const crypto = require('crypto');

var BigInt = require("big-integer");

// Hàm băm SHA-512

function sha512(input) {

  const hash = crypto.createHash('sha512');

  hash.update(input);

  return hash.digest('hex');

}

// Hàm sinh số nguyên tố

function generatePrime() {

    // Tạo số nguyên tố p đủ lớn, ở đây có thể điều chỉnh giá trị ngẫu nhiên

    // Số lượng bit của p càng nhiều thì thời gian tính toán càng lớn

    var p;

    while (true) {

        p = BigInt.randBetween(BigInt("1e50"), BigInt("1e51"));

        if (p.isPrime(true)) {

            return p;

        }

    }

}

// Hàm sinh đường cong Elliptic

function generateEllipticCurvesEquation(p) {

    var a = BigInt(1);

    var b = BigInt(1);

    while (true) {

        a = BigInt.randBetween(BigInt(0), BigInt(20));

        b = BigInt.randBetween(BigInt(0), BigInt(20));

        if ((a.pow(3).multiply(4).add(b.pow(2).multiply(27))).mod(p).notEquals(0)) {

            return { a, b }

        }

    }

}

// Hàm tìm điểm sinh trên đường cong Elliptic

function generateGeneratedPoint(p, a, b) {

    var x = BigInt(0);

    while (true) {

        var squaredY = x.pow(3).add(x.multiply(a)).add(b).mod(p);

        if (isQuadraticResidue(squaredY, p)) {

            return {x: x, y: sqrtMod(squaredY, p)}

        }

        x = x.add(1);

    }

}

// Hàm kiểm tra xem a có phải là thặng dư bậc 2 trên trường hữu hạn F\_p không

function isQuadraticResidue(a, p) {

    const jacobiSymbol = a.modPow((p.minus(1)).divide(2), p);

    if (jacobiSymbol.equals(1)) {

        return true;

    }

    return false;

}

// Hàm cộng hai điểm m và n trên đường cong Elliptic

function addition(m, n, a, p) {

    var phi;

    var x1 = m.x;

    var y1 = m.y;

    var x2 = n.x;

    var y2 = n.y;

    var phi\_numerator;

    var phi\_denominator;

    if (x1.eq(x2) && y1.eq(y2)) {

        phi\_numerator = (x1.pow(2).multiply(3)).add(a);

        phi\_denominator = (y1.multiply(2));

    } else {

        phi\_numerator = y2.minus(y1);

        phi\_denominator = x2.minus(x1);

    }

    if (phi\_denominator.eq(0)) {

        return { isFinite: false }

    }

    phi = phi\_denominator.modInv(p).multiply(phi\_numerator).mod(p);

    if (phi.lt(0)) {

        phi = p.add(phi);

    }

    var x3 = phi.pow(2).minus(x1).minus(x2).mod(p);

    var y3 = phi.multiply(x1.minus(x3)).minus(y1).mod(p);

    if (x3 < 0) {

        x3 = p.add(x3);

    }

    if (y3 < 0) {

        y3 = p.add(y3);

    }

    return { x: x3, y: y3, isFinite: true }

}

// Hàm tính căn bậc 2

function sqrt(p) {

  if (p.equals(0) || p.equals(1)) {

    return p;

  }

  let x = p.divide(2);

  let y = (x.add(p.divide(x))).divide(2);

  while (y.lt(x)) {

    x = y;

    y = (x.add(p.divide(x))).divide(2);

  }

  return x;

}

// Hàm xác định khoảng trên đường cong Elliptic

function getNumberOfPoint(p) {

    var min = p.add(1).minus(sqrt(p).times(2))

    var max = p.add(1).add(sqrt(p).times(2))

    return {min, max};

}

// Hàm tính giá trị khi lấy số nguyên m nhân với điểm d thuộc đường cong Elliptic

function multiplication(m, d, a, p) {

    var dBinary = d.toString(2);

    var multi = m;

    for (i = 1; i < dBinary.length; i++) {

        multi = addition(multi, multi, a, p)

        if (dBinary[i] === '1') {

            multi = addition(multi, m, a, p);

        }

    }

    return multi;

}

// Hàm trả về phần tử đối

function opposite(m) {

    return { x: m.x, y: BigInt(0).minus(m.y) }

}

// Hàm trả về giá trị khi chuyển bản tin từ hệ chữ cái sang hệ thập phân

function messageToDecimal(mess) {

    var value = BigInt(0);

    for (var i = 0; i < mess.length; i++) {

        var number = 0;

        var charCode = mess.charCodeAt(i);

        if (charCode >= 97 && charCode <= 122) {

            number = charCode - 97;

        }

        if (charCode >= 65 && charCode <= 90) {

            number = charCode - 65;

        }

        value = value.add(BigInt(26).pow(mess.length - i - 1).multiply(number))

    }

    return value

}

// Hàm chuyển bản tin trở thành một điểm nằm trên đường cong Elliptic

function makePointForMessage(messageBinary, p, a, b) {

    while (true) {

        var x = BigInt(messageBinary + "11111", 2);

        var squareY = x.pow(3).add(x.multiply(a)).add(b).mod(p);

        if (isQuadraticResidue(squareY, p)) {

            return { messagePoint: { x: x.mod(p), y: sqrtMod(squareY, p) }, quotient: x.divide(p) }

        }

        messageBinary = messageBinary + "1";

    }

}

// Hình tính luỹ thừa theo modulo

function modPow(base, exponent, modulus) {

    if (modulus.equals(1)) return BigInt(0);

    let result = BigInt(1);

    base = base.mod(modulus);

    while (exponent.gt(0)) {

        if (exponent.mod(2).equals(1)) {

            result = result.multiply(base).mod(modulus);

        }

        exponent = exponent.divide(2);

        base = base.square().mod(modulus);

    }

    return result;

}

function legendreSymbol(a, p) {

    const ls = modPow(a, p.subtract(1).divide(2), p);

    if (ls.equals(p.subtract(1))) return -1;

    return ls;

}

// Hàm tính giá trị căn bậc 2 trong modulo p

function sqrtMod(num, p) {

    let q = p.subtract(1);

    let s = 0;

    while (q.mod(2).equals(0)) {

        q = q.divide(2);

        s++;

    }

    if (s === 1) {

        const result = modPow(num, p.add(1).divide(4), p);

        return result;

    }

    let z = BigInt(2);

    while (legendreSymbol(z, p) !== -1) {

        z = z.add(1);

    }

    let c = modPow(z, q, p);

    let r = modPow(num, q.add(1).divide(2), p);

    let t = modPow(num, q, p);

    let m = s;

    while (!t.equals(1)) {

        let i = 1;

        let temp = modPow(t, BigInt(2), p);

        while (!temp.equals(1)) {

            temp = modPow(temp, BigInt(2), p);

            i++;

        }

        const b = modPow(c, BigInt(2).pow(m - i - 1), p);

        r = r.multiply(b).mod(p);

        t = t.multiply(b.square()).mod(p);

        c = b.square().mod(p);

        m = i;

    }

    return r;

}

// Hàm chuyển xâu nhị phân sang bản tin hệ chữ cái

function binaryToMessage(messageBinaryDecrypt) {

    var decimalValue = BigInt(messageBinaryDecrypt, 2);

    const base = BigInt(26);

    let message = "";

    while (decimalValue.gt(0)) {

        const remainder = decimalValue.mod(base);

        decimalValue = decimalValue.divide(base);

        let charCode = BigInt(0);

        if (remainder.greaterOrEquals(0)) {

            charCode = remainder.plus(97);

            if (charCode.gt(122)) {

                charCode = charCode.plus(6); // Điều chỉnh giá trị charCode nếu vượt quá giá trị 122 (ký tự z)

            }

        }

        message = String.fromCharCode(charCode.toJSNumber()) + message;

    }

    return message;

}

function findMaxD(m, a, p) {

    var multi2 = m;

    count = 1;

    while (multi2.isFinite) {

        multi2 = addition(multi2, m, a, p)

        count++;

    }

    console.log(count);

}

function sleep(ms) {

    return new Promise(resolve => setTimeout(resolve, ms));

}

/\*

    Phần tiến hành chính

\*/

async function main() {

    console.log("Quy trình thiết lập sơ đồ chữ ký trên đường cong Elliptic");

    await sleep(1000);

    // Tìm số nguyên tô p đủ lớn, có độ dài khoảng 160 bit

    var p = generatePrime();

    console.log("Ta chọn p có độ dài khoảng 160 bit, và p = " + p.toString());

    var temp;

    while (true) {

        temp = BigInt.randBetween(BigInt("1e30"), BigInt("1e45"));

        if (temp.isPrime(true)) {

            break;

        }

    }

    while (true) {

        n = BigInt.randBetween(p.minus(temp), p.add(temp));

        if (n.isPrime(true)) {

            break;

        }

    }

    // Tìm các hệ số a và b cho phương trình đường cong Elliptic

    var { a, b } = generateEllipticCurvesEquation(p);

    console.log("Tìm thấy đường cong Elliptic: y^2 = x^3 + " + a.toString() + "x + " + b.toString() + " mod " + p.toString());

    await sleep(1000);

    console.log("Số điểm thuộc đường cong E là n = " + n.toString())

    await sleep(1000);

    // Tìm điểm sinh trên đường cong Elliptic

    var generatePoint = generateGeneratedPoint(p, a, b);

    console.log("Điểm sinh P của đường cong ta sẽ sử dụng: P(" + generatePoint.x.toString() + ", " + generatePoint.y.toString() + ")" );

    await sleep(1000);

    var message = "motdieumaanhdabiettruocranglachicanconkhoangcachsekhongthaynhonhaunhungmaemdaloluusoncuaemlenbuctranhanhtomauvaythigioanhmuonduocbenemthatlauboivinanglanguoilamtraitimanhthaybotdaubabyemcugiudianhkhongcandau";

    console.log("Alice có một bản tin muốn gửi với nội dung:" + message);

    await sleep(1000);

    // Tạo cặp khoá dùng trong ECDSA

    console.log("Giả sử Alice là người ký trên bản rõ M. Khi đó Alice thực hiện các bước sau để tạo ra cặp khóa công khai và khóa riêng");

    await sleep(2000);

    var d;

    d = BigInt.randBetween(1, n - 1);

    var Q = multiplication(generatePoint, d, a, p);

    console.log("1. Khoá riêng của Alice là d = " + d.toString());

    await sleep(1000);

    console.log("2. Khoá công khai của người gửi tổ hợp là (Ep(a,b), P, n, Q)");

    await sleep(3000);

    // Tạo chữ ký bằng ECDSA

    console.log("Alice sử dụng khóa riêng của mình để tạo chữ ký trên bản tin M bằng các bước sau:");

    await sleep(2000);

    console.log("1. Alice cần chọn một số nguyên k bí mật");

    await sleep(500);

    var k;

    var m1;

    var r = BigInt(0);

    var h = BigInt(0);

    var s = BigInt(0);

    while (r == 0 || s == 0) {

        k = BigInt.randBetween(BigInt("1e5"), BigInt("1e40"));

        m1 = multiplication(generatePoint, k, a, p);

        r = m1.x.mod(n);

        h = sha512(message);

        s = BigInt(h,16).add(d.multiply(r)).multiply(k.modInv(n)).mod(n)

    }

    console.log("Giá trị của k là " + k.toString());

    await sleep(1500);

    console.log("2. Tính kP = (x1 , y1), trong đó x1 là số nguyên")

    await sleep(500);

    console.log("Ta có M1 = kP = M1(" + m1.x.toString() + ", " + m1.y.toString() + ")" );

    await sleep(1000);

    console.log("3. Tính r = x1 mod n; Nếu r = 0, thì quay lại bước 1");

    console.log("Ta có r = " + r.toString());

    await sleep(1500);

    console.log("4. Tính h = H(M), trong đó H là SHA-512")

    await sleep(1000);

    console.log("Ta có h = " + h.toString());

    await sleep(1000);

    console.log("Tính s = (h + d\*r)\* (k ^(- 1)) mod n; Nếu s = 0, thì quay lại bước 1");

    await sleep(1000);

    console.log("Ta có s = " + s.toString());

    await sleep(1000);

    console.log("Chữ ký của Alice trên bản tin M là cặp số nguyên (r,s) = (" + r.toString() + "," + s.toString() + ")");

    await sleep(5000);

    // Xác thực chữ ký bằng ECDSA

    console.log("Người nhận Bob có thể xác minh tính xác thực của chữ ký của Alice là (r, s) trên bản tin M bằng cách thực hiện các bước sau:");

    await sleep(5000);

    console.log("1. Nhận được chữ ký trên Khóa công khai (E, P, n, Q) của Alice");

    await sleep(2000);

    console.log("2. Xác minh rằng các giá trị r và s nằm trong khoảng [1, n-1]");

    await sleep(2000);

    console.log("3.Tính w = s ^(- 1) mod n");

    var w = s.modInv(n);

    await sleep(2000);

    console.log("Ta có w = " + w.toString());

    await sleep(1000);

    console.log("4. Tính h = H (M), trong đó H là thuật toán băm an toàn tương tự được sử dụng bởi A");

    await sleep(2000);

    console.log("5. Tính u1 = hw mod n");

    var u1 = (BigInt(h,16).multiply(w)).mod(n);

    await sleep(1000);

    console.log("Ta có u1 = " + u1.toString());

    await sleep(1000);

    await sleep(1000);

    console.log("6. Tính u2 = rw mod n");

    var u2 = (r.multiply(w)).mod(n);

    await sleep(1000);

    console.log("Ta có u2 = " + u2.toString());

    await sleep(1000);

    await sleep(1000);

    console.log("7. Tính F = u1P + u2Q = (x0, y0)");

    var F1 = multiplication(generatePoint, u1, a, p);

    var F2 = multiplication(Q, u2, a, p);

    var F = addition(F1, F2, a, p);

    await sleep(1000);

    console.log("Ta có F = (x0, y0) = (" + F.x.toString() + "," + F.y.toString() + ")");

    await sleep(1000);

    console.log("8. Tính v = x0mod n")

    var v = F.x.mod(n);

    await sleep(1000);

    console.log("Ta có v = " + v.toString());

    await sleep(1000);

    console.log("9. Chữ ký cho tin nhắn M chỉ được xác minh nếu v = r");

    await sleep(1000);

    if (v == r) {

        console.log("Chữ ký được xác minh");

    } else {

        console.log("Xác minh không thành công!")

    }

}

main();