Министерство образования Республики Беларусь

Учреждение образования «Белорусский государственный технологический университет»

Факультет информационных технологий

Кафедра информационных систем и технологий

**Отчет**

«Исследование алгоритмов генерации и верификации

электронной цифровой подписи»

Выполнил:

студент 3 курса 4 группы

специальности ПОИТ

Чернявский А. Л.

Минск 2020

**Цель**: изучение алгоритмов генерации и верификации электронной цифровой подписи и приобретение практических навыков их реализации

**Задачи**:

1.Закрепить теоретические знания по алгебраическому описанию и

алгоритмам реализации операций генерации и верификации электронной цифровой подписи (ЭЦП).

2. Получить навыки практической реализации методов генерации и

верификации ЭЦП на основе хеширования подписываемых сообщений и алгоритмов RSA, Эль-Гамаля и Шнорра, а также DSA.

3. Разработать приложение для реализации заданных алгоритмов генерации и верификации ЭЦП.

4. Оценить скорость генерации и верификации ЭЦП.

5. Результаты выполнения лабораторной работы оформить в виде описания разработанного приложения, методики выполнения экспериментов с использованием приложения и результатов эксперимента.

**Практическое задание**:

1. Разработать авторское оконное приложение в соответствии с целью

лабораторной работы. При этом можно воспользоваться результатами выполнения предыдущих лабораторных работ, а также доступными библиотеками либо программными кодами.

Приложение должно реализовывать следующие операции:

• генерацию и верификацию ЭЦП на основе алгоритмов RSA, ЭльГамаля и Шнорра;

• оценку времени выполнения указанных процедур при реальных (требуемых) ключевых параметрах.

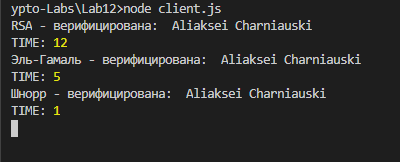
Для вычисления хешей можно также воспользоваться доступными online-средствами, например, katvin (https://katvin.com/tools/hash-generator.html).

2. Для выполнения необходимых операций передачи (по сети)/верификации информации обменяйтесь открытой ключевой информацией с получателем подписанного сообщения для каждого исследуемого алгоритма (по согласованию с преподавателем).

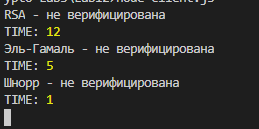
3. Результаты оформить в виде отчета по установленным правилам.

**Результат выполнения работы приложения:**

Одинаковые файлы



Добавим один пробел в один из файлов:



Листинги

Перед запуском нужно скачать необходимые пакеты с помощью команды “npm install”

RSA

const crypto = require('crypto');

let prKey, pubKey, s;

class ServerSignRSA{

    constructor(){

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

            modulusLength: 2048,

            publicKeyEncoding: {type: 'pkcs1', format: 'pem'},

            privateKeyEncoding: {type: 'pkcs1', format: 'pem'}

        });

        prKey = privateKey;

        pubKey = publicKey;

        s = crypto.createSign('SHA256');

    }

    getSignContext(rs,cb){

        rs.pipe(s);

        rs.on('end', () => {

            cb({

                signature: s.sign(prKey).toString('hex'),

                publicKey: pubKey.toString('hex')

            });

        });

    };

}

// Client Verification

let v;

class ClientVerifyRSA{

    constructor(){

        v = crypto.createVerify('SHA256')

    }

    verify(signContext, rs, cb){

        rs.pipe(v);

        rs.on('end', () => {

            cb(v.verify(signContext.publicKey, signContext.signature, 'hex'));

        });

    };

}

module.exports.ServerSignRSA = ServerSignRSA;

module.exports.ClientVerifyRSA = ClientVerifyRSA;

Shnorr

let crypto = require('crypto');

let prKey, pubKey, s;

class ServerSignShnorr {

    constructor() {

        const { privateKey, publicKey } = crypto.generateKeyPairSync('ed25519', {

            modulusLength: 2048,

            publicKeyEncoding: { type: 'spki', format: 'pem' },

            privateKeyEncoding: { type: 'pkcs8', format: 'pem' }

        });

        prKey = privateKey;

        pubKey = publicKey;

    }

    getSignContext(rs) {

        return {

            signature: crypto.sign(null, Buffer.from(rs), prKey),

            publicKey: pubKey.toString('hex')

        };

    };

}

let v;

class ClientVerifyShnorr {

    constructor() {

    }

    verify(signContext, rs) {

        return crypto.verify(null, Buffer.from(rs), signContext.publicKey, Buffer.from(signContext.signature));

    };

}

module.exports.ServerSignShnorr = ServerSignShnorr;

module.exports.ClientVerifyShnorr = ClientVerifyShnorr;

ElGamal

const crypto = require('crypto');

let prKey, pubKey, s;

class ServerSignElgam{

    constructor(){

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

            modulusLength: 2048,

            publicKeyEncoding: {type: 'spki', format: 'pem'},

            privateKeyEncoding: {type: 'pkcs8', format: 'pem'}

        });

        prKey = privateKey;

        pubKey = publicKey;

        s = crypto.createSign('SHA256');

    }

    getSignContext(rs,cb){

        rs.pipe(s);

        rs.on('end', () => {

            cb({

                signature: s.sign(prKey).toString('hex'),

                publicKey: pubKey.toString('hex')

            });

        });

    };

}

// Clint Verification

let v;

class ClientVerifyElgam{

    constructor(){

        v = crypto.createVerify('SHA256')

    }

    verify(signContext, rs, cb){

        rs.pipe(v);

        rs.on('end', () => {

            cb(v.verify(signContext.publicKey, signContext.signature, 'hex'));

        });

    };

}

module.exports.ServerSignElgam = ServerSignElgam;

module.exports.ClientVerifyElgam = ClientVerifyElgam;