**Izvještaj laboratorijskih vježbi**

Rato Kuzmanić, 250

|  |  |
| --- | --- |
| **Vježba:** | 5. Asymmetric crypto: RSA signatures and DH key exchange |
| **Grupa:** | Grupa 2 |
| **Rješenje:** | Chuck Norris did in fact, build Rome in a day. |

client.js

const fs = require('fs');

const http = require('http');

const crypto = require('crypto');

const { decryptChallenge } = require('./decrypt');

const { prettyLogSuccess, prettyLogError } = require('./logger');

const { getRequest, postRequest } = require('./utils');

const { RSA, diffieHellman, getChallenge: getChallengeConfig } = require('./config');

const clientRSA = {

publicKey: fs.readFileSync('keys/public.pem'),

privateKey: fs.readFileSync('keys/private.pem')

}

const diffieHellmanService = crypto.getDiffieHellman('modp15');

diffieHellmanService.generateKeys();

const clientDiffieHellman = {

publicKey: diffieHellmanService.getPublicKey('hex')

}

getServerRSAPublicKey = () => getRequest(RSA.getServerPublicKey);

postClientRSAPublicKey = (key) => {

const data = JSON.stringify({ key });

return postRequest(data, RSA.postClientPublicKey);

}

postClientDiffieHellmanPublicKey = (key, signature) => {

const data = JSON.stringify({ key, signature });

return postRequest(data, diffieHellman.postClientPublicKey);

}

getChallenge = () => getRequest(getChallengeConfig);

digitallySignWithPrivateRSAKey = (elementToSign) => {

const sign = crypto.createSign('RSA-SHA256');

sign.write(elementToSign);

sign.end();

return sign.sign(clientRSA.privateKey, 'hex');

}

verifySignatureWithPublicRSAKey = (publicKey, signature, ...content) => {

const verify = crypto.createVerify('RSA-SHA256');

verify.write(content.join(''));

verify.end();

return verify.verify(Buffer.from(publicKey, 'hex'), signature, 'hex');

}

(async () => {

const { key: serverRSAPublicKey } = await getServerRSAPublicKey();

await postClientRSAPublicKey(clientRSA.publicKey.toString('hex'));

await postClientDiffieHellmanPublicKey(clientDiffieHellman.publicKey, digitallySignWithPrivateRSAKey(clientDiffieHellman.publicKey));

const { key, signature, challenge } = await getChallenge();

const isSignatureOk = verifySignatureWithPublicRSAKey(serverRSAPublicKey, signature, key, clientDiffieHellman.publicKey.toString('hex'));

if(isSignatureOk)

{

const sharedSecretForKeyDerivation = diffieHellmanService.computeSecret(key, 'hex');

const plaintext = await decryptChallenge(sharedSecretForKeyDerivation, challenge);

prettyLogSuccess('Joke decrypted', plaintext);

}

else

{

prettyLogError('Signature invalid', 'Challenge signature is invalid');

}

})();

utils.js

const http = require('http');

postRequest = (jsonData, config) =>

new Promise((resolve, reject) => {

const request = http.request(config, response => {

response.setEncoding('utf8');

response.on('data', data => resolve(JSON.parse(data)));

response.on('error', error => reject());

});

request.write(jsonData);

request.end();

});

getRequest = (config) =>

new Promise((resolve, reject) => {

const request = http.request(config, response => {

let data = '';

response.on('data', chunk => data += chunk);

response.on('end', () => resolve(JSON.parse(data)));

});

request.end();

});

module.exports = {

getRequest: getRequest,

postRequest: postRequest

}

decrypt.js

const crypto = require('crypto');

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

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 = (sharedDiffieHellmanKey, challenge) =>

new Promise((resolve, reject) => {

crypto.pbkdf2(sharedDiffieHellmanKey, pbkdf2.salt, pbkdf2.iterations, pbkdf2.size, pbkdf2.hash, (error, key) =>

error

? reject(`Failed to generate a key with error: ${error}`)

: resolve(decrypt('aes-256-ctr', 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)}`;

}

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 = {

prettyLogError: logError,

prettyLogSuccess: logSuccess

}

config.js

const commonRequest = {

host: '10.0.0.6',

port: 80,

headers: {

'Content-Type': 'application/json'

}

};

const getServerRSAPublicKey = {

...commonRequest,

path: '/asymm/rsa/server',

method: 'GET'

};

const postClientRSAPublicKey = {

...commonRequest,

path: '/asymm/rsa/client',

method: 'POST'

};

const postClientDiffieHellmanPublicKey = {

...commonRequest,

path: '/asymm/dh/client',

method: 'POST'

};

const getChallenge = {

...commonRequest,

path: '/asymm/challenge',

method: 'GET'

};

const pbkdf2 = {

salt: 'ServerClient',

iterations: 1,

size: 32,

hash: 'sha512'

};

module.exports = {

RSA: {

getServerPublicKey: getServerRSAPublicKey,

postClientPublicKey: postClientRSAPublicKey

},

diffieHellman: {

postClientPublicKey: postClientDiffieHellmanPublicKey

},

getChallenge: getChallenge,

pbkdf2: pbkdf2

}