

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ**

**ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ  
«ДОНСКОЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»**

**(ДГТУ)**

Факультет «Информатика и вычислительная техника»

Кафедра «Кибербезопасность информационных систем»

Лабораторная работа №7

по дисциплине: «Методы алгебраической геометрии в криптографии»

на тему: «Генерация псевдослучайных последовательностей над эллиптическими кривыми»

Выполнил обучающийся гр. ВКБ41

Смирнова Александра Павловна

          Проверила:

Ст. пр. Артамонова Е.А.

Ростов-на-Дону

2025

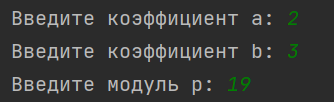
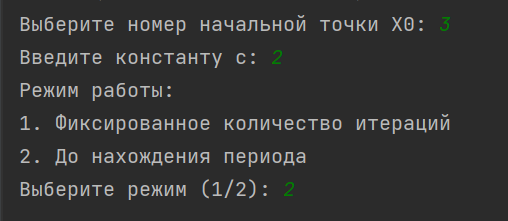
Лабораторная работа №7

«Генерация псевдослучайных последовательностей над эллиптическими кривыми»

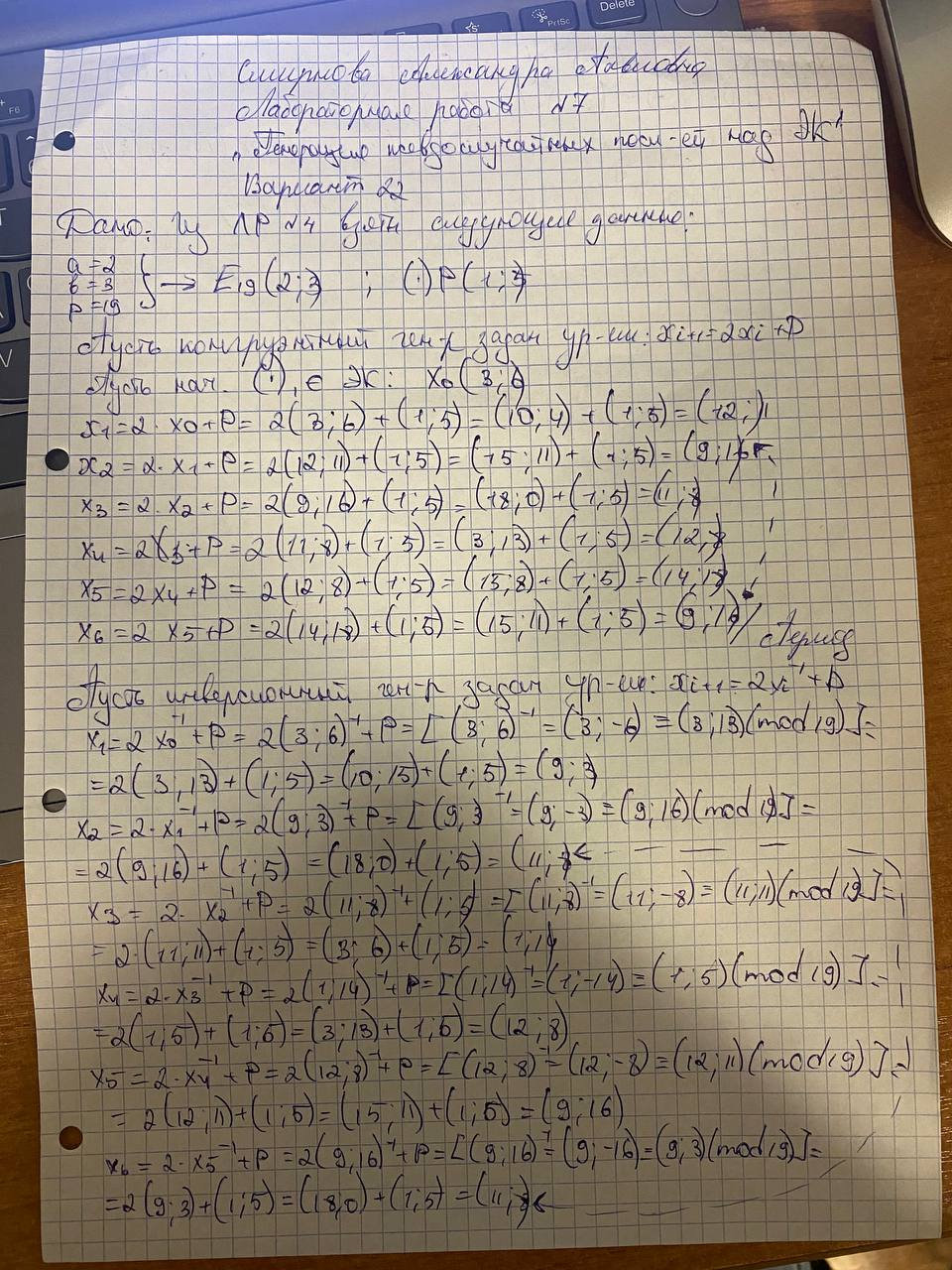
Вариант №22

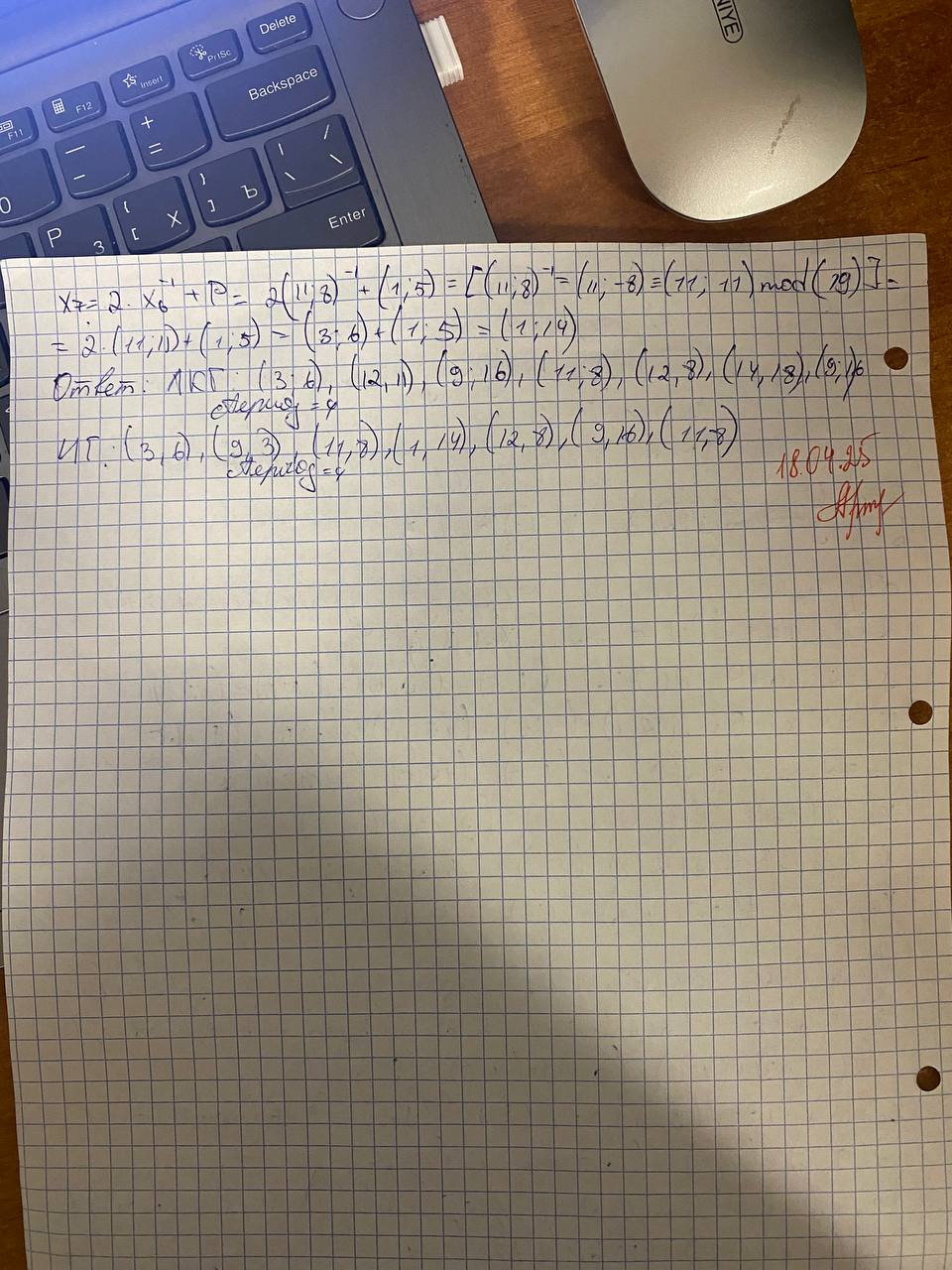
**Задание.** На эллиптической кривой 𝐸𝑝(𝑎;𝑏), полученной в лабораторной работе №4, найти псевдослучайную последовательность на основе линейного конгруэнтного генератора и инверсивного генератора. Характеристики генераторов подобрать самостоятельно. Написать программу, реализующие генерацию псевдослучайных последовательностей над эллиптической кривой на основе линейного конгруэнтного генератора и инверсивного генератора.

**Входные данные.**

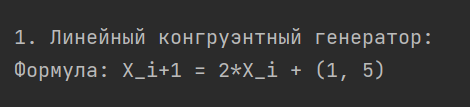
 ****

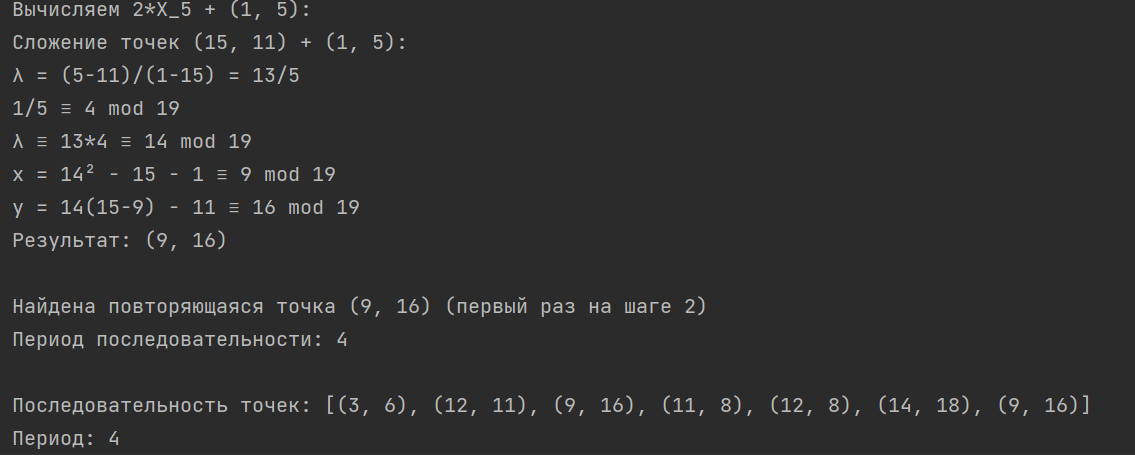
**Ручная отработка.**

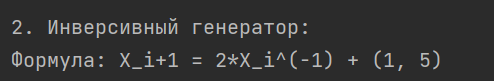


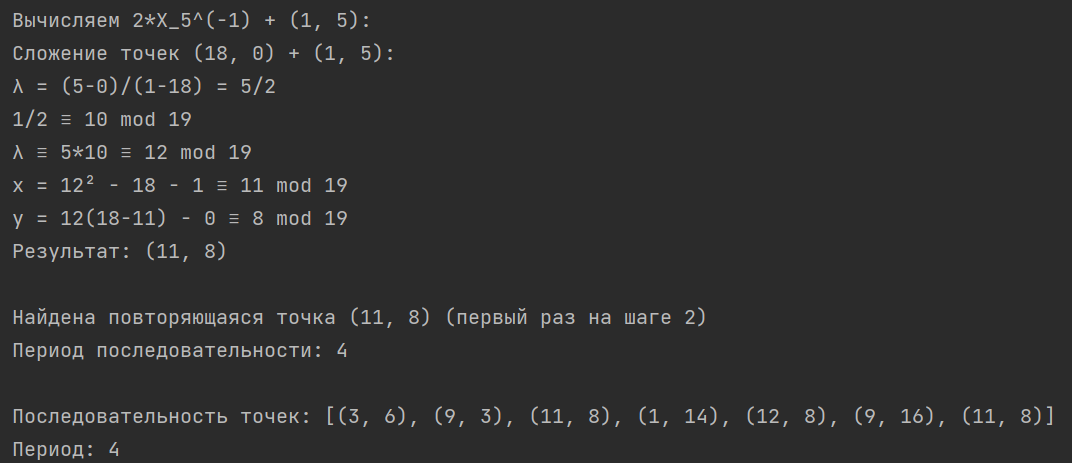


**Результат.**

****

****

****

****

**Листинг программы.**

import math  
from typing import List, Tuple, Optional, Dict  
  
  
def inverse\_mod(k: int, p: int) -> int:  
 *"""Вычисляет обратное k по модулю p"""* if k == 0:  
 raise ValueError("Деление на ноль")  
 return pow(k, -1, p)  
  
  
def find\_curve\_points(a: int, b: int, p: int) -> List[Optional[Tuple[int, int]]]:  
 *"""Находит все точки эллиптической кривой y² ≡ x³ + ax + b (mod p)"""* points = []  
  
 for x in range(p):  
 rhs = (pow(x, 3, p) + a \* x + b) % p  
 y\_values = []  
 for y in range(p):  
 if pow(y, 2, p) == rhs:  
 y\_values.append(y)  
 points.append((x, y))  
  
 print(f"x = {x}: y² ≡ {rhs} mod {p}", end=" ")  
 if y\_values:  
 print(f"→ Точки: {[(x, y) for y in y\_values]}")  
 else:  
 print("→ Нет решений")  
  
 points.append(None)  
 print("Добавлена точка в бесконечности O")  
 return points  
  
  
def elliptic\_add(P: Optional[Tuple[int, int]],  
 Q: Optional[Tuple[int, int]],  
 a: int, p: int, verbose: bool = False) -> Optional[Tuple[int, int]]:  
 *"""Сложение точек на эллиптической кривой"""* if P is None:  
 return Q  
 if Q is None:  
 return P  
  
 x1, y1 = P  
 x2, y2 = Q  
  
 if x1 == x2 and y1 == (-y2) % p:  
 if verbose: print(f"{P} + {Q} = O (точка в бесконечности)")  
 return None  
  
 if P == Q:  
 if verbose: print(f"Удвоение точки {P}:")  
 numerator = (3 \* x1 \* x1 + a) % p  
 denominator = (2 \* y1) % p  
 if verbose: print(f"λ = (3\*{x1}² + {a})/(2\*{y1}) = {numerator}/{denominator}")  
 else:  
 if verbose: print(f"Сложение точек {P} + {Q}:")  
 numerator = (y2 - y1) % p  
 denominator = (x2 - x1) % p  
 if verbose: print(f"λ = ({y2}-{y1})/({x2}-{x1}) = {numerator}/{denominator}")  
  
 try:  
 inv\_denominator = inverse\_mod(denominator, p)  
 if verbose: print(f"1/{denominator} ≡ {inv\_denominator} mod {p}")  
 except ValueError:  
 if verbose: print("Ошибка: обратный элемент не существует!")  
 return None  
  
 lam = (numerator \* inv\_denominator) % p  
 if verbose: print(f"λ ≡ {numerator}\*{inv\_denominator} ≡ {lam} mod {p}")  
  
 x3 = (lam \* lam - x1 - x2) % p  
 y3 = (lam \* (x1 - x3) - y1) % p  
 if verbose:  
 print(f"x = {lam}² - {x1} - {x2} ≡ {x3} mod {p}")  
 print(f"y = {lam}({x1}-{x3}) - {y1} ≡ {y3} mod {p}")  
 print(f"Результат: ({x3}, {y3})")  
  
 return (x3, y3)  
  
  
def elliptic\_multiply(P: Optional[Tuple[int, int]],  
 k: int,  
 a: int, p: int, verbose: bool = False) -> Optional[Tuple[int, int]]:  
 *"""Умножение точки на скаляр без лишних вычислений"""* if k == 0:  
 return None  
 if k == 1:  
 return P  
  
 result = None  
 current = P  
  
 while k > 0:  
 #if verbose: print(f"k = {k} (бинарно: {bin(k)})")  
 if k % 2 == 1:  
 if verbose: print(f"Добавляем текущую точку {current} к результату {result}")  
 result = elliptic\_add(result, current, a, p, verbose)  
 k = k // 2  
 if k > 0:  
 if verbose: print(f"Удваиваем точку {current}")  
 current = elliptic\_add(current, current, a, p, verbose)  
  
 return result  
  
  
def linear\_congruential\_ec\_generator(a: int, b: int, p: int,  
 c: int,  
 X0: Tuple[int, int],  
 P: Tuple[int, int],  
 max\_iter: int,  
 find\_period: bool) -> Tuple[List[Tuple[int, int]], List[int], int]:  
 *"""Линейный конгруэнтный генератор на эллиптической кривой"""* points = [X0]  
 bits = []  
 period = 0  
  
 print("\nВычисления для линейного конгруэнтного генератора:")  
 print(f"Начальная точка X0 = {X0}")  
  
 for i in range(1, max\_iter + 1):  
 print(f"\nШаг {i}: X\_{i} = {c}\*X\_{i - 1} + {P}")  
  
 print(f"Вычисляем {c}\*X\_{i - 1}:")  
 term = elliptic\_multiply(points[i - 1], c, a, p, verbose=True)  
  
 print(f"Вычисляем {c}\*X\_{i - 1} + {P}:")  
 next\_point = elliptic\_add(term, P, a, p, verbose=True)  
  
 points.append(next\_point)  
 if next\_point is None:  
 bits.append(0)  
 else:  
 bits.append(next\_point[1] & 1)  
  
 if find\_period:  
 if next\_point in points[:-1]:  
 first\_index = points.index(next\_point)  
 period = i - first\_index  
 print(f"\nНайдена повторяющаяся точка {next\_point} (первый раз на шаге {first\_index})")  
 print(f"Период последовательности: {period}")  
 return points[1:first\_index + period + 1], bits[:first\_index + period], period  
  
 if find\_period:  
 print("\nПериод не найден в заданном количестве итераций")  
 period = 0  
  
 return points[1:], bits, period  
  
  
def inverse\_ec\_generator(a: int, b: int, p: int,  
 c: int,  
 X0: Tuple[int, int],  
 P: Tuple[int, int],  
 max\_iter: int,  
 find\_period: bool) -> Tuple[List[Tuple[int, int]], List[int], int]:  
 *"""Инверсивный генератор на эллиптической кривой"""* points = [X0]  
 bits = []  
 period = 0  
  
 print("\nВычисления для инверсивного генератора:")  
 print(f"Начальная точка X0 = {X0}")  
  
 for i in range(1, max\_iter + 1):  
 print(f"\nШаг {i}: X\_{i} = {c}\*X\_{i - 1}^(-1) + {P}")  
  
 if points[i - 1] is None:  
 print("X\_{i-1} = O, обратная точка также O")  
 inv\_current = None  
 else:  
 x, y = points[i - 1]  
 inv\_current = (x, (-y) % p)  
 print(f"Обратная точка X\_{i - 1}^(-1) = {inv\_current}")  
  
 print(f"Вычисляем {c}\*X\_{i - 1}^(-1):")  
 term = elliptic\_multiply(inv\_current, c, a, p, verbose=True)  
  
 print(f"Вычисляем {c}\*X\_{i - 1}^(-1) + {P}:")  
 next\_point = elliptic\_add(term, P, a, p, verbose=True)  
  
 points.append(next\_point)  
 if next\_point is None:  
 bits.append(0)  
 else:  
 bits.append(next\_point[1] & 1)  
  
 if find\_period:  
 if next\_point in points[:-1]:  
 first\_index = points.index(next\_point)  
 period = i - first\_index  
 print(f"\nНайдена повторяющаяся точка {next\_point} (первый раз на шаге {first\_index})")  
 print(f"Период последовательности: {period}")  
 return points[1:first\_index + period + 1], bits[:first\_index + period], period  
  
 if find\_period:  
 print("\nПериод не найден в заданном количестве итераций")  
 period = 0  
  
 return points[1:], bits, period  
  
  
def select\_point(points: List[Optional[Tuple[int, int]]], prompt: str) -> Tuple[int, int]:  
 *"""Выбор точки из списка"""* print("\nДоступные точки:")  
 for i, point in enumerate(points):  
 if point is None:  
 print(f"{i + 1}. O (точка в бесконечности)")  
 else:  
 print(f"{i + 1}. {point}")  
  
 while True:  
 try:  
 idx = int(input(prompt)) - 1  
 if 0 <= idx < len(points):  
 if points[idx] is None:  
 print("Нельзя выбрать точку в бесконечности!")  
 continue  
 return points[idx]  
 print("Некорректный номер точки")  
 except ValueError:  
 print("Введите число")  
  
  
def main():  
 print("Генерация псевдослучайных последовательностей на эллиптических кривых")  
  
 p = int(input("Введите простое число p (модуль): "))  
 a = int(input("Введите коэффициент a: "))  
 b = int(input("Введите коэффициент b: "))  
  
 print("\nПоиск всех точек кривой:")  
 points = find\_curve\_points(a, b, p)  
  
 P = select\_point(points, "Выберите номер точки P: ")  
 X0 = select\_point(points, "Выберите номер начальной точки X0: ")  
  
 c = int(input("Введите константу c: "))  
  
 mode = input(  
 "Режим работы:\n1. Фиксированное количество итераций\n2. До нахождения периода\nВыберите режим (1/2): ")  
 if mode == '1':  
 max\_iter = int(input("Введите количество итераций: "))  
 find\_period\_flag = False  
 else:  
 max\_iter = p \* 3  
 find\_period\_flag = True  
  
 print("\n1. Линейный конгруэнтный генератор:")  
 print(f"Формула: X\_i+1 = {c}\*X\_i + {P}")  
 lc\_points, lc\_bits, lc\_period = linear\_congruential\_ec\_generator(a, b, p, c, X0, P, max\_iter, find\_period\_flag)  
 print("\nПоследовательность точек:", [X0] + lc\_points)  
 if lc\_period > 0:  
 print(f"Период: {lc\_period}")  
  
 print("\n2. Инверсивный генератор:")  
 print(f"Формула: X\_i+1 = {c}\*X\_i^(-1) + {P}")  
 inv\_points, inv\_bits, inv\_period = inverse\_ec\_generator(a, b, p, c, X0, P, max\_iter, find\_period\_flag)  
 print("\nПоследовательность точек:", [X0] + inv\_points)  
 if inv\_period > 0:  
 print(f"Период: {inv\_period}")  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()

**Вывод по работе:** по результатам лабораторной работы были приобретены навыки по нахождению псевдослучайной последовательности на основе линейного конгруэнтного генератора и инверсивного генератора.