**密码学原理**

**实验三：公钥加密**

学号：2022112266 姓名：魏圣卓

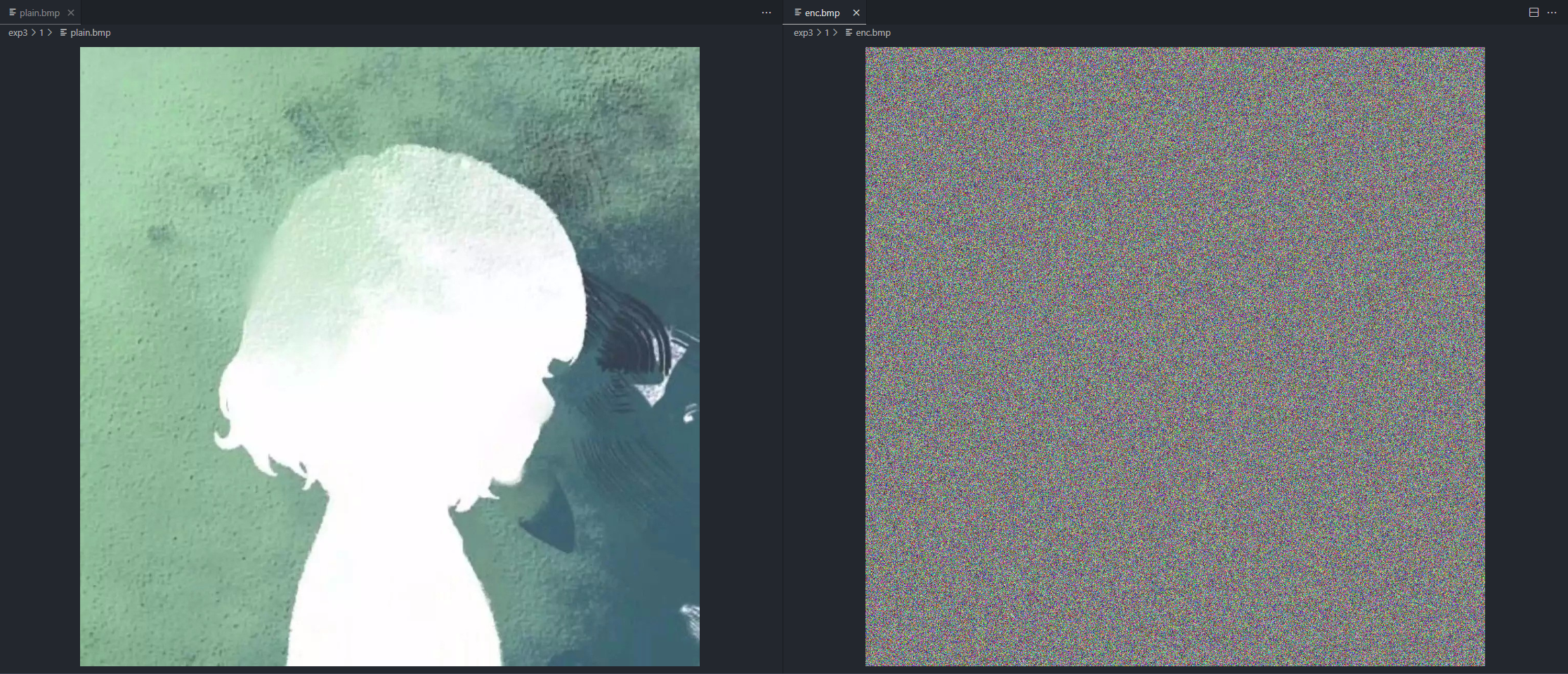
**实验目的**：本实验旨在让学生掌握运用密码学工具生成RSA密钥，进行非对称加解密，并分析相同因子RSA公钥理解RSA的安全性。

1. **公钥加密**
2. 使用密码学工具实现混合加密过程

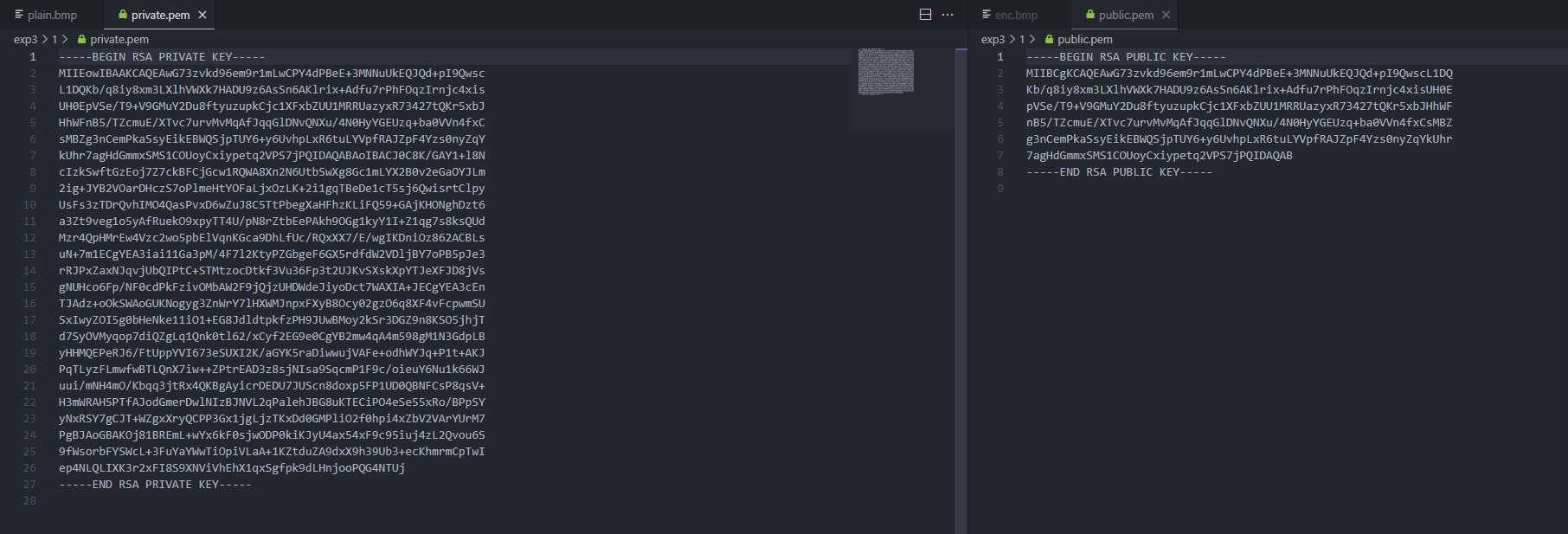
要求：生成RSA密钥对（公钥1和私钥1），用公钥1加密对称密钥并采用对称加密方法加密图片，用私钥1解密对称密钥，然后解密图片。（你可以使用实验二中的对称加密方案）

基本原理同实验2，首先随机生成AES的密钥和iv值，然后使用其对图片进行加密

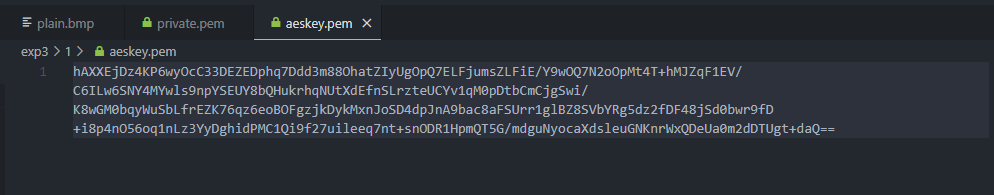
加密前后图片对比：



随机生成一对RSA密钥，并存于本地.pem文件内



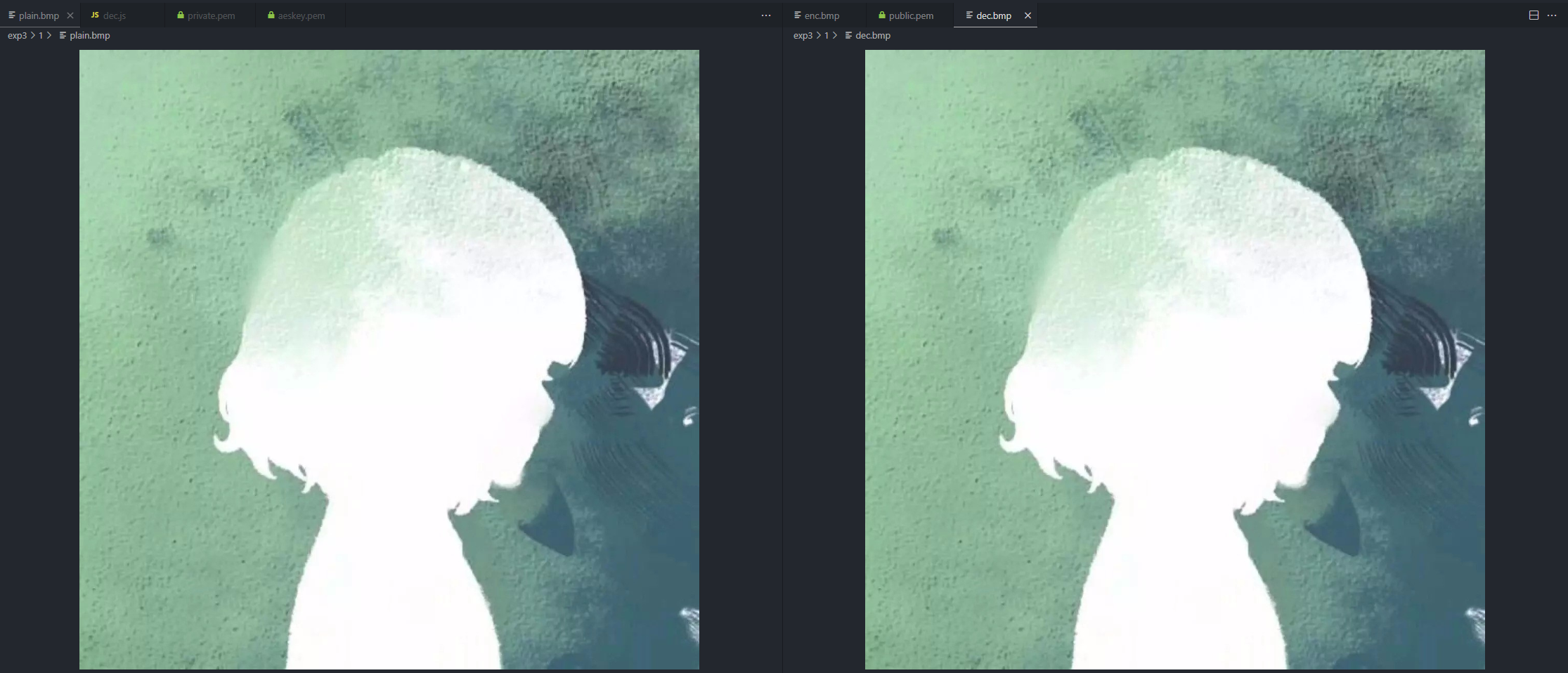
使用此RSA密钥对json化的AES密钥和iv值进行加密，并将其保存到本地aeskey.pem中



解密时，首先读取本地加密后的aeskey，然后使用私钥对其进行解密



然后再使用AES密钥和iv值对图片进行解密



成功解密出图片

1. **相同因子公钥分析**
2. 通过分析RSA公钥的因子得到私钥并破解加密信息

要求：分析附件中给出的两个公钥（公钥1、公钥2，均为pem格式）中大整数的公共因子，得到公钥1的私钥，并用私钥解密对称密钥，再解密对称加密的图片。

* 1. 对称加密方法为密钥长度128位的AES-CBC，对明文采用[PKCS #7](https://datatracker.ietf.org/doc/html/rfc2315#section-10.3)填充，128位IV放在密文开头。
  2. 对称加密的明文为RGBA四通道图像中的所有像素，为使密文图片尺寸合法，对密文进行了填充，以四字节（一个像素）为单位，与[PKCS #7](https://datatracker.ietf.org/doc/html/rfc2315#section-10.3)类似，即如果密文图像最后一个像素转换为四字节整数的值为k（大端序），说明密文图像的后k个像素是padding。由此填充方法产生的密文图片比明文图片多一行像素。
  3. 对称加密产生的密文图片为enc1.png
  4. 128位对称密钥先进行Base64编码，再使用公钥1加密，加密方法为RSA-OAEP，密文的Base64编码在下面给出。

|  |
| --- |
| **公钥1：**  -----BEGIN PUBLIC KEY-----  MIICIjANBgkqhkiG9w0BAQEFAAOCAg8AMIICCgKCAgEAuz20BUTcqVDjzEOKiJF9  66LbQB/59lnXTj/SmiD07mV1XE03BLrWfi7jFh/iq5ZPzVXfbNPjHiojO9WRhWzr  wiQGZNVZ7qFoO/PzXOT8OyHyOMcrb6ogtCyFvDOeximr3M/ICmliU2JxbLSfteZj  AplHJVgs5bJ5LTW7eSy1x2Z5aOsHjesK3rkLi1yB2jM0MeaNIB/Enb82bBMKzAam  vN6tY8bQbEoRbTnlX6PUfkU9w7XsWLMa3QbpIH9mNam1Qz4ynCjWXcDo6KzYotUf  TgGlIIOOJKsAqgOgSHqTz83e8bBizPwJg+CxBzP4Ha8C9phc41i2GiEgDf4J1J0R  0BZDcJEgZIlI+B5tlvJTy/uQyvmEP+hyMD8d83RdzLYy9h8u0MNHjJygY/Kktftp  wPtZPThpMOWWbOMM72a8Y2usz5rKTBAe+bN5QyELCErc/aQB0ABUSsNf4XxaQWbz  gJdb3hEvUkas0PfHui8UB6Yuaa7RmEE6EPIELx2WF2BGw1AG8vg5mi3I+HYxpk9W  mxy2gj63UPqr1f0u7+fnig7ANlyyPYG3LLUfhBT/d9VH0W644lqF8eZo0INEHfQf  +g4qvVVSTWfuC84ky5gTnWMbzB0iqVsZD3xw4wfSrSKyK6QFNESNdOo+1E0nz83I  cQAFD+zSSMLgodHCgA9GlGECAwEAAQ==  -----END PUBLIC KEY-----  **公钥2：**  -----BEGIN PUBLIC KEY-----  MIICIjANBgkqhkiG9w0BAQEFAAOCAg8AMIICCgKCAgEAymf92H5ljvvfTE8QjuOd  xv7YPOxXC05VceuSjtZN1aDb/4gqpWxDyMzRrPS8VRQTxkqWia4nd//zj+dheHNv  6+Emb3f00IyC2bcAFvDgQmnQB0sJZf2UI3mbMfLdnsIYW2YCbvxEiFYmUUOnh6xP  AnYFtZuvh9EDpyUwT95thQS23UEO2M1y5Q9SRUZo4EeQGb6/iqB6Q5FYabRqbsXe  Ckqxk1ENkPpuLkiQCtra++bICj4WbfVCOiiYpaN/faVud6qMHxsCxxkk+2p7kcs3  ZsCSEmLBzFNmzT32pMK9pq/rAXyXbGh4ECDuTdk1va/cCxIr5Ongven4oe4qGdnj  OCD9xPNfQZDSpYMaBcn1UveM9Rrv/GYaC9AgMnVvG5PaQOYKJzETU2gJm4rdPp/M  Hc9CvN30B6X9ewsLYIaA8ES/DRIqMG4GKAgMz0siROwLXMSkLXg1u4+mLeQzBQP5  TPJ5qwAwKJc6uPPoXo9ZmFnFW4THCoEJ+caax9M0Urg4+B6ids73C2u8A6xqVXld  ng0pAdt5exZqckhPWaWajFt4mmbUmlot7GU9PxV+NDhCn4YDmhBKQRin4lkuLilM  0/WmvnVxD7IhgXbDYrP7E1j/IO9VZQOGkntVT/BtvhJLQauF6J2bxyct9GD6Ahg4  BBKL1/FPLaDsmzWqbNiJKp8CAwEAAQ==  -----END PUBLIC KEY-----  **使用公钥1加密的对称密钥：**   |

直接解析两个公钥，得到两个n值

const bigInt1 = 763876749766176106907097075499136932542506309462790134238145662864029448357672329384240786429825506651735471293192857800292074432517560147934296565646804636440319232118174807874707278193157951022791126432548954297872603841823471708730019040711843329411852093347261870244367947957553762079472966281313126343653099184604229852078759250659887230164618217053184654632257026197939177514038547022906343467564156731888905345558825896885445137274880047279398158012899019023727313182644527939451811489338720718307850784366603399048017626264140284123069450132380165349204708385658847188179122993601433823666052262671276550074507090466095270790435243944044613783720793684122472997119531680070699442004485228493923284784588609625299335494520546908568332978745943998585734768939661601368054137698386418517184469311920103617525768647224332348763739953622815888621102511666165138133372157328365286393142987641723981258561805886361917214660712608451187202188877282119338253380439361187091882619543835302625472246720924991620467674200185807782910046428532568783970685234593602888422305491686373915683830020610512240092345516269641178633217279587340585381147938669982611180322446105897291197929327110789282688669532090634700770073422988590964225250401n;

const bigInt2 = 825745323035272798720599608361192492746787612045876953262289211169342268346322375299634388183381967644903733559912417214496145427027254956721988308512463763281120693615792222840534071393074980762077238506791126339909131208264147978200838704703070369328468495474164128065966583172696020650361090028374095147747900931623331337880180057566279781633562179594032007854587452253456011800679608763699327977686789327009650791462970241997207731238948474262065077685273924666529377565303189508779580475499013651124403952110758192421559847639791002451557155022419866908364191649354978851250736301653152786792834185002549289578088732252348814786712647056246737399818451688819799945683784702174807918167354060334981848371709009902941634174705067621057569867546563489955228609184225558415642776670302913547498803162317655424566214364429366367792388796456095209075308157031922113847324082792128272456086838179448027864861814855945170296255542403540294916048693047962499625508963680441400589569308234136277898543659042895812833949323592040527735756576476005229995515153526350952978941824427177078891317284336320446076086808680961181640941049041256254665150119759613810158261696578922846119768440364563591963843063934695815733130597665757383432415903n;

e值为固定的65573

计算两者的最大公约数，得到：

26627466049142782101189461574085020996235089156991135799522067647594275206527359658599180991364006790412470178983959891166450396562210736651239193396713286992136142461550070059217212936987738288675942968812227629223275980884750849305664154083685317995658517078413156737417252300440129121261807308751775299709307288521049604030621405834040779873490914247498118977348086366949061671661649633292359659769411644172311163846383112345844023711735501538620108619130135669124180573103605584257477958903301414926536901348873911468112406031503884991837362914821473607550589146165095676280897853974555931846786448411416266568257

计算n/gcd:

28687549478286447305146301808820842871798355347808511796763842356627593005030328848480720213117750144513421186137860559649154083515009664247554827799501024133098958404909017410028170365412343796140776819727221505068521698940404560209783113688903093022609616035265362645252705315437799400836157486001500993243670003794804588069401223165708260378509555227852581333329050238791040257668765437973719129113541232156247278324652705042833100915529598271890947796146042538417561932092028675573982547945630723568571809713726783466488292652828439877475955635530023889178260968955056364449369412700059199558943646426934407682593n

猜测认为此为公钥1的p、q值

计算私钥所需相关数值

RSA Parameters:

n:



e: 10001

d:



p: 

q:



d % (p-1):



d % (q-1):



q-1 mod p:



拼接后可得到私钥

-----BEGIN PRIVATE KEY-----



-----END PRIVATE KEY-----

解密，得到

Decrypted Text:

PGNlXnFHO3RRXVVyM1BQYQ==

Base64解密得到AES密钥：

3c63655e71473b74515d557233505061

分析密文图片，为标准png格式，可使用pngjs包处理

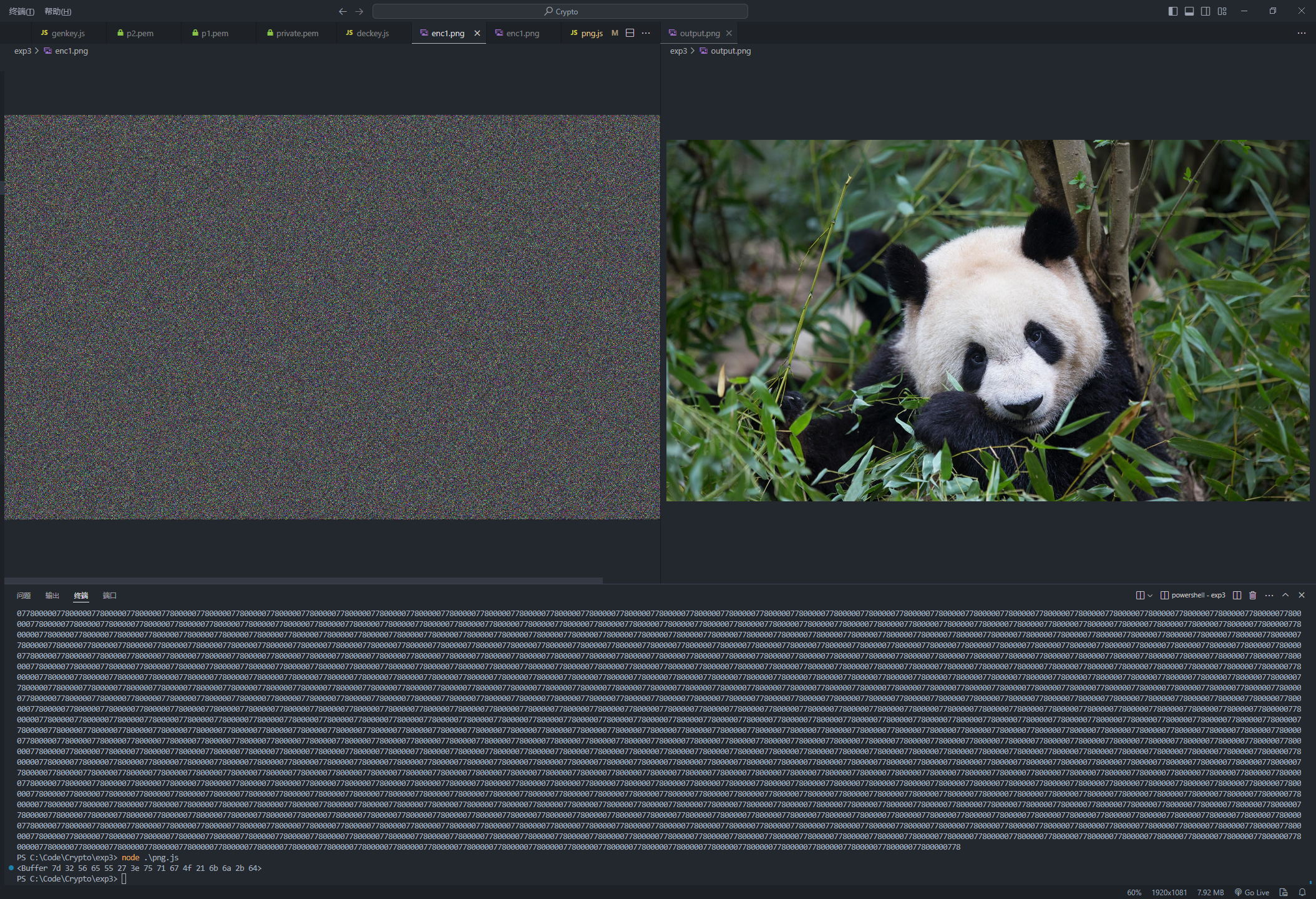
查看密文图片，大小为1920\*1081，由拼接方式可知原图片大小为1920\*1080

解析密文图片的最后一个像素，转译值为0x00000778，即填充像素为1912个像素，对应密文前128占用的1920-1912=8个像素

读取密文前8像素，转译得到128位的iv值：7d 32 56 65 55 27 3e 75 71 67 4f 21 6b 6a 2b 64

将密文去除前8位和后1912像素后进行AES解密，使用前文破解的密钥和iv值。

得到明文后，其长度对应1920\*1080的图片，将其生成新的明文png图片



Enc.js

const fs = require('fs');

const crypto = require('crypto');

const { program } = require('commander');

program

    .option('-k,--key <key>', 'Key file', 'key.json')

    .option('-i,--in <in>', 'File to enc', "plain.bmp")

    .option('-o,--out <out>', 'Output file', "enc.bmp")

    .parse(process.argv);

// 生成 RSA 密钥对

const { publicKey, privateKey } = crypto.generateKeyPairSync('rsa', {

    modulusLength: 2048, // RSA 密钥长度

    publicKeyEncoding: {

        type: 'pkcs1', // 公钥编码格式

        format: 'pem'  // 输出格式为 PEM

    },

    privateKeyEncoding: {

        type: 'pkcs1', // 私钥编码格式

        format: 'pem'  // 输出格式为 PEM

    }

});

// 将公钥保存到文件

fs.writeFileSync('public.pem', publicKey);

console.log('Public key saved to public.pem');

// 将私钥保存到文件

fs.writeFileSync('private.pem', privateKey);

console.log('Private key saved to private.pem');

function generateAESKeyAndIV() {

    const aesKey = crypto.randomBytes(16); // 16 bytes for AES-128

    const iv = crypto.randomBytes(16); // 16 bytes for AES-128

    return { aesKey, iv };

}

// 读取 BMP 文件

fs.readFile(program.opts().in, (err, data) => {

    // fs.readFile('input\_image.bmp', (err, data) => {

    if (err) {

        console.error('Error reading file:', err);

        return;

    }

    // 解析 BMP 文件

    const bmpData = parseBMP(data);

    // 提取像素数据并转换为十六进制字符串

    const pixelDataHex = extractPixelDataHex(bmpData, 32);

    // 使用 AES CBC 加密

    const encryptedData = encryptAES(pixelDataHex);

    writeEncryptedDataToFile(bmpData, encryptedData);

});

// 解析 BMP 文件

function parseBMP(data) {

    const headerSize = data.readUInt32LE(14); // 读取文件头的大小

    const imageDataOffset = data.readUInt32LE(10); // 图像数据偏移量

    // 提取图像宽度和高度

    const width = data.readUInt32LE(18);

    const height = data.readUInt32LE(22);

    console.log(width + ' ' + height);

    return {

        headerSize,

        imageDataOffset,

        width,

        height,

        headerData: data.slice(0, 54),

        imageData: data.slice(imageDataOffset)

    };

}

// 提取像素数据并转换为十六进制字符串

function extractPixelDataHex(bmpData, bitsPerPixel) {

    const bytesPerPixel = bitsPerPixel / 8;

    const imageData = bmpData.imageData;

    const pixelDataHex = [];

    for (let i = 0; i < bmpData.height; i++) {

        for (let j = 0; j < bmpData.width; j++) {

            const offset = i \* bmpData.width \* bytesPerPixel + j \* bytesPerPixel;

            const pixel = imageData.slice(offset, offset + bytesPerPixel);

            const pixelHex = pixel.toString('hex');

            pixelDataHex.push(pixelHex);

        }

    }

    console.log(pixelDataHex.length);

    return pixelDataHex;

}

function readkey(kfile) {

    const data = fs.readFileSync(kfile, 'utf8');

    const jsonData = JSON.parse(data)

    const key = Buffer.from(jsonData.key, 'hex');

    const iv = Buffer.from(jsonData.iv, 'hex');

    return { key, iv }

}

// 使用 AES CBC 加密

function encryptAES(data) {

    const { aesKey: key, iv: iv } = generateAESKeyAndIV()

    console.log(key);

    console.log(iv);

    const encryptedaes = crypto.publicEncrypt(publicKey, Buffer.from(JSON.stringify({ key, iv }), 'utf8'));

    fs.writeFileSync("aeskey.pem", encryptedaes.toString('base64'));

    const cipher = crypto.createCipheriv('aes-128-cbc', key, iv);

    let encryptedData = cipher.update(data.join(''), 'hex', 'hex');

    encryptedData += cipher.final('hex');

    return {

        key: key.toString('hex'),

        iv: iv.toString('hex'),

        encryptedData

    };

}

// 将加密后的数据写入新的 BMP 文件

function writeEncryptedDataToFile(bmpData, encryptedData) {

    const { key, iv, encryptedData: data } = encryptedData;

    const bmpHeader = bmpData.headerData.slice(0, 14);

    const bmpInfoHeader = bmpData.headerData.slice(14, 54);

    // 创建新的 BMP 文件

    const encryptedImageData = Buffer.from(data, 'hex');

    const encryptedFileData = Buffer.concat([bmpHeader, bmpInfoHeader, encryptedImageData]);

    fs.writeFile(program.opts().out, encryptedFileData, (err) => {

        if (err) {

            console.error('Error writing to file:', err);

            return;

        }

        console.log('Encrypted image file saved as :' + program.opts().out);

        console.log('Encryption Key:', key);

        console.log('Initialization Vector:', iv);

    });

}

Dec.js

const fs = require('fs');

const crypto = require('crypto');

const { program } = require('commander');

program

    .option('-k,--key <key>', 'Key file', 'key.json')

    .option('-i,--in <in>', 'File to dec', "enc.bmp")

    .option('-o,--out <out>', 'Output file', "dec.bmp")

    .parse(process.argv);

let enfile=program.opts().in

fs.readFile(enfile, (err, data) => {

    if (err) {

        console.error('Error reading file:', err);

        return;

    }

    // 解密 BMP 文件

    const decryptedData = decryptBMP(data);

    // 将解密后的数据写入新的 BMP 文件

    writeDecryptedDataToFile(decryptedData);

});

// 解密 BMP 文件

function decryptBMP(data) {

    // 从文件中提取加密数据

    const encryptedImageData = data.slice(54); // 从偏移量 54 处开始，跳过文件头和信息头

    const encryptedDataHex = encryptedImageData.toString('hex');

    const privateKey = fs.readFileSync("private.pem", 'utf8');

    const encryptedaes = fs.readFileSync("aeskey.pem", 'utf8');

    const decryptedaes = crypto.privateDecrypt(privateKey, Buffer.from(encryptedaes,"base64"));

    const aes=JSON.parse(decryptedaes)

    const key=Buffer.from(aes.key,"hex")

    const iv=Buffer.from(aes.iv,"hex")

    console.log(key);

    console.log(iv);

    // 使用 AES CBC 解密

    const decipher = crypto.createDecipheriv('aes-128-cbc', key, iv);

    let decryptedData = decipher.update(encryptedDataHex, 'hex', 'hex');

    decryptedData += decipher.final('hex');

    return {

        headData:data.slice(0,54),

        pixelData:Buffer.from(decryptedData, 'hex')

    };

}

// 将解密后的数据写入新的 BMP 文件

function writeDecryptedDataToFile(decryptedData) {

    // 读取原始 BMP 文件头和信息头

    const bmpHeader = decryptedData.headData.slice(0, 14);

    const bmpInfoHeader = decryptedData.headData.slice(14, 54);

    // 创建新的 BMP 文件

    const decryptedImageData = decryptedData.pixelData;

    const decryptedFileData = Buffer.concat([bmpHeader, bmpInfoHeader, decryptedImageData]);

    fs.writeFile(program.opts().out, decryptedFileData, (err) => {

        if (err) {

            console.error('Error writing to file:', err);

            return;

        }

        console.log('Decrypted image file saved as :'+program.opts().out);

    });

}

Gcd.js

// 定义函数来计算最大公约数

function gcd(bigInt1, bigInt2) {

    // 使用辗转相除法来计算最大公约数

    while (bigInt2 !== 0n) {

        const temp = bigInt2;

        // console.log(bigInt2);

        bigInt2 = bigInt1 % bigInt2;

        bigInt1 = temp;

    }

    return bigInt1;

}

// 两个超大整数

const bigInt1 = 763876749766176106907097075499136932542506309462790134238145662864029448357672329384240786429825506651735471293192857800292074432517560147934296565646804636440319232118174807874707278193157951022791126432548954297872603841823471708730019040711843329411852093347261870244367947957553762079472966281313126343653099184604229852078759250659887230164618217053184654632257026197939177514038547022906343467564156731888905345558825896885445137274880047279398158012899019023727313182644527939451811489338720718307850784366603399048017626264140284123069450132380165349204708385658847188179122993601433823666052262671276550074507090466095270790435243944044613783720793684122472997119531680070699442004485228493923284784588609625299335494520546908568332978745943998585734768939661601368054137698386418517184469311920103617525768647224332348763739953622815888621102511666165138133372157328365286393142987641723981258561805886361917214660712608451187202188877282119338253380439361187091882619543835302625472246720924991620467674200185807782910046428532568783970685234593602888422305491686373915683830020610512240092345516269641178633217279587340585381147938669982611180322446105897291197929327110789282688669532090634700770073422988590964225250401n;

const bigInt2 = 825745323035272798720599608361192492746787612045876953262289211169342268346322375299634388183381967644903733559912417214496145427027254956721988308512463763281120693615792222840534071393074980762077238506791126339909131208264147978200838704703070369328468495474164128065966583172696020650361090028374095147747900931623331337880180057566279781633562179594032007854587452253456011800679608763699327977686789327009650791462970241997207731238948474262065077685273924666529377565303189508779580475499013651124403952110758192421559847639791002451557155022419866908364191649354978851250736301653152786792834185002549289578088732252348814786712647056246737399818451688819799945683784702174807918167354060334981848371709009902941634174705067621057569867546563489955228609184225558415642776670302913547498803162317655424566214364429366367792388796456095209075308157031922113847324082792128272456086838179448027864861814855945170296255542403540294916048693047962499625508963680441400589569308234136277898543659042895812833949323592040527735756576476005229995515153526350952978941824427177078891317284336320446076086808680961181640941049041256254665150119759613810158261696578922846119768440364563591963843063934695815733130597665757383432415903n;

// 计算最大公约数

const result = gcd(bigInt1, bigInt2);

console.log("最大公约数:", result.toString());

console.log("n/gcd:", bigInt1/result);

gen\_key.js

function gcd(a, b) {

    if (b === 0n) {

        return a;

    } else {

        return gcd(b, a % b);

    }

}

function modInverse(a, m) {

    let m0 = m, t, q;

    let x0 = 0n, x1 = 1n;

    if (m === 1n) {

        return 0n;

    }

    while (a > 1n) {

        // q 是商，t 是余数

        q = a / m;

        t = m;

        // 使用辗转相除法计算下一次的余数

        m = a % m;

        a = t;

        t = x0;

        x0 = x1 - q \* x0;

        x1 = t;

    }

    if (x1 < 0n) {

        x1 += m0;

    }

    return x1;

}

function calculateRSAParameters(p, q, e) {

    const n = p \* q; // 计算 n

    const phi = (p - 1n) \* (q - 1n); // 计算 φ(n)

    // 计算 e 的模反演

    const d = modInverse(e, phi);

    // 计算 d mod (p-1) 和 d mod (q-1)

    const dModPMinus1 = d % (p - 1n);

    const dModQMinus1 = d % (q - 1n);

    // 计算 q-1 mod p

    const qMinus1ModP = modInverse(q,p);

    return { n, d, e, dModPMinus1, dModQMinus1, qMinus1ModP };

}

// 输入参数

const p = 26627466049142782101189461574085020996235089156991135799522067647594275206527359658599180991364006790412470178983959891166450396562210736651239193396713286992136142461550070059217212936987738288675942968812227629223275980884750849305664154083685317995658517078413156737417252300440129121261807308751775299709307288521049604030621405834040779873490914247498118977348086366949061671661649633292359659769411644172311163846383112345844023711735501538620108619130135669124180573103605584257477958903301414926536901348873911468112406031503884991837362914821473607550589146165095676280897853974555931846786448411416266568257n;

const q = 28687549478286447305146301808820842871798355347808511796763842356627593005030328848480720213117750144513421186137860559649154083515009664247554827799501024133098958404909017410028170365412343796140776819727221505068521698940404560209783113688903093022609616035265362645252705315437799400836157486001500993243670003794804588069401223165708260378509555227852581333329050238791040257668765437973719129113541232156247278324652705042833100915529598271890947796146042538417561932092028675573982547945630723568571809713726783466488292652828439877475955635530023889178260968955056364449369412700059199558943646426934407682593n;

// const p = 31518177765729290663403010315154533667310066471820536150882352253187426428247159769337479534653423738821768501345529542773928342634103326265872170108177206389414528456330175611986376442242709489900545792518147116309667540404783265952252880811482945093582072154043254087637670560024497065578691811991205720034291468011546485312464110290447442852337677201758544082252134666121558317057430879765284946151638987497632189632696466260901136009082880395233503154188421892089684949692337708651227532790549249487924345806572291605795640743410952840257681873645767819513390745479080355480421857998383133498671274254771689623513n;

// const q = 31228081429393107474561507274704265600542738839947908127871172602006351575517376477163089629958966716385768981568323503198732859203260112076326737807837839304480551464482717938445000720641960331912609673151794274982469060611550239385506555762917086451959672994083603228030150293525783304059251116128699302033650788176023836016857608350962781822536781996416270562070537788179927029515690837853666832833306484662833121172527790502589560553133726615797251656225315973013313972878173566860357351604051025661210465741817532602382196606695404526669028342131645403783017839266710361200049839548767544661001788947463593677737n;

const e = 65537n;

// 计算RSA参数

const { n, d, dModPMinus1, dModQMinus1, qMinus1ModP } = calculateRSAParameters(p, q, e);

console.log("RSA Parameters:");

console.log("n:", n.toString(16));

console.log("e:", e.toString(16));

console.log("d:", d.toString(16));

console.log("p:", p.toString(16));

console.log("q:", q.toString(16));

console.log("d % (p-1):", dModPMinus1.toString(16));

console.log("d % (q-1):", dModQMinus1.toString(16));

console.log("q-1 mod p:", qMinus1ModP.toString(16));

deckey.js

const fs = require('fs');

const crypto = require('crypto');

// 读取本地 PEM 格式的私钥

const privateKey = fs.readFileSync('private.pem', 'utf8');

console.log(privateKey);

// 待解密的文本

const encryptedText = ''; // 替换为实际的加密文本

// 使用私钥解密文本

const decryptedText = crypto.privateDecrypt(privateKey,

    Buffer.from(encryptedText, 'base64')

).toString();

console.log('Decrypted Text:');

console.log(decryptedText);

decpng.js

const fs = require('fs');

const PNG = require('pngjs').PNG;

const crypto = require('crypto');

const { program } = require('commander');

// 将数字转换为两位的十六进制字符串

function toHex(num) {

    const hex = num.toString(16);

    return hex.length === 1 ? '0' + hex : hex;

}

// 读取PNG图片并解析像素值

function parsePNG(filePath) {

    let enctext = ""

    fs.createReadStream(filePath)

        .pipe(new PNG())

        .on('parsed', function () {

            let hexString = '';

            for (let y = 0; y < this.height; y++) {

                for (let x = 0; x < this.width; x++) {

                    const idx = (this.width \* y + x) << 2;

                    const rgba = {

                        r: this.data[idx],

                        g: this.data[idx + 1],

                        b: this.data[idx + 2],

                        a: this.data[idx + 3]

                    };

                    const hex = toHex(rgba.r) + toHex(rgba.g) + toHex(rgba.b) + toHex(rgba.a);

                    hexString += hex;

                }

            }

            enctext = hexString

            // console.log(enctext);

            const ivhex = enctext.slice(0, 32)

            enctext = enctext.slice(32, -1912 \* 8)

            const keyhex = "3c63655e71473b74515d557233505061"

            const key = Buffer.from(keyhex, 'hex');

            const iv = Buffer.from(ivhex, 'hex');

            console.log(iv);

            // 使用 AES CBC 解密

            const decipher = crypto.createDecipheriv('aes-128-cbc', key, iv);

            let decrypted = decipher.update(enctext, 'hex', 'hex');

            decrypted += decipher.final('hex');

            // createnewpng(decrypted)

            // console.log(decrypted);

        });

    // return enctext

}

// 从hex字符串解析颜色值

function parseColor(hex) {

    const r = parseInt(hex.slice(0, 2), 16);

    const g = parseInt(hex.slice(2, 4), 16);

    const b = parseInt(hex.slice(4, 6), 16);

    const a = parseInt(hex.slice(6, 8), 16);

    return [r, g, b, a];

}

function createnewpng(hex) {

    // 创建1920x1080的PNG图像

    const width = 1920;

    const height = 1080;

    const img = new PNG({ width, height });

    // 从hex字符串读取像素数据并写入图像

    const hexString = hex; // 替换为实际的hex字符串

    let index = 0;

    for (let y = 0; y < height; y++) {

        for (let x = 0; x < width; x++) {

            const color = parseColor(hexString.slice(index, index + 8));

            index += 8;

            const idx = (width \* y + x) << 2;

            img.data[idx] = color[0]; // Red channel

            img.data[idx + 1] = color[1]; // Green channel

            img.data[idx + 2] = color[2]; // Blue channel

            img.data[idx + 3] = color[3]; // Alpha channel

        }

    }

    // 写入PNG图像到文件

    const outputStream = fs.createWriteStream('output.png');

    img.pack().pipe(outputStream);

    outputStream.on('finish', () => console.log('PNG image created.'));

}

// 调用函数并传入PNG图片路径

const pngFilePath = 'enc1.png';

parsePNG(pngFilePath);

// // 使用 AES CBC 解密

// const decipher = crypto.createDecipheriv('aes-128-cbc', key, iv);

// let decryptedData = decipher.update(enctext, 'hex', 'hex');

// decryptedData += decipher.final('hex');

// console.log(decryptedData);