// Em sử dụng ngôn ngữ Javascript để thực hiện bài tập

// Phần thực hiện

var BigInt = require("big-integer")

// 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)) {

//console.log(p.toString());

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("Tiến hành quá trình thiết lập hệ mật ECC");

await sleep(2000);

console.log("Thiết lập bộ khoá công khai");

await sleep(1000);

var p = generatePrime();

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

await sleep(1000);

var {min, max} = getNumberOfPoint(p);

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

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

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);

// Phần việc của Bob

var s = BigInt.randBetween(BigInt("1e5"), BigInt("1e10"));

console.log("Bob tiến hành chọn một số nguyên s làm khoá riêng của mình");

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

await sleep(1000);

var B = multiplication(generatePoint, s, a, p);

console.log("Ta tính B = sP, khi đó B có toạ độ B(" + B.x.toString() + ", " + B.y.toString() + ")" );

await sleep(1000);

//Phần công việc của Alice

console.log("Alice tiến hành tải bộ khoá công khai");

var message = "motdieumaanhdabiettruocranglachicanconkhoangcachsekhongthaynhonhaunhungmaemdaloluusoncuaemlenbuctranhanhtomauvaythigioanhmuonduocbenemthatlauboivinanglanguoilamtraitimanhthaybotdaubabyemcugiudianhkhongcandau";

await sleep(1000);

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

var messageBinary = messageToDecimal(message).toString(2);

var messageBinaryLength = messageBinary.length;

var { messagePoint, quotient } = makePointForMessage(messageBinary, p, a, b)

console.log("sau đó tiến hành mã hoá bản tin thành một điểm trên Elliptic");

await sleep(1000);

console.log("Bản tin sau được mã hoá chính là điểm M trên đường cong, với toạ độ là M(" + messagePoint.x.toString() + ", " + messagePoint.y.toString() + ")" );

await sleep(1000);

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

await sleep(1000);

var k = BigInt.randBetween(BigInt("1e5"), BigInt("1e10"));

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

await sleep(1000);

console.log("Đến đây, ta tiến hành tạo bản mã M = (M1, M2)");

await sleep(1000);

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

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

await sleep(500);

var m2 = addition(messagePoint, multiplication(B, k, a, p), a, p);

console.log("và M2 = M + kB = M2(" + m2.x.toString() + ", " + m2.y.toString() + ")" );

await sleep(1500);

console.log("Bob nhận được bản mã M và tiến hành giải mã");

await sleep(1000);

//Bob

var messagePointDecrypt = addition(m2, opposite(multiplication(m1, s, a, p)), a, p)

console.log("Giải mã điểm M = M2 - sM1 = M(" + messagePointDecrypt.x.toString() + ", " + messagePointDecrypt.y.toString() + ")")

await sleep(1000);

var messageBinaryDecrypt = p.times(quotient).add(messagePointDecrypt.x).toString(2).slice(0, messageBinaryLength);

var messageDecrypt = binaryToMessage(messageBinaryDecrypt);

console.log("Bản tin được giải mã có nội dung như sau:" + messageDecrypt);

}

main();