Izvještaj laboratorijskih vježbi

Rato Kuzmanić, 250

Vježba: 2. ECB mode

Grupa: Grupa 2

Rješenje: With the rising cost of gasoline, Chuck Norris is beginning to worry about his drinking habit.

client.js

```
const http = require('http');
const { prettyLogHex, prettyLogError, prettyLogSuccess } = require('./logger');
const { request: { get: getRequest, post: postRequest }, app } =
require('./config');
const { decryptChallenge } = require('./decrypt');

takeFirstBlockFromCiphertext = ciphertext =>
    ciphertext.slice(0, app.ciphertextBlockSize);

getNextCharacter = character =>
    String.fromCharCode(character.charCodeAt(0) + 1);

getChallenge = () =>
    new Promise((resolve, reject) => {
        const request = http.request(getRequest, response => {
            let data = '';
            response.on('data', chunk => data += chunk);
            response.on('end', () => resolve(JSON.parse(data)));
        });
        request.end();
    });

getCiphertext = plaintext =>
    new Promise((resolve, reject) => {
        const data = JSON.stringify({ plaintext });

        const request = http.request(postRequest, response => {
            response.setEncoding('utf8');

            response.on('data', data => {
                const { ciphertext } = JSON.parse(data);
                prettyLogHex(`Response for '${plaintext}'`, ciphertext);
                resolve(ciphertext);
            });

            response.on('error', error => {
                prettyLogError('Error on POST request', error);
                reject();
            });
        });
```

```
        request.write(data);
        request.end();
    });

(async () => {
    let cookie = '';

    for(let cookieCharacterCount = 0; cookieCharacterCount <
app.numberOfCookieCharacters; cookieCharacterCount++) {
        const initialPadding = 'a'.repeat((app.numberOfCookieCharacters - 1) -
cookie.length);
        const goalCiphertext = await getCiphertext(initialPadding);
        const goalBlock = takeFirstBlockFromCiphertext(goalCiphertext);

        let character = app.firstCharacterInSpace;
        for(let characterCount = 0; characterCount < app.characterIterationSpace;
characterCount++) {
            const padding = 'a'.repeat((app.numberOfCookieCharacters - 1) -
cookie.length);
            const plaintext = `${padding}${cookie}${character}`;

            const ciphertext = await getCiphertext(plaintext);
            const firstBlock = takeFirstBlockFromCiphertext(ciphertext);

            if(firstBlock === goalBlock) {
                cookie += character;
                break;
            }
            character = getNextCharacter(character);
        }
    }

    prettyLogSuccess('Cookie discovered', `The seeked cookie is "${cookie}"`);

    const challenge = await getChallenge();
    decryptChallenge(cookie, challenge)
    .then(plaintext => prettyLogSuccess('Joke decrypted', plaintext))
    .catch(error => prettyLogError('Error on joke decrypt', error));
})();


const crypto = require('crypto');
const { pbkdf2 } = require('./config');
```

decrypt.js

```
decrypt = (mode, key, iv, ciphertext) => {
    const padding = true;
    const inputEncoding = 'hex';
    const outputEncoding = 'utf8';

    const decipher = crypto.createDecipheriv(mode, key, Buffer.from(iv,
inputEncoding));
    decipher.setAutoPadding(padding);
    let plaintext = decipher.update(ciphertext, inputEncoding, outputEncoding);
    plaintext += decipher.final(outputEncoding);
    return plaintext;
}

decryptChallenge = (cookie, challenge) =>
    new Promise((resolve, reject) => {
        crypto.pbkdf2(cookie, pbkdf2.salt, pbkdf2.iterations, pbkdf2.size,
pbkdf2.hash, (error, key) =>
            error
            ? reject(`Failed to generate a key with error: ${error}`)
            : resolve(decrypt('aes-256-cbc', key, challenge.iv,
challenge.ciphertext))
        )
    });

module.exports = {
    decryptChallenge: decryptChallenge
}
```

logger.js

```
const chalk = require('chalk');

String.prototype.addWhitespacePadding = function(numberOfWhitespaces = 8) {
    return `${' '.repeat(numberOfWhitespaces)}${this}${'
'.repeat(numberOfWhitespaces)}`;
}

String.prototype.hexFormat = function () {
    return this.replace(/(.{2})/g,"$1:").slice(0, -1);
```

```
}

String.prototype.bitCount = function() {
    return this.length * 4;
}

String.prototype.byteCount = function() {
    return this.length / 2;
}

logHex = (title, string) => {
    console.log(`\n${chalk.inverse(title.addWhitespacePadding())}`);
    console.log(`String:     ${string}`);
    console.log(`Hex format: ${chalk.yellow(string.hexFormat())}`);
    console.log(`Length of hex string is ${chalk.green(`${string.length}
characters`)} equal to ${chalk.green(`${string.bitCount()} bits`)} and
${chalk.green(`${string.byteCount()} bytes`)}\n`);
}

logError = (title, error) => {
    console.log(`\n${chalk.white.bgRed(title.addWhitespacePadding())}`);
    console.log(`Details: ${error}\n`);
}

logSuccess = (title, details) => {
    console.log(`\n${chalk.black.bgGreen(title.addWhitespacePadding())}`);
    console.log(`Details: ${details}\n`);
}

module.exports = {
    prettyLogHex: logHex,
    prettyLogError: logError,
    prettyLogSuccess: logSuccess
}
```

config.js

```
const app = {
    numberOfCookieCharacters: 16,
    ciphertextBlockSize: 32,
    characterIterationSpace: 93,
    firstCharacterInSpace: "!"
};
```

```
const commonRequest = {
    host: '10.0.0.6',
    port: 80,
    headers: {
        'Content-Type': 'application/json'
    }
};

const getRequest = {
    ...commonRequest,
    path: '/ecb/challenge',
    method: 'GET'
};

const postRequest = {
    ...commonRequest,
    path: '/ecb',
    method: 'POST'
};

const pbkdf2 = {
    salt: 'salt',
    iterations: 300000,
    size: 32,
    hash: 'sha512'
};

module.exports = {
    app: app,
    request: {
        get: getRequest,
        post: postRequest
    },
    pbkdf2: pbkdf2
}
```