Учреждение образования

«Белорусский государственный технологический университет»

**Кафедра информационных систем и технологий**

**«Отчёт по лабораторной работе №11»**

“Исследование криптографических алгоритмов на основе эллиптических кривых”

**Выполнил:** студент 4 курса

1 группы специальности ИСИт

Палазник Арсений Викторович

**Проверил:** преподаватель

Сазонова Дарья Владимировна

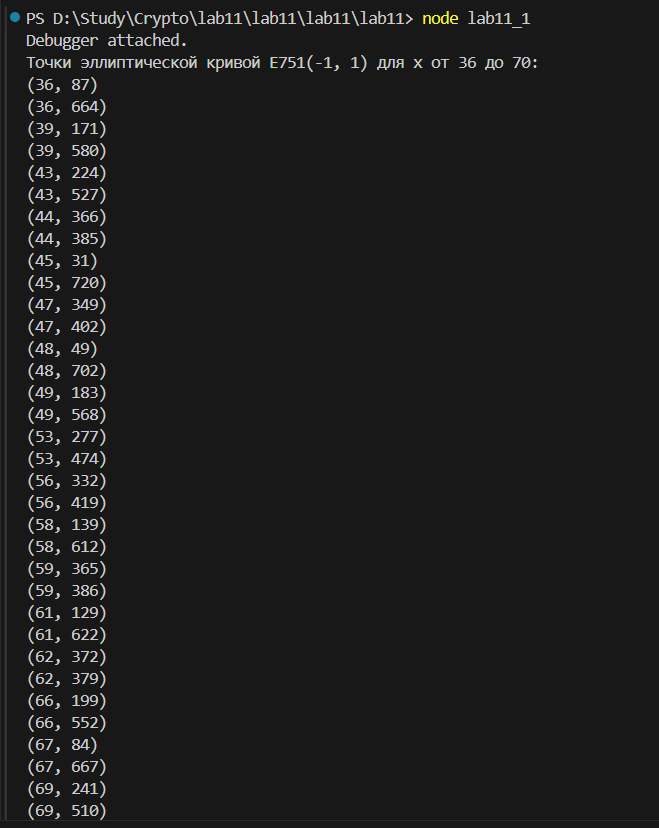
Минск 2024

**Найти точки ЭК для значений х, указанных в табл. 11.6**

Реализация приложения:

|  |
| --- |
| // Лабораторная работа №12: Поиск точек эллиптической кривой E751(-1, 1) для x от 36 до 70  // Язык программирования: JavaScript (Node.js)  // =============================  // Параметры эллиптической кривой E751(-1, 1)  // =============================  // Все параметры объявлены как BigInt для избежания ошибок типов  const a = -1n; // Коэффициент a  const b = 1n; // Коэффициент b  const p = 751n; // Простое число p (модуль)  const xmin = 36n; // Минимальное значение x  const xmax = 70n; // Максимальное значение x  // =============================  // Основные функции  // =============================  // Быстрое возведение в степень по модулю (алгоритм "быстрого возведения в степень")  function modPow(base, exponent, modulus) {  if (modulus === 1n) return 0n;  let result = 1n;  base = base % modulus;  while (exponent > 0n) {  if (exponent % 2n === 1n) { // Если нечетный  result = (result \* base) % modulus;  }  exponent = exponent >> 1n; // Деление на 2  base = (base \* base) % modulus;  }  return result;  }  // Проверка, является ли число квадратным вычетом по модулю p с использованием критерия Эйлера  function isQuadraticResidue(ySquared, p) {  const lhs = modPow(ySquared, (p - 1n) / 2n, p);  return lhs === 1n;  }  // Поиск всех y, удовлетворяющих y^2 ≡ RHS mod p  function findYs(RHS, p) {  const ys = [];  for (let y = 0n; y < p; y++) {  if (modPow(y, 2n, p) === RHS) {  ys.push(y);  }  }  return ys;  }  // Вычисление правой части уравнения кривой: RHS = x^3 - x + 1 mod p  function computeRHS(x, a, b, p) {  return (modPow(x, 3n, p) + a \* x + b) % p;  }  // Основная функция для поиска точек кривой в заданном диапазоне x  function findPoints(a, b, p, xmin, xmax) {  const points = [];  for (let x = xmin; x <= xmax; x++) {  const RHS = computeRHS(x, a, b, p);  // Проверяем, является ли RHS квадратным вычетом  if (isQuadraticResidue(RHS, p)) {  const ys = findYs(RHS, p);  ys.forEach(y => {  points.push({ x: Number(x), y: Number(y) });  });  }  }  return points;  }  // Запуск поиска точек  const pointsOnCurve = findPoints(a, b, p, xmin, xmax);  // Вывод результатов  if (pointsOnCurve.length === 0) {  console.log(`Нет точек на кривой E${p}(${a}, ${b}) для x от ${xmin} до ${xmax}.`);  } else {  console.log(`Точки эллиптической кривой E${p}(${a}, ${b}) для x от ${xmin} до ${xmax}:`);  pointsOnCurve.forEach(point => {  console.log(`(${point.x}, ${point.y})`);  });  } |

Результат выполнения:



**Разработать приложение для выполнения операций над точками кривой: а) kР; б) Р + Q; в) kР + lQ – R; г) Р – Q + R**

Реализация приложения:

|  |
| --- |
| // Лабораторная работа №12: Операции над точками эллиптической кривой E751(-1, 1)  // Язык программирования: JavaScript (Node.js)  // =============================  // Параметры эллиптической кривой E751(-1, 1)  // =============================  const a = -1n; // Коэффициент a  const b = 1n; // Коэффициент b  const p = 751n; // Простое число p (модуль)  const xmin = 36n; // Минимальное значение x  const xmax = 70n; // Максимальное значение x  // =============================  // Основные функции  // =============================  // Быстрое возведение в степень по модулю (алгоритм "быстрого возведения в степень")  function modPow(base, exponent, modulus) {  if (modulus === 1n) return 0n;  let result = 1n;  base = base % modulus;  while (exponent > 0n) {  if (exponent % 2n === 1n) { // Если нечетный  result = (result \* base) % modulus;  }  exponent = exponent >> 1n; // Деление на 2  base = (base \* base) % modulus;  }  return result;  }  // Проверка, является ли число квадратным вычетом по модулю p с использованием критерия Эйлера  function isQuadraticResidue(ySquared, p) {  const lhs = modPow(ySquared, (p - 1n) / 2n, p);  return lhs === 1n;  }  // Поиск всех y, удовлетворяющих y^2 ≡ RHS mod p  function findYs(RHS, p) {  const ys = [];  for (let y = 0n; y < p; y++) {  if (modPow(y, 2n, p) === RHS) {  ys.push(y);  }  }  return ys;  }  // Вычисление правой части уравнения кривой: RHS = x^3 - x + 1 mod p  function computeRHS(x, a, b, p) {  return (modPow(x, 3n, p) + a \* x + b) % p;  }  // Проверка принадлежности точки кривой  function checkPoint(P, a, b, p) {  if (P === 'O') return true; // Точка на бесконечности всегда принадлежит  const {x, y} = P;  return (modPow(y, 2n, p) === (modPow(x, 3n, p) + a \* x + b) % p);  }  // Поиск точки на кривой (используется при необходимости)  function findPointOnCurve(a, b, p) {  for (let x = 0n; x < p; x++) {  const rhs = (modPow(x, 3n, p) + a \* x + b) % p;  for (let y = 0n; y < p; y++) {  if (modPow(y, 2n, p) === rhs) {  return {x, y};  }  }  }  throw new Error("Подходящая точка не найдена");  }  // Сложение двух точек на кривой  function pointAddition(P, Q, a, p) {  // Обработка случаев, когда одна из точек - O (бесконечность)  if (P === 'O') return Q;  if (Q === 'O') return P;  const x1 = P.x;  const y1 = P.y;  const x2 = Q.x;  const y2 = Q.y;  if (x1 === x2 && (y1 + y2) % p === 0n) {  return 'O';  }  let lambda;  if (x1 === x2 && y1 === y2) {  // Удвоение точки P  if (y1 === 0n) {  return 'O';  }  const numerator = (3n \* x1 \* x1 + a) % p;  const denominator = (2n \* y1) % p;  const inv = modInverse(denominator, p);  if (inv === null) {  throw new Error("Не удалось найти обратное для удвоения точки");  }  lambda = (numerator \* inv) % p;  } else {  // Сложение различных точек P и Q  const numerator = (y2 - y1 + p) % p;  const denominator = (x2 - x1 + p) % p;  const inv = modInverse(denominator, p);  if (inv === null) {  throw new Error("Не удалось найти обратное для сложения точек");  }  lambda = (numerator \* inv) % p;  }  // Вычисление координат новой точки R = P + Q  const x3 = (modPow(lambda, 2n, p) - x1 - x2 + p) % p;  const y3 = (lambda \* (x1 - x3) - y1 + p) % p;  return {x: x3, y: y3};  }  // Нахождение обратного элемента по модулю p с использованием расширенного алгоритма Евклида  function modInverse(a, p) {  let t = 0n, newT = 1n;  let r = p, newR = a;  while (newR !== 0n) {  const quotient = r / newR;  [t, newT] = [newT, t - quotient \* newT];  [r, newR] = [newR, r - quotient \* newR];  }  if (r > 1n) return null; // Обратного элемента не существует  if (t < 0n) t += p;  return t;  }  // Умножение точки P на скаляр k (скалярное умножение) с использованием метода "Double-and-Add"  function scalarMultiplication(k, P, a, p) {  let result = 'O'; // Начинаем с нейтрального элемента  let addend = P;  while (k > 0n) {  if (k % 2n === 1n) {  result = pointAddition(result, addend, a, p);  }  addend = pointAddition(addend, addend, a, p);  k = k / 2n;  }  return result;  }  // =============================  // Основная функция для выполнения операций  // =============================  function main() {  console.log("=== Лабораторная работа №12 ===");  console.log(`Эллиптическая кривая E${p}(${a}, ${b})`);  console.log(`Диапазон x: ${xmin} до ${xmax}\n`);  // Поиск точек на кривой для заданного диапазона x  const pointsOnCurve = [];  for (let x = xmin; x <= xmax; x++) {  const RHS = computeRHS(x, a, b, p);  if (isQuadraticResidue(RHS, p)) {  const ys = findYs(RHS, p);  ys.forEach(y => {  pointsOnCurve.push({x, y});  });  }  }  // Вывод найденных точек  if (pointsOnCurve.length === 0) {  console.log(`Нет точек на кривой E${p}(${a}, ${b}) для x от ${xmin} до ${xmax}.`);  } else {  console.log(`Точки эллиптической кривой E${p}(${a}, ${b}) для x от ${xmin} до ${xmax}:`);  pointsOnCurve.forEach(point => {  console.log(`(${point.x}, ${point.y})`);  });  }  // Проверка, что есть как минимум 3 точки для P, Q, R  if (pointsOnCurve.length < 3) {  console.error("Недостаточно точек для выполнения операций (требуются как минимум 3 точки).");  return;  }  // Выбор точек P, Q, R (первые три точки)  const P = pointsOnCurve[0];  const Q = pointsOnCurve[1];  const R = pointsOnCurve[2];  console.log("\nВыбранные точки:");  console.log(`P: (${P.x}, ${P.y})`);  console.log(`Q: (${Q.x}, ${Q.y})`);  console.log(`R: (${R.x}, ${R.y})\n`);  // Заданные скаляры  const k = 6n;  const l = 10n;  // а) kP  const kP = scalarMultiplication(k, P, a, p);  if (kP === 'O') {  console.log(`а) ${k}P = O (точка на бесконечности)`);  } else {  console.log(`а) ${k}P = (${kP.x}, ${kP.y})`);  }  // б) P + Q  const P\_plus\_Q = pointAddition(P, Q, a, p);  if (P\_plus\_Q === 'O') {  console.log(`б) P + Q = O (точка на бесконечности)`);  } else {  console.log(`б) P + Q = (${P\_plus\_Q.x}, ${P\_plus\_Q.y})`);  }  // в) kP + lQ - R  // Вычисляем kP  const kp = scalarMultiplication(k, P, a, p);  // Вычисляем lQ  const lq = scalarMultiplication(l, Q, a, p);  // Суммируем kP + lQ  let kp\_plus\_lq;  if (kp === 'O') {  kp\_plus\_lq = lq;  } else if (lq === 'O') {  kp\_plus\_lq = kp;  } else {  kp\_plus\_lq = pointAddition(kp, lq, a, p);  }  // Вычитание R: kp\_plus\_lq - R = kp\_plus\_lq + (-R)  const minusR = {x: R.x, y: (p - R.y) % p};  let kp\_plus\_lq\_minus\_r;  if (kp\_plus\_lq === 'O') {  kp\_plus\_lq\_minus\_r = minusR;  } else {  kp\_plus\_lq\_minus\_r = pointAddition(kp\_plus\_lq, minusR, a, p);  }  if (kp\_plus\_lq\_minus\_r === 'O') {  console.log(`в) ${k}P + ${l}Q - R = O (точка на бесконечности)`);  } else {  console.log(`в) ${k}P + ${l}Q - R = (${kp\_plus\_lq\_minus\_r.x}, ${kp\_plus\_lq\_minus\_r.y})`);  }  // г) P - Q + R  // P - Q = P + (-Q)  const minusQ = {x: Q.x, y: (p - Q.y) % p};  let p\_minus\_q;  if (P === 'O') {  p\_minus\_q = minusQ;  } else if (minusQ === 'O') {  p\_minus\_q = P;  } else {  p\_minus\_q = pointAddition(P, minusQ, a, p);  }  // P - Q + R  let p\_minus\_q\_plus\_r;  if (p\_minus\_q === 'O') {  p\_minus\_q\_plus\_r = R;  } else if (R === 'O') {  p\_minus\_q\_plus\_r = p\_minus\_q;  } else {  p\_minus\_q\_plus\_r = pointAddition(p\_minus\_q, R, a, p);  }  if (p\_minus\_q\_plus\_r === 'O') {  console.log(`г) P - Q + R = O (точка на бесконечности)`);  } else {  console.log(`г) P - Q + R = (${p\_minus\_q\_plus\_r.x}, ${p\_minus\_q\_plus\_r.y})`);  }  }  // Запуск основной функции  main(); |

Результат выполнения:

