- | --- |
| **№**  **варианта** |  |  |  |
| 10 | 3 | 3 | 11 |

# Ход работы

1. Ознакомиться с теорией.
2. Получить вариант у преподавателя.
3. Сгенерировать ЭЦП для сообщения.
4. Результаты и промежуточные вычисления оформить в виде отчета

# Листинг программ

Ссылка на репозиторий:

<https://github.com/PashcalE2/IS/tree/main/cryptography/second_part>

## Файл utils.py

def extended\_euclidean\_algorithm(a, b):s, old\_s = 0, 1  
 t, old\_t = 1, 0  
 r, old\_r = b, a  
  
 while r != 0:  
 quotient = old\_r // r  
 old\_r, r = r, old\_r - quotient \* r  
 old\_s, s = s, old\_s - quotient \* s  
 old\_t, t = t, old\_t - quotient \* t  
  
 return old\_r, old\_s, old\_t  
  
  
def inverse(n, p):gcd, x, y = extended\_euclidean\_algorithm(n, p)  
 assert (n \* x + p \* y) % p == gcd  
  
 if gcd != 1:  
 raise ValueError(  
 '{} has no multiplicative inverse '  
 'modulo {}'.format(n, p))  
 else:  
 return x % p  
  
  
\_p = 751  
  
  
class Point:  
 def \_\_init\_\_(self, x: int, y: int):  
 self.x = x  
 self.y = y  
  
 def \_\_add\_\_(self, other):  
 if self == other:  
 return self.double()  
  
 m = ((other.y - self.y) \* inverse(other.x - self.x, \_p)) % \_p  
 x = (m \* m - self.x - other.x) % \_p  
 y = (-self.y - m \* (x - self.x)) % \_p  
  
 return Point(x, y)  
  
 def double(self):  
 m = ((3 \* (self.x \*\* 2) - 1) \* inverse(2 \* self.y, \_p)) % \_p  
 x = (m \* m - 2 \* self.x) % \_p  
 y = (-self.y - m \* (x - self.x)) % \_p  
  
 return Point(x, y)  
  
 def mul(self, n: int):  
 P = self  
 nP = None  
  
 while n > 0:  
 if n % 2 == 1:  
 if nP is None:  
 nP = P  
 else:  
 nP = nP + P  
  
 P = P.double()  
  
 n >>= 1  
  
 return nP  
  
 def \_\_eq\_\_(self, other):  
 return (self.x == other.x) and (self.y == other.y)  
  
 def \_\_str\_\_(self):  
 return f"({self.x}, {self.y})"  
  
 def \_\_repr\_\_(self):  
 return self.\_\_str\_\_()

## Файл lab9.py

from utils import inverse, Point  
  
  
\_G = Point(416, 55)  
\_n = 13  
  
  
def lab9(e: int, d: int, k: int):  
 kG = \_G.mul(k)  
 print(f"kG = {kG}")  
  
 r = kG.x % \_n  
 print(f"r = {kG.x} % {\_n} = {r}")  
  
 z = inverse(k, \_n)  
 print(f"z = ({k}^-1) % {\_n} = {z}")  
  
 s = (z \* (e + d \* r)) % \_n  
 print(f"s = ({z} \* ({e} + {d} \* {r})) % {\_n} = {s}")  
  
 print(f"r, s = ({r}, {s})")  
  
 return r, s  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 result = lab9(3, 3, 11)

# Выполнение

## Результат выполнения программы

kG = (384, 276)

r = 384 % 13 = 7

z = (11^-1) % 13 = 6

s = (6 \* (3 + 3 \* 7)) % 13 = 1

r, s = (7, 1)

# Вывод

Ознакомился с идеей метода шифрования на основе эллиптических кривых. Реализовал алгоритм генерации электронной подписи на основе эллиптических кривых.