

Министерство образования Республики Беларусь
Учреждение образования «Белорусский государственный университет
информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра информатики

Дисциплина: Методы защиты информации

ОТЧЁТ
к лабораторной работе №7
на тему

**КРИПТОГРАФИЯ С ИСПОЛЬЗОВАНИЕМ ЭЛЛИПТИЧЕСКИХ
КРИВЫХ**

Выполнил: студент гр.253501
Станишевский А.Д.

Проверил: ассистент кафедры информатики
Герчик А.В.

Минск 2025

СОДЕРЖАНИЕ

1 Цель работы	3
2 Теоретические сведения	4
3 Ход работы.....	5
Заключение	6
Приложение А (обязательное) Листинг программного кода	7

1 ЦЕЛЬ РАБОТЫ

Целью данной лабораторной работы является изучение принципов работы асимметричных криптосистем на основе эллиптических кривых на примере аналога алгоритма шифрования Эль-Гамала. В рамках работы требуется разработать программное средство для шифрования и дешифрования данных, закрепить практические навыки работы с эллиптическими кривыми и освоить ключевые этапы обеспечения конфиденциальности информации.

Результатом выполнения работы должно быть программное средство, который позволяет:

- 1 Генерировать ключевую пару: закрытый ключ (скаляр) и открытый ключ (точка на кривой).
- 2 Шифровать текстовое сообщение с использованием открытого ключа получателя.
- 3 Дешифровать полученный шифротекст с использованием закрытого ключа получателя.

2 ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Алгоритм реализует асимметричную криптосистему Эль-Гамала, адаптированную для работы с эллиптическими кривыми. В основе лежит широко известная криптографическая кривая secp256k1 , заданная уравнением вида $y^2 = x^3 + 7$ над конечным полем простого порядка p , где все арифметические операции выполняются по модулю этого большого простого числа. Кривая обладает заранее определённой базовой точкой, которая используется как генератор циклической подгруппы порядка n . Эта подгруппа обеспечивает необходимую криптостойкость: задача дискретного логарифмирования на ней считается вычислительно неосуществимой для современных компьютеров.

Процесс начинается с генерации ключевой пары. Приватный ключ выбирается как случайное целое число в диапазоне от 1 до $n-1$, а публичный ключ вычисляется путём скалярного умножения базовой точки на приватный ключ.

Особую сложность представляет преобразование произвольного текстового сообщения в точку, лежащую на кривой, поскольку напрямую интерпретировать байты сообщения как координаты невозможно – большинство таких пар не будут удовлетворять уравнению кривой. Для решения этой задачи применяется метод проб: исходное сообщение сначала кодируется в большое целое число, затем к нему добавляется один дополнительный байт-суффикс, перебираемый последовательно от 0 до 255. Для каждого полученного кандидата на координату x вычисляется правая часть уравнения кривой, после чего предпринимается попытка извлечь квадратный корень по модулю. Как только находится такое значение суффикса, при котором квадратный корень существует, формируется точка с координатами (x, y) , которая и становится представлением исходного сообщения на кривой. Чтобы реализовать обратное преобразование достаточно отбросить младший байт координаты x , чтобы восстановить исходное числовое значение i , следовательно, текст.

Шифрование осуществляется с использованием публичного ключа получателя и случайного скаляра. Генерируются две компоненты шифротекста: первая – результат умножения базовой точки на случайный скаляр, вторая – сумма точки-сообщения и результата умножения публичного ключа на тот же случайный скаляр. Дешифрование использует приватный ключ: из второй компоненты вычитается результат умножения приватного ключа на первую компоненту шифротекста. Благодаря свойствам скалярного умножения, случайный множитель сокращается, и остаётся исходная точка-сообщение, из которой затем восстанавливается текст.

3 ХОД РАБОТЫ

Программное средство реализовано при помощи языка программирования JavaScript. На рисунке 3.1 изображен процесс шифрования исходного сообщения.

Шифрование сообщения

Введите сообщение для шифрования

hello привет 5

ЗАШИФРОВАТЬ СООБЩЕНИЕ

Зашифрованное сообщение:

C1:

(19455592382363030285479707594045959843071970887
7504550594151273708446622716,

C2:

(11105187571726083151238999890261219836640008689
2462937706914453078857908042385,

Рисунок 3.1 – Процесс шифрования исходного сообщения

На рисунке 3.2 изображен процесс дешифрования.

Зашифрованное сообщение:

C1:

(19455592382363030285479707594045959843071970887
7504550594151273708446622716,

C2:

(11105187571726083151238999890261219836640008689
2462937706914453078857908042385,

Дешифрование

ДЕШИФРОВАТЬ **ОЧИСТИТЬ ВСЕ**

Результат дешифрования:

hello привет 5

✓ Сообщение успешно расшифровано

Рисунок 3.2 – Процесс дешифрования

Таким образом, реализованная программа успешно справляется с поставленными задачами.

ЗАКЛЮЧЕНИЕ

В ходе выполнения лабораторной работы были изучены теоретические основы и практические аспекты применения криптографии на эллиптических кривых для обеспечения конфиденциальности данных. Разработанное программное средство успешно реализует полный цикл схемы шифрования, аналога схемы Эль-Гамала, демонстрируя процессы генерации ключей, шифрования с помощью открытого ключа и дешифрования с помощью закрытого.

Полученные навыки позволяют глубже понять механизмы асимметричного шифрования и обмена ключами на основе задачи дискретного логарифмирования на эллиптической кривой. Цель лабораторной работы полностью достигнута.

ПРИЛОЖЕНИЕ А

(обязательное)

Листинг программного кода

```
const CURVE = {
  a: 0n,
  b: 7n,
  p:
11579208923731619542357098500868790785326998466564056403945758400790883467166
3n,
  n:
11579208923731619542357098500868790785283756427907490438260516314151816149433
7n,
  G: {
    x:
55066263022277343669578718895168534326250603453777594175500187360389116729240
n,
    y:
32670510020758816978083085130507043184471273380659243275938904335757337482424
n
  }
};

class Point {
  constructor(x, y) {
    this.x = x;
    this.y = y;
  }

  equals(other) {
    return this.x === other.x && this.y === other.y;
  }

  toString() {
    return `${this.x}, ${this.y}`;
  }
}

class ECEG {
  constructor() {
    this.curve = CURVE;
  }

  mod(a, b) {
    const result = ((a % b) + b) % b;
    return result;
  }

  modInverse(a, m) {
    if (m === 0n) {
      throw new Error("modul not zero");
    }
    if (m < 0n) {
      throw new Error("modul not negative");
    }

    a = this.mod(a, m);
    if (a === 0n) {
      throw new Error("cant find inverse of zero");
    }
  }
}
```

```

let [old_r, r] = [a, m];
let [old_s, s] = [1n, 0n];
let [old_t, t] = [0n, 1n];

while (r !== 0n) {
  const 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];
}

if (old_r !== 1n) {
  throw new Error("number has no inverse");
}

return this.mod(old_s, m);
}

isOnCurve(point) {
  if (point.x === null && point.y === null) return true;

  const { a, b, p } = this.curve;
  const left = this.mod(point.y * point.y, p);
  const right = this.mod(this.mod(point.x * point.x * point.x, p) + this.mod(a
* point.x, p) + b, p);
  return left === right;
}

addPoints(P, Q) {
  const { p } = this.curve;

  if (P.x === null && P.y === null) return Q;
  if (Q.x === null && Q.y === null) return P;

  if (P.x === Q.x && P.y !== Q.y) {
    return new Point(null, null);
  }

  let lambda;
  if (P.equals(Q)) {
    if (P.y === 0n) {
      return new Point(null, null);
    }
    const numerator = this.mod(3n * P.x * P.x + this.curve.a, p);
    const denominator = this.mod(2n * P.y, p);
    lambda = this.mod(numerator * this.modInverse(denominator, p), p);
  } else {
    const numerator = this.mod(Q.y - P.y, p);
    const denominator = this.mod(Q.x - P.x, p);
    lambda = this.mod(numerator * this.modInverse(denominator, p), p);
  }

  const x3 = this.mod(lambda * lambda - P.x - Q.x, p);
  const y3 = this.mod(lambda * (P.x - x3) - P.y, p);

  return new Point(x3, y3);
}

multiplyPoint(k, point) {
  if (k === 0n) return new Point(null, null);
  if (k < 0n) {
    const positiveResult = this.multiplyPoint(-k, point);
    return new Point(positiveResult.x, this.mod(-positiveResult.y,
this.curve.p));
  }

```



```

    }
    if (k === 1n) return new Point(point.x, point.y);

    let result = new Point(null, null);
    let addend = new Point(point.x, point.y);
    let kTemp = k;

    while (kTemp > 0n) {
        if (kTemp & 1n) {
            result = this.addPoints(result, addend);
        }
        addend = this.addPoints(addend, addend);
        kTemp = kTemp >> 1n;
    }

    return result;
}

generateKeys() {
    const privateKey = this.generatePrivateKey();
    const publicKey = this.multiplyPoint(privateKey, new Point(this.curve.G.x,
this.curve.G.y));

    return {
        privateKey: privateKey.toString(),
        publicKey: publicKey.toString()
    };
}

generatePrivateKey() {
    const max = this.curve.n - 1n;
    return 1n + BigInt(Math.floor(Math.random() * Number(max)));
}

messageToBigInt(message) {
    const encoder = new TextEncoder();
    const data = encoder.encode(message);
    let result = 0n;

    for (let i = 0; i < data.length; i++) {
        result = (result << 8n) + BigInt(data[i]);
    }

    return result;
}

bigIntToMessage(bigInt) {
    if (bigInt === 0n) {
        return "";
    }

    let temp = bigInt;
    const bytes = [];

    while (temp > 0n) {
        bytes.unshift(Number(temp & 255n));
        temp = temp >> 8n;
    }

    const decoder = new TextDecoder();
    return decoder.decode(new Uint8Array(bytes));
}

modSqrt(a, p) {

```

```

    if (p % 4n === 3n) {
      const r = this.modPow(a, (p + 1n) / 4n, p);
      if (this.mod(r * r, p) === a) return r;
      return null;
    }

    return null;
  }

  modPow(base, exponent, modulus) {
    if (modulus === 1n) return 0n;
    if (exponent < 0n) {
      return this.modPow(this.modInverse(base, modulus), -exponent, modulus);
    }

    base = this.mod(base, modulus);
    let result = 1n;

    while (exponent > 0n) {
      if (exponent % 2n === 1n) {
        result = this.mod(result * base, modulus);
      }
      exponent = exponent >> 1n;
      base = this.mod(base * base, modulus);
    }

    return result;
  }

  messageToPoint(message) {
    const m = this.messageToBigInt(message);
    if ((m << 8n) >= this.curve.p) {
      throw new Error("too long message");
    }
    for (let i = 0; i < 256; i++) {
      const x = (m << 8n) | BigInt(i);
      const y_sq = this.mod(x*x*x + this.curve.a*x + this.curve.b,
this.curve.p);
      const y = this.modSqrt(y_sq, this.curve.p);
      if (y !== null) {
        return new Point(x, y);
      }
    }
    throw new Error("invalid message");
  }

  pointToMessage(point) {
    const m = point.x >> 8n;
    return this.bigIntToMessage(m);
  }

  encrypt(publicKeyStr, message) {
    const publicKey = this.parsePoint(publicKeyStr);

    if (!this.isOnCurve(publicKey)) {
      throw new Error("Public key not on curve");
    }

    const messagePoint = this.messageToPoint(message);

    const k = this.generatePrivateKey();

    //C1 = k * G

```

```

    const C1 = this.multiplyPoint(k, new Point(this.curve.G.x,
this.curve.G.y));

    //C2 = M + k * pbkey
    const kTimesPubKey = this.multiplyPoint(k, publicKey);
    const C2 = this.addPoints(messagePoint, kTimesPubKey);

    return {
      C1: C1.toString(),
      C2: C2.toString()
    };
  }

  decrypt(privateKeyStr, ciphertext) {
    const privateKey = BigInt(privateKeyStr);

    if (privateKey <= 0n || privateKey >= this.curve.n) {
      throw new Error("invalid private key");
    }

    const C1 = this.parsePoint(ciphertext.C1);
    const C2 = this.parsePoint(ciphertext.C2);

    if (!this.isOnCurve(C1) || !this.isOnCurve(C2)) {
      throw new Error("points not on curve");
    }

    //S = prkey * C1
    const S = this.multiplyPoint(privateKey, C1);

    //M = C2 - S = C2 + (-S)
    const minusS = new Point(S.x, this.mod(-S.y, this.curve.p));
    const messagePoint = this.addPoints(C2, minusS);

    return this.pointToMessage(messagePoint);
  }

  parsePoint(pointStr) {
    if (pointStr === "(null, null)") {
      return new Point(null, null);
    }

    const match = pointStr.match(/\((\d+n?), (\d+n?)\)\/);
    if (!match) {
      throw new Error('invalid point');
    }

    const x = BigInt(match[1].replace('n', ''));
    const y = BigInt(match[2].replace('n', ''));

    const point = new Point(x, y);

    if (!this.isOnCurve(point)) {
      throw new Error('point not on curve');
    }

    return point;
  }
}

export default ECEG;

```