Міністерство освіти і науки України
Національний технічний університет України
«Київський політехнічний інститут ім. Ігоря Сікорського»
Факультет інформатики та обчислювальної техніки
Кафедра обчислювальної техніки

ЛАБОРАТОРНА РОБОТА №3

З ДИСЦИПЛІНИ

“ Протоколи і алгоритми електронного голосування”

Виконав:
студент групи ІВ-92
Карпека Д. Ю.

Перевірив:
Нестерук А. О.

м. Київ – 2022 р.

Тема:
Протокол Е-голосування з двома виборчими дільницями

Мета роботи:
Дослідити протокол Е-голосування з двома виборчими дільницями

Хід роботи:

Основні класи:

Voter

```
class Voter {
    readonly name: string;
    readonly gen: number;
    readonly privateKey: crypto.KeyObject
    randomID: number;
    #channel: Channel;
    #message: Message;
    constructor(name: string, message: string, gen: number) {
        this.name = name;
        this.gen = gen;
        this.#channel = elgamal.ElGamal(elgamal.Alphabet, this.gen);
        this.randomID = 0;
        const { publicKey, privateKey } = crypto.generateKeyPairSync("rsa", {
            // The standard secure default length for RSA keys is 2048 bits
            modulusLength: 2048,
          });
        this.privateKey = privateKey;
        this.#message = {
            ID: 0,
            randomID: 0,
            buileten: message,
            signature: Buffer.from(""),
            publicKey: publicKey
        }
    }

    get voterGen() {
        return this.gen;
    }

    cipherMessage(m: string, pubKey: number[]) {
        let ciphered: string = this.#channel.encrypt(m, pubKey);
        return ciphered;
    }

    receiveRandomID(id: number) {
```

```
        this.randomID = id;
    }

    signBuileten(buileten: string) {
        const verifiableData = buileten;
        const signature = crypto.sign("sha256", Buffer.from(verifiableData), {
            key: this.privateKey,
            padding: crypto.constants.RSA_PKCS1_PSS_PADDING,
          });

        return signature;
    }

    createMessage(pubKey: number[]) {
        this.#message.ID = crypto.randomInt(100000, 1000000);
        this.#message.randomID = this.randomID;
        this.#message.buileten = this.cipherMessage(this.#message.buileten,
pubKey);
        this.#message.signature = this.signBuileten(this.#message.buileten);
        return this.#message;
    }
}
```

Buro

```
class Buro {
    #register: BuroArtifact[];
    #statistic: Statistic;
    constructor() {
        this.#register = [];
        this.#statistic = {
            "Second voting": 0,
            "Signature violation": 0,
            "Attempt to vote without proper registration": 0,
            "Not unique ID": 0
        }
    }

    createPublicKey(gen: number) {
        let newChannel: Channel = elgamal.ElGamal(elgamal.Alphabet, gen);
        this.#register.push(
            {
                name: undefined,
                registrationNumber: undefined,
                channel: newChannel
            }
        )
        return newChannel.pubKey;
```

```
    }

    receiveCipheredMessage(m: string) {
        // decipher message using last channel of communication
        let lastChannel: Channel = this.#register.slice(-1)[0].channel;
        let decipheredMessage: string = lastChannel.decrypt(m);
        if (this.#register.find((buroArt: BuroArtifact) => {
            return buroArt.name === decipheredMessage;
        })) {
            console.log("Attempt to vote second time");
            this.#statistic["Second voting"]++;
            return 0;
        } else {
            let newArtifact: BuroArtifact = {
                name: decipheredMessage,
                registrationNumber: crypto.randomInt(100000, 1000000000000),
                channel: lastChannel
            }
            this.#register[this.#register.length - 1] = newArtifact;
            if (newArtifact.registrationNumber) {
                return newArtifact.registrationNumber
            } else {
                return 0;
            }

        }
    }

    get ListOfID() {
        let listOfID: number[] = [];
        this.#register.forEach((buroArt) => {
            let ID  = buroArt.registrationNumber;
            if (ID) {
                listOfID.push(ID);
            }

        })
        return listOfID;
    }

    get statistic() {
        return this.#statistic;
    }
}
```

CVK

```
class CVK {
    listOfID: number[];
    #listOfChannels: Channel[];
    #statistic: Statistic;
    #voting: Candidates;
    #finalList: finalVote[];
    constructor(listOfIDs: number[], statistic: Statistic, candidateNames:
string[]) {
        this.listOfID = listOfIDs;
        this.#listOfChannels = [];
        this.#statistic = statistic;
        this.#voting = {};
        candidateNames.map((candidateName) => this.#voting[candidateName] = 0)
        this.#finalList = [];
    }

    createPublicKey(gen: number) {
        let newChannel: Channel = elgamal.ElGamal(elgamal.Alphabet, gen);
        this.#listOfChannels.push(newChannel);
        return newChannel.pubKey;
    }

    verification(message: Message) {
        // check signature for violation
        const isVerified: boolean = crypto.verify(
            "sha256",
            Buffer.from(message.buileten),
            {
              key: message.publicKey,
              padding: crypto.constants.RSA_PKCS1_PSS_PADDING,
            },
            message.signature
          );
        return isVerified;
    }

    checkForUniqueID(message: Message) {
        let uniqueness = this.listOfID.includes(message.randomID);

        // console.log("this id is unique: ", uniqueness);
        if (!uniqueness) {
            console.log("Attempt to vote without proper registration");
            this.#statistic["Attempt to vote without proper registration"]++;
        } else {
            const index: number = this.listOfID.indexOf(message.randomID);
            this.listOfID.splice(index, 1);
```

```
        }

        return uniqueness;
    }

    receiveCipheredBuileten(message: Message) {
        let verified: boolean = this.verification(message);
        let unique: boolean = this.checkForUniqueID(message);
        if (verified) {
            if (unique) {
                let lastChannel: Channel = this.#listOfChannels.slice(-1)[0];
                let decipheredMessage: string =
lastChannel.decrypt(message.buileten);
                this.#finalList.push({
                    id: message.ID,
                    buileten: decipheredMessage
                });
                this.#voting[decipheredMessage]++;
            } else {
                this.#statistic["Not unique ID"]++;
            }
        } else {
            this.#statistic["Signature violation"]++;
        }
    }

    retrieveResults() {
        Object.entries(this.#voting).forEach(([candidate, votes]) => {
            console.log(`${candidate} has scored ${votes} votes`);
        })
        let votingSorted = Object.entries(this.#voting).sort(([name1, vote1],
[name2, vote2]) => vote2 - vote1);
        console.log("The winner is ", votingSorted[0][0], " with votes of ",
votingSorted[0][1]);

        console.log("Table of participants: ");
        this.#finalList.forEach((finalVote) => {
            console.log("Participant with ID", finalVote.id, "voted for",
finalVote.buileten);
        })

        console.log("Violations fixed during voting process:")
        Object.entries(this.#statistic).forEach(([name, violations]) => {
            console.log(name, ":", violations);
        })
    }
}
```

Main():

```
function main() {
    // initialize Buro and Voters
    let testBuro: Buro = new Buro();
    let testVoters: Voter[] = [
        new Voter("Sashko", "Option1", 2),
        new Voter("Sashko", "Option2", 3), // second time voting
        new Voter("Daryna", "Option1", 5),
        new Voter("Alex", "Option2", 10),
        new Voter("Antin", "Option1", 17),
        new Voter("Dmytrii", "Option2", 19),
        new Voter("Olena", "Option2", 23),
        new Voter("Artem", "Option2", 61),
        new Voter("Lidiia", "Option1", 31),
        new Voter("Sophia", "Option2", 37)
    ];

    testVoters.forEach((testVoter) => {
        // create gen in voter
        let voterGen: number = testVoter.voterGen;
        // give gen to BR to create public key
        let pubKey: number[] = testBuro.createPublicKey(voterGen);
        // return public key from BR to Voter to cipher
        let cipheredMessage = testVoter.cipherMessage(testVoter.name, pubKey);
        // console.log(typeof cipheredMessage);
        // give ciphered message to Buro
        let lastRegistrationNumber: number =
testBuro.receiveCipheredMessage(cipheredMessage);
        // console.log(lastRegistrationNumber);
        // voter gets his generated ID for voting
        testVoter.receiveRandomID(lastRegistrationNumber);
    });

    // create CVK with list of ID from Buro and their statistic of violations
    let testCVK: CVK = new CVK(testBuro.ListOfID, testBuro.statistic, ["Option1",
"Option2"]);

    testVoters.forEach((testVoter) => {
        // create CVK public key to cipher
        let pubKey = testCVK.createPublicKey(testVoter.gen);
        // voter creates message (buileten) for CVK
        let cipheredMessage = testVoter.createMessage(pubKey);
        // CVK deciphers voter's message and their buileten
        testCVK.receiveCipheredBuileten(cipheredMessage);
    });

    testCVK.retrieveResults();
```

```
}

main();
```

Детальніше ознайомитись із проєктом лабораторної можна за цим посиланням: https://github.com/dmytrii-karpeka/labs-from-PAEV

Результат:

```
PS D:\Studying\7\ПАЕГ\lw1\lw3> npx tsx .\start.ts
Attempt to vote second time
Attempt to vote without proper registration
Option1 has scored 4 votes
Option2 has scored 5 votes
The winner is  Option2  with votes of  5
Table of participants:
Participant with ID 814011 voted for Option1
Participant with ID 216467 voted for Option1
Participant with ID 575426 voted for Option2
Participant with ID 748597 voted for Option1
Participant with ID 144592 voted for Option2
Participant with ID 732652 voted for Option2
Participant with ID 154738 voted for Option2
Participant with ID 589015 voted for Option1
Participant with ID 277608 voted for Option2
Violations fixed during voting process:
Second voting : 1
Signature violation : 0
Attempt to vote without proper registration : 1
Not unique ID : 1
```

Дослідження протоколу:

1. Перевірити чи можуть голосувати ті, хто не має на це права.

   Ні, для голосування потрібно пройти попередню реєстрацію у РБ або ж надзвичайною удачею вгадати ID виборців, які вже проголосували

2. Перевірити чи може виборець голосувати кілька разів.

   Ні, це фіксується як при реєстрації так і на самому голосуванні

3. Чи може хтось (інший виборець, ЦВК, стороння людина) дізнатися за кого проголосували інші виборці?

Не можуть, якщо не знають чиє ім’я стоїть за ID. Інший виборець не знає хто і як проголосував, окрім себе. ЦВК не може пов’язати ім’я виборця із ID, тому також не може. РБ знає імена виборців і ID, які вона видає виборцям, тому може їх пов’язати. При змові ЦВК і РБ, ЦВК зрештою може дізнатись хто за кого проголосував, адже матиме доступ до зв’язку «ім’я – ID».

4. Перевірити чи може інший виборець чи стороння людина проголосувати замість іншого зареєстрованого виборця.

   Може, якщо відгадає виданий ID.

5. Чи може хтось (інший виборець, ЦВК, стороння людина) таємно змінити голос в бюлетені?

   Не може ніхто, адже при такому шахрайстві виборець побачить результат у фінальній таблиці виборців. Він пам’ятає свій ID і голос.

6. Чи може виборець перевірити, що його голос врахований при підведенні кінцевих результатів?

   Так, може. У фінальній таблиці голосування це відображено.

Висновок

У лабораторній роботі було використано метод шифрування El Gamal із ЕЦП у вигляді DSA. Також новим елементом стало реєстраційне бюро, яке дозволило безпечно обійти поширені проблеми попередніх протоколів.