

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 №6
на тему

ЦИФРОВАЯ ПОДПИСЬ

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 работы является изучение принципов формирования и проверки электронной цифровой подписи (ЭЦП) на основе алгоритма ГОСТ 34.10. В рамках лабораторной работы нужно реализовать программное средство, позволяющее сгенерировать ключевую пару (секретный и открытый ключи), создавать цифровую подпись для произвольного сообщения и проверять её корректность. Практическая часть включает работу с эллиптическими кривыми, хеш-функцией ГОСТ 34.11 и реализацией математических операций над точками эллиптической кривой.

2 ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Электронная цифровая подпись (ЭЦП) — это криптографический механизм, обеспечивающий проверку целостности, аутентичности и неотрекаемости цифровых данных. Отправитель с помощью закрытого ключа создает подпись, которую может проверить любой получатель, имея соответствующий открытый ключ. Стандарт ГОСТ Р 34.10 использует математический аппарат эллиптических кривых (ЕСС).

В основе алгоритма лежит использование эллиптической кривой, заданной над простым конечным полем. Кривая описывается уравнением вида: $y^2 = x^3 + a \cdot x + b \pmod{p}$.

Для работы схемы ЭЦП устанавливаются единые параметры: коэффициенты a , b , модуль p , базовая точка G и её простой порядок q . Генерация ключевой пары осуществляется следующим образом: закрытый ключ d представляет собой случайное целое число из диапазона $[1, q-1]$, а открытый ключ Q вычисляется как точка на эллиптической кривой $Q = d \cdot G$ и подлежит свободному распространению.

Процесс формирования подписи для сообщения M начинается с вычисления хеш-функции по ГОСТ Р 34.11, результат которого преобразуется в число e . Затем генерируется случайное эфемерное число k из диапазона $[1, q-1]$, и вычисляется точка $R = k \cdot G$. Первая компонента подписи r определяется как x -координата точки R по модулю q . Если $r = 0$, процесс повторяется с новым k . Вторая компонента подписи s вычисляется по формуле $s = (r \cdot d + k \cdot e) \pmod{q}$. При $s = 0$ процедура также повторяется. Итоговая подпись представляет собой пару чисел (r, s) .

Проверка подписи включает несколько этапов. Сначала осуществляется верификация принадлежности r и s диапазону $[1, q-1]$. Затем вычисляется хеш сообщения и соответствующее число e . После этого определяются вспомогательные величины: $v = e^{-1} \pmod{q}$, $z_1 = s \cdot v \pmod{q}$ и $z_2 = -r \cdot v \pmod{q}$. Вычисляется точка $R' = z_1 \cdot G + z_2 \cdot Q$. Подпись считается действительной, если x -координата точки R' по модулю q равна r .

Криптостойкость всей схемы основывается на вычислительной сложности проблемы дискретного логарифмирования в группе точек эллиптической кривой. При использовании стандартизированных 512-битных параметров практическая реализация атак на данную схему считается вычислительно неосуществимой. Этот механизм находит широкое применение в системах защищенного электронного документооборота, обеспечивая юридическую значимость и криптографическую стойкость цифровых взаимодействий.

3 ХОД РАБОТЫ

Программное средство реализовано при помощи языка программирования JavaScript. На рисунке 3.1 изображен процесс генерации ключевой пары.

🔑 Генерация ключевой пары

Сгенерируйте пару ключей: приватный для создания подписи и публичный для проверки.

СГЕНЕРИРОВАТЬ КЛЮЧИ

🔒 Приватный ключ

E35C6ACC0D1B3B437
5AEAED0F76C0428B6
6150AC3BDA1F661DA

🔓 Публичный ключ

X координата
5E442E65D752DD974EEB558EE539B7BA23541354
AB86B3228B09D640F2969DD4
08F24BA01EDB6D5E2D7E0D9C2DF20C3163F6694

Y координата
9FABA49A1CC72A73202AEC179A45F0F5B04095DC
BFC896CC99D3CE9B7DDCBE10
E568B32DE344E8D1032F3916AE85DFDB121C4A51

ПЕРЕЙТИ К ПОДПИСИ

Рисунок 3.1 – Процесс генерации ключевой пары

На рисунке 3.2 изображен процесс проверки цифровой подписи.

✔ Проверка цифровой подписи

Проверьте подлинность цифровой подписи с помощью публичного ключа.

Сообщение для проверки
шифрую

Публичный ключ для проверки

Подпись для проверки:
R: F6C7F9FEC5975020DF738C431CE26AF9... S: B5475A33091820D0188836E596A341E...

ПРОВЕРИТЬ ЦИФРОВУЮ ПОДПИСЬ

✔ Подпись верна!
Цифровая подпись успешно подтверждена. Сообщение подлинное и не было изменено.

Рисунок 3.2 – Процесс проверки цифровой подписи

Таким образом, реализованная программа успешно справляется с поставленными задачами.

ЗАКЛЮЧЕНИЕ

В ходе лабораторной работы было разработано программное средство для формирования и проверки электронной цифровой подписи на основе алгоритма ГОСТ 34.10. Реализация включала работу с эллиптическими кривыми, выполнение операций над точками и использование хеш-функции ГОСТ 34.11.

Программа успешно генерирует ключевую пару, создаёт цифровую подпись для сообщений и проверяет ее подлинность. Графический интерфейс обеспечивает удобство взаимодействия с пользователем, позволяя генерировать ключевую пару и отображать результаты работы алгоритма.

Лабораторная работа позволила закрепить теоретические знания о принципах работы ЭЦП и получить практические навыки реализации криптографических алгоритмов. Реализованное программное средство может быть использовано для демонстрации работы ГОСТ 34.10 и изучения его особенностей.

Листинг программного кода

[illegible]

```

equals(other) {
  if (this.isInfinity && other.isInfinity) return true;
  if (this.isInfinity || other.isInfinity) return false;
  return this.x === other.x && this.y === other.y;
}

add(other) {
  if (this.isInfinity) return other;
  if (other.isInfinity) return this;

  const { p, a } = CURVE_PARAMS;

  if (this.x === other.x) {
    if (this.y === other.y) {
      if (this.y === 0n) return ECPoint.INFINITY;
      const lambda = modMul(
        modMul(3n, modMul(this.x, this.x, p), p) + a,
        modInverse(modMul(2n, this.y, p), p),
        p
      );
      const x3 = mod(modMul(lambda, lambda, p) - modMul(2n, this.x,
p), p);
      const y3 = mod(modMul(lambda, mod(this.x - x3, p), p) - this.y,
p);
      return new ECPoint(x3, y3);
    } else {
      return ECPoint.INFINITY;
    }
  }

  const lambda = modMul(
    other.y - this.y,
    modInverse(mod(other.x - this.x, p), p),
    p
  );
  const x3 = mod(modMul(lambda, lambda, p) - this.x - other.x, p);
  const y3 = mod(modMul(lambda, mod(this.x - x3, p), p) - this.y, p);

  return new ECPoint(x3, y3);
}

multiply(k) {
  if (k === 0n || this.isInfinity) return ECPoint.INFINITY;

  let result = ECPoint.INFINITY;
  let addend = this;

  while (k > 0n) {
    if (k & 1n) {
      result = result.add(addend);
    }
    addend = addend.add(addend);
    k = k >> 1n;
  }

  return result;
}

const G = new ECPoint(CURVE_PARAMS.x, CURVE_PARAMS.y);

export function generateKeyPair() {
  const { q } = CURVE_PARAMS;

```



```

const privateKey = generateRandomBigInt(1n, q - 1n);

const publicKey = G.multiply(privateKey);

return {
  privateKey: privateKey.toString(16).toUpperCase(),
  publicKey: {
    x: publicKey.x.toString(16).toUpperCase(),
    y: publicKey.y.toString(16).toUpperCase()
  }
};
}

function generateRandomBigInt(min, max) {
  const range = max - min;
  const bits = range.toString(2).length;
  const bytes = Math.ceil(bits / 8);

  let randomValue = 0n;
  const randomBytes = new Uint8Array(bytes);

  do {
    crypto.getRandomValues(randomBytes);
    randomValue = 0n;
    for (let i = 0; i < bytes; i++) {
      randomValue = (randomValue << 8n) | BigInt(randomBytes[i]);
    }
  } while (randomValue > range);

  return randomValue + min;
}

export function sign(message, privateKeyHex) {
  const { q } = CURVE_PARAMS;
  const privateKey = BigInt('0x' + privateKeyHex);

  const hashHex = gost3411(message, 512);
  let e = BigInt('0x' + hashHex) % q;
  if (e === 0n) e = 1n;

  let r, s;
  do {
    const k = generateRandomBigInt(1n, q - 1n);

    const R = G.multiply(k);
    r = R.x % q;
    if (r === 0n) continue;

    s = modMul(k, e, q) + modMul(r, privateKey, q);
    s %= q;

  } while (r === 0n || s === 0n);

  return {
    r: r.toString(16).toUpperCase(),
    s: s.toString(16).toUpperCase()
  };
}

export function verify(message, signature, publicKey) {
  const { q } = CURVE_PARAMS;
  const { r, s } = signature;

```

```

const rBig = BigInt('0x' + r);
const sBig = BigInt('0x' + s);

if (rBig <= 0n || rBig >= q || sBig <= 0n || sBig >= q) {
  return false;
}

const hashHex = gost3411(message, 512);
let e = BigInt('0x' + hashHex) % q;
if (e === 0n) e = 1n;

const v = modInverse(e, q);

const z1 = modMul(sBig, v, q);
const z2 = modMul(mod(-rBig, q), v, q);

const publicKeyPoint = new ECPPoint(
  BigInt('0x' + publicKey.x),
  BigInt('0x' + publicKey.y)
);

const R = G.multiply(z1).add(publicKeyPoint.multiply(z2));

if (R.isInfinity) return false;

const R_x_mod_q = R.x % q;
return R_x_mod_q === rBig;
}

```