**密码学原理**

**实验二：对称加密与认证**

学号：2022112266 姓名：魏圣卓

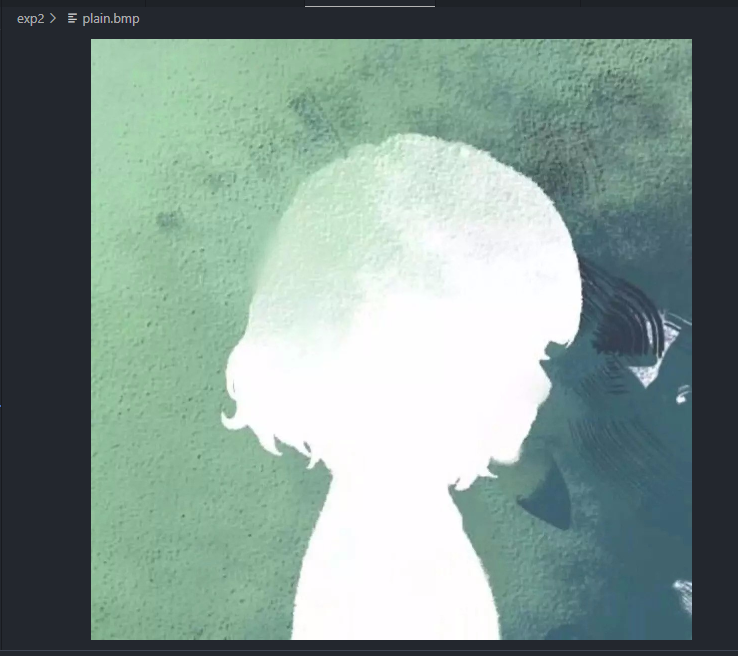
**实验目的**：本实验旨在掌握运用密码学工具实现CPA安全加密与CCA安全加密，并采用CCA能力攻击CPA安全加密方案。

1. **使用密码学工具实现CPA安全加密算法**
2. 使用密码学工具实现CPA安全加密方案

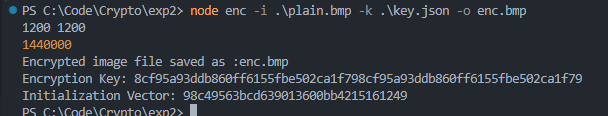
要求：选择CPA安全的加密方案对一个图片内容进行加密和解密，密文文件可用图片浏览器打开。

先随机创建AES密钥和初始向量存于本地key.json，并对图片的像素存储部分进行加密，然后将加密后的密文和原文件头打包生成新的密文图片：

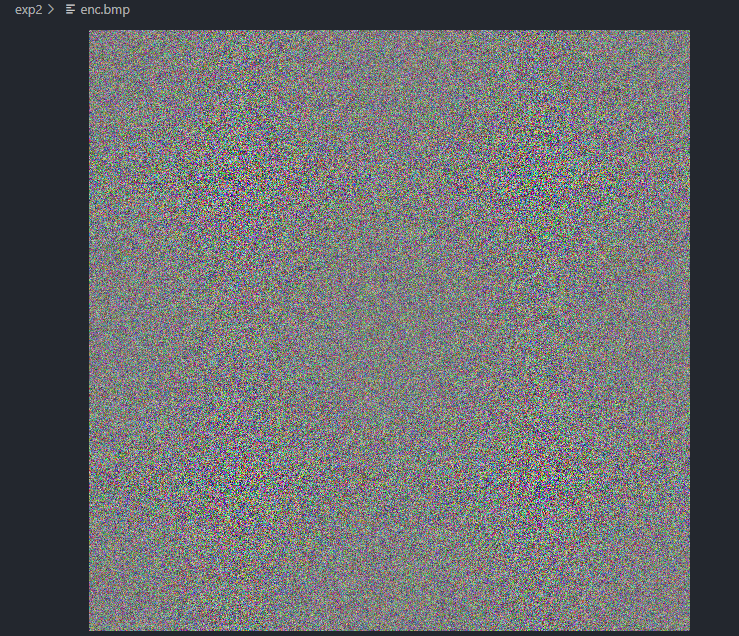
原图片：



使用AES方法对图片进行加密：



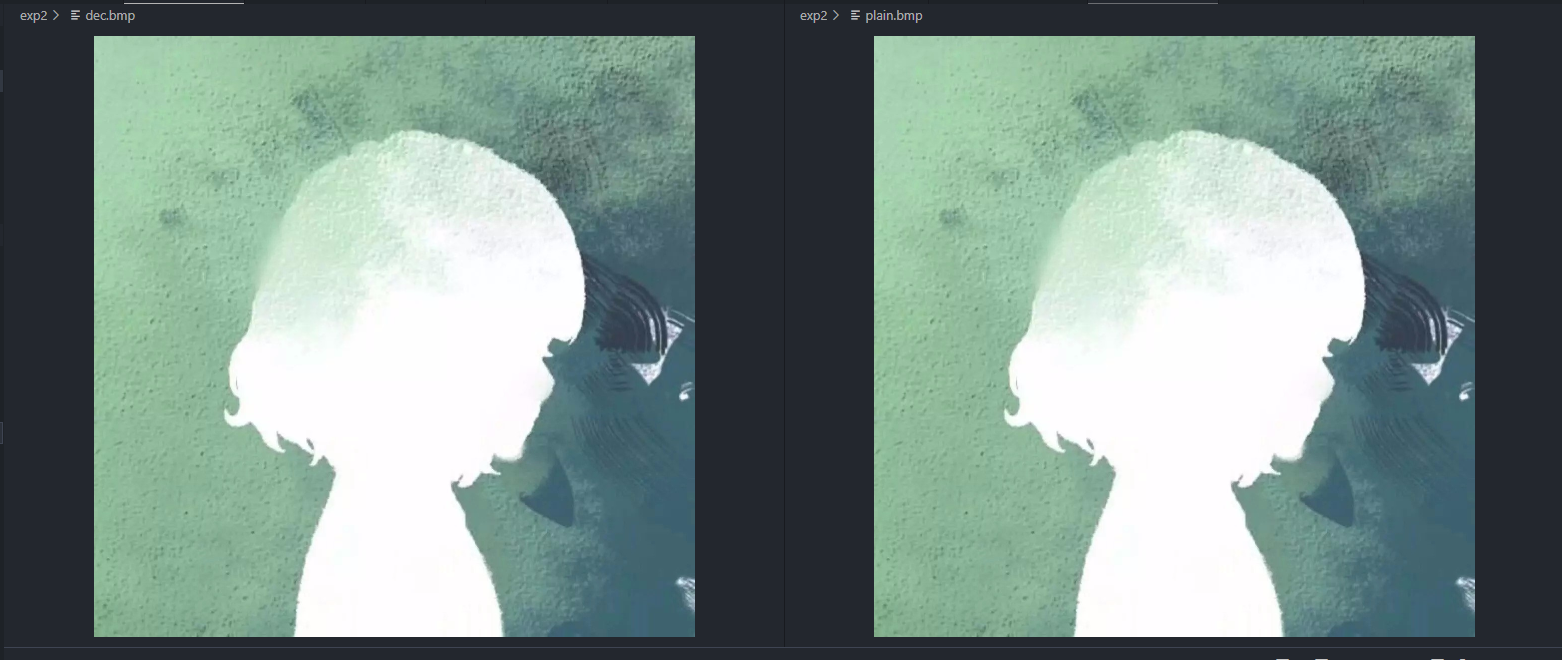
加密后的图片：



使用AES方法解密图片：



解密后图片与原图片对比：



1. **采用CCA攻击分析CPA安全加密方案**
2. 利用CCA能力敌手攻击CPA安全的加密方案

要求：对上一步中CPA安全加密方案加密的图片文件进行CCA攻击：篡改密文图片，然后用解密预言机对篡改图片解密。

使用攻击命令，随机修改密文中的部分字节：



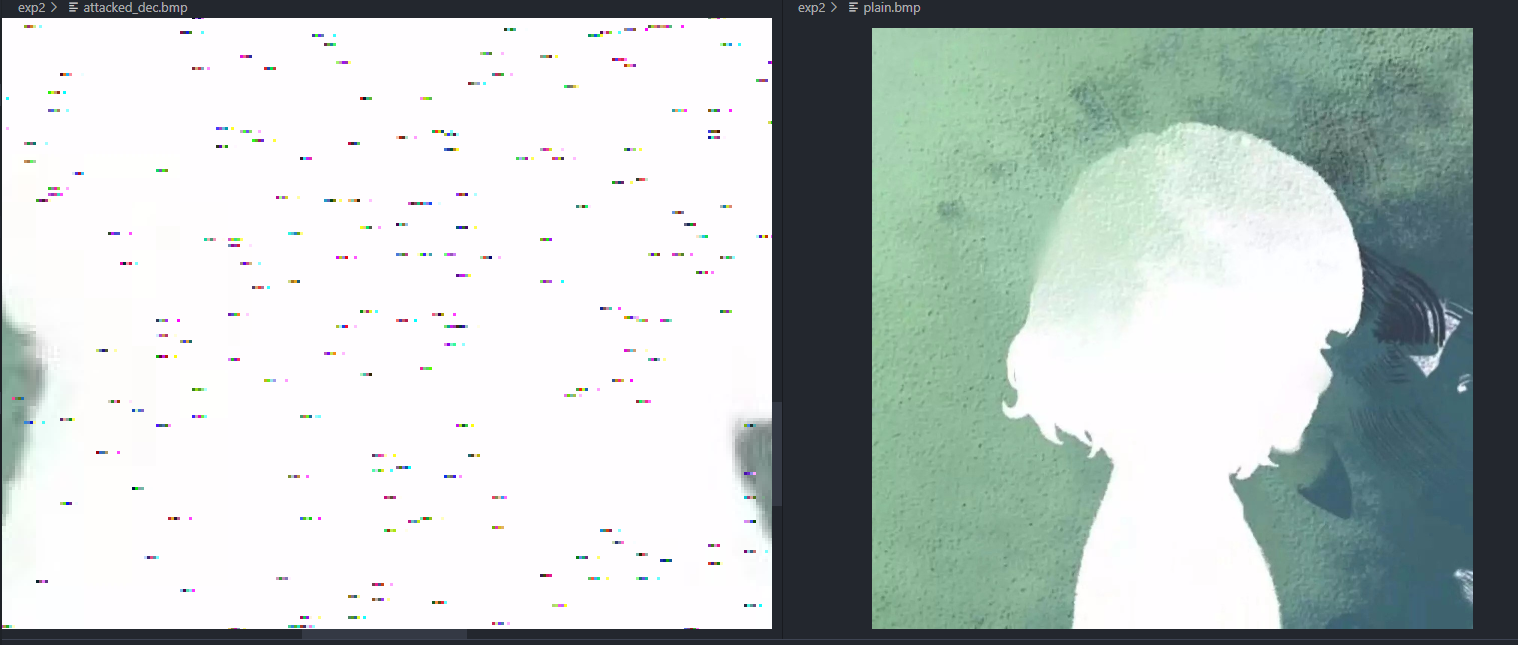
使用AES解密被攻击后的密文图片：



发现解密后的图片多出许多噪音



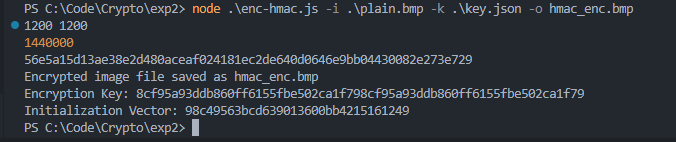
放大后可以发现部分像素点已被改变，原密文被破坏：

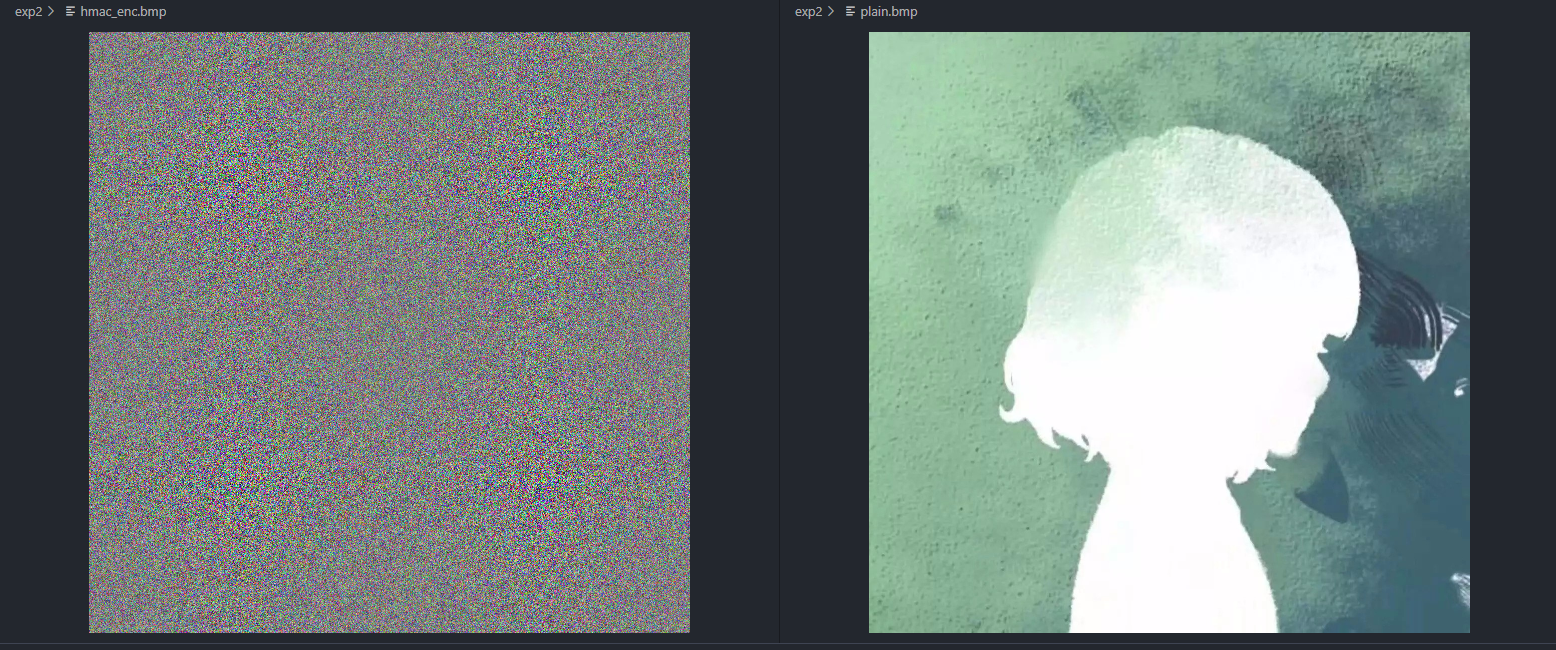


1. **使用密码学工具实现CCA安全的加密算法**
2. 使用密码学工具实现CCA安全加密方案

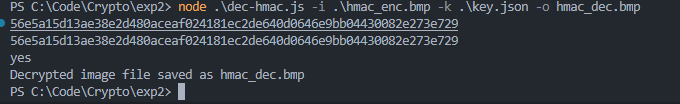
要求：选择CCA安全的加密方案对一个图片内容进行加密和解密，密文文件可用图片浏览器打开。

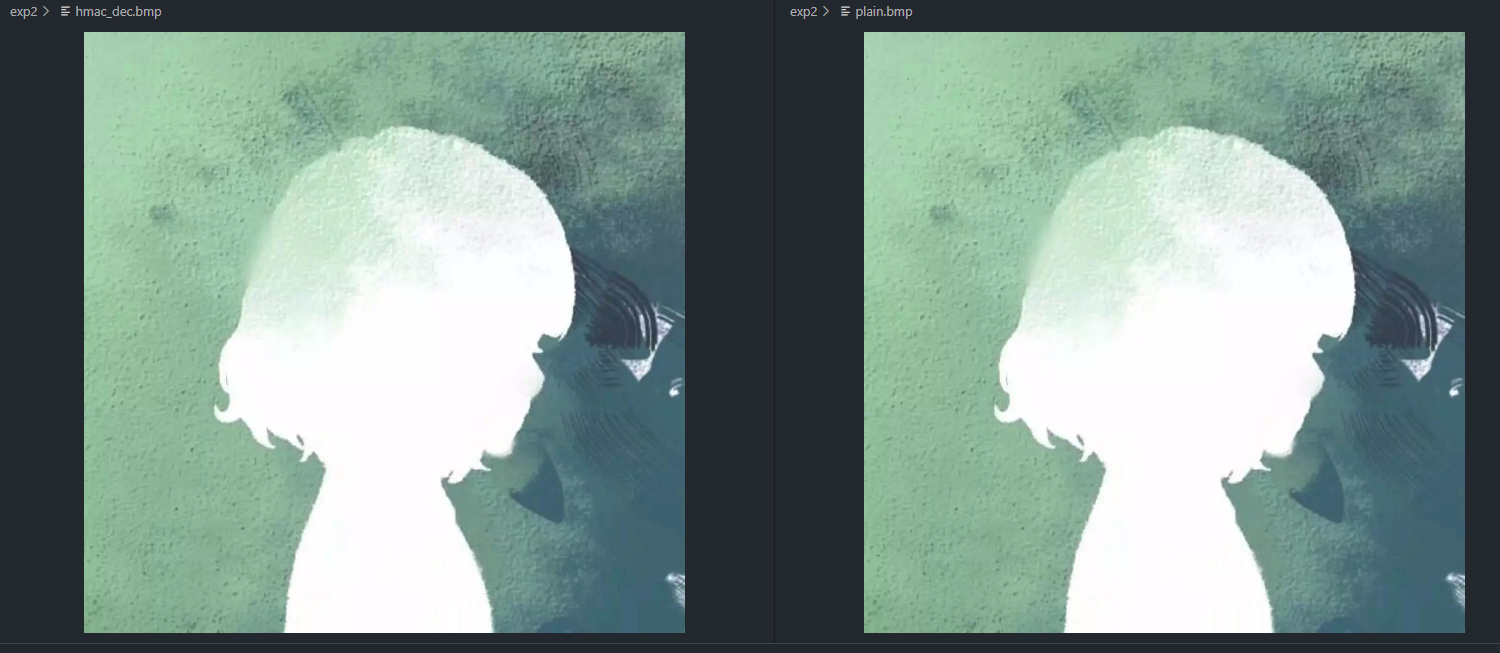
使用包含hmac标识的密文图片，加密后如图：



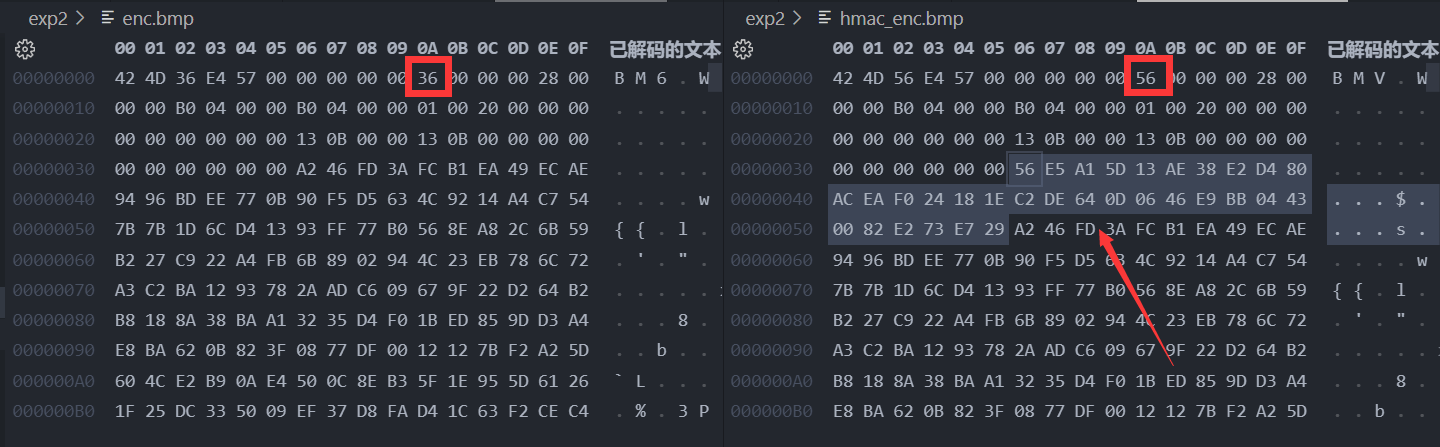


未被攻击的密文图片可以正常解密：





原理的在加密后对像素信息计算hmac值，并将32位的hmac值写入图片的文件头，同时修改offset值使图片可以被正常打开浏览。



1. CCA敌手能否攻击成功CCA安全的加密方案

要求：尝试用第2步CCA攻击来攻击CCA安全加密的图片。

使用攻击脚本篡改密文图片，随机修改一部分字节



尝试使用解密预言机，hmac对应失败，拒绝解密



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', "dec.bmp")

    .parse(process.argv);

// 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 { key, iv } = readkey(program.opts().key)

    const cipher = crypto.createCipheriv('aes-256-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', "plain.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);

});

function readkey(kfile) {

    // 读取文件

    // 同步读取文件内容

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

    // 解析JSON数据

    const jsonData = JSON.parse(data);

    // console.log(jsonData);

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

 return { key, iv }

}

// 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 { key, iv } = readkey(program.opts().key)

    // 使用 AES CBC 解密

    const decipher = crypto.createDecipheriv('aes-256-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);

    });

}

Attack.js

const fs = require('fs');

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

program

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

    .option('-o,--out <out>', 'Output file', "attacked\_image.bmp")

    .parse(process.argv);

// 读取加密的 BMP 文件

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

    if (err) {

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

        return;

    }

    // 解密 BMP 文件

    const attackedData = attackBMP(data,5000);

    // 将解密后的数据写入新的 BMP 文件

    writeattackedDataToFile(attackedData);

});

// 解密 BMP 文件

function attackBMP(data,num) {

    // 从文件中提取加密数据

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

    const oriDataHex = oriImageData.toString('hex');

    // 将十六进制字符串转换为 Buffer

    const buffer = Buffer.from(oriDataHex, 'hex');

    // 随机选择要更改的字节索引

    const bytesToChange = new Set();

    while (bytesToChange.size < num) {

        bytesToChange.add(Math.floor(Math.random() \* buffer.length));

    }

    // 随机更改选定的字节

    for (const byteIndex of bytesToChange) {

        buffer[byteIndex] = Math.floor(Math.random() \* 256); // 0 到 255 之间的随机数

    }

    return {

        headData:data.slice(0,54),

        pixelData:buffer

    };

}

// 将解密后的数据写入新的 BMP 文件

function writeattackedDataToFile(attackedData) {

    // 读取原始 BMP 文件头和信息头

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

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

    // 创建新的 BMP 文件

    const attackedImageData = attackedData.pixelData;

    const attackedFileData = Buffer.concat([bmpHeader, bmpInfoHeader, attackedImageData]);

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

        if (err) {

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

            return;

        }

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

    });

}

Enc-hmac.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', "dec.bmp")

    .parse(process.argv);

// 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); // 选择每像素 32 位的数据

// console.log(bmpData.imageData.toString('hex'));

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

    // 使用 AES CBC 加密

    const encryptedData = encryptAES(pixelDataHex);

    // console.log(encryptedData.encryptedData.toString('hex'));

    // 将加密后的数据写入新的 BMP 文件

    writeEncryptedDataToFile(bmpData,encryptedData);

});

// 解析 BMP 文件

function parseBMP(data) {

    const fileSize = data.readUInt32LE(2); // 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 {

        fileSize,

        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) {

    // 读取文件

    // 同步读取文件内容

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

    // 解析JSON数据

    const jsonData = JSON.parse(data);

    // console.log(jsonData);

    const key = Buffer.from(jsonData.key, 'hex'); // 将密钥替换为加密时使用的密钥

    const iv = Buffer.from(jsonData.iv, 'hex'); // 将初始化向量替换为加密时使用的初始化向量

  return { key, iv }

}

// 使用 AES CBC 加密

function encryptAES(data) {

 const { key, iv } = readkey(program.opts().key)

    const cipher = crypto.createCipheriv('aes-256-cbc', key, iv);

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

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

    const secretKey="12312312387867867867868678653453"

    const hmac = crypto.createHmac('sha256', secretKey);

    hmac.update(encryptedData.toString('hex'));

    const hmacResult = hmac.digest();

    return {

        key: key.toString('hex'),

        iv: iv.toString('hex'),

        hmacResult,

        encryptedData

    };

}

// 将加密后的数据写入新的 BMP 文件

function writeEncryptedDataToFile(bmpData,encryptedData) {

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

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

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

    bmpHeader.writeUInt32LE(bmpData.fileSize + 32, 2); // 文件大小

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

    bmpHeader.writeUInt32LE(54+32,10)

    const hmac=Buffer.from(hmacResult)

    console.log(hmacResult.toString('hex'));

    // 创建新的 BMP 文件

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

    const encryptedFileData = Buffer.concat([bmpHeader, bmpInfoHeader,hmac, 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-hmac.js:

const fs = require('fs');

const crypto = require('crypto');

const { error } = require('console');

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

program

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

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

    .option('-o,--out <out>', 'Output file', "decrypted\_image.bmp")

    .parse(process.argv);

// 读取加密的 BMP 文件

// let enfile='hmac\_encrypted\_image.bmp'

// let enfile = 'hmac\_attacked\_image.bmp'

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);

});

function readkey(kfile) {

    // 读取文件

    // 同步读取文件内容

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

    // 解析JSON数据

    const jsonData = JSON.parse(data);

    // console.log(jsonData);

    const key = Buffer.from(jsonData.key, 'hex'); // 将密钥替换为加密时使用的密钥

    const iv = Buffer.from(jsonData.iv, 'hex'); // 将初始化向量替换为加密时使用的初始化向量

   return { key, iv }

}

// 解密 BMP 文件

function decryptBMP(data) {

    // 从文件中提取加密数据

    const encryptedImageData = data.slice(54 + 32); // 从偏移量 54 处开始，跳过文件头和信息头

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

    // 计算HMAC

    const secretKey = "12312312387867867867868678653453"

    const hmac = crypto.createHmac('sha256', secretKey);

    hmac.update(encryptedImageData.toString('hex'));

    const hmacResult = hmac.digest('hex');

    const orihmac = data.slice(54, 54 + 32).toString('hex')

    console.log(hmacResult);

    console.log(orihmac);

    if (orihmac == hmacResult) {

        console.log('yes');

    } else {

        return {

            success:0,

            headData: null,

            pixelData: null

        };

    }

    const { key, iv } = readkey(program.opts().key)

    // 使用 AES CBC 解密

    const decipher = crypto.createDecipheriv('aes-256-cbc', key, iv);

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

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

    return {

        success:1,

        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) {

    if (decryptedData.success==0) {

        console.log('HMAC不匹配');

        console.log('解密失败');

        return

    }

    // 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);

    });

}