Міністерство освіти і науки України

Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського"

Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

Звіт

З лабораторної роботи № 4 з дисципліни

«Протоколи й алгоритми електронного голосування»

“**Протокол Е-голосування з перемішуванням**”

Виконав(ла) *ІП-13 Бабіч Денис* _____

(шифр, прізвище, ім'я, по батькові)

Перевірив *Нестерук А. О.* _____

(посада, прізвище, ім'я, по батькові)

Київ 2024

ЛАБОРАТОРНА РОБОТА № 4

Тема роботи: Протокол Е-голосування з перемішуванням.

Мета роботи: Дослідити протокол Е-голосування з перемішуванням.

Основне завдання:

Змоделювати протокол Е-голосування з перемішуванням будь-якою мовою програмування та провести його дослідження. Для кодування повідомлень використовувати метод RSA, для реалізації ЕЦП використовувати алгоритм Ель-Гамаля.

Умови: В процесі голосування повинні приймати участь не менше 2 кандидатів та не менше 4 виборців. Повинні бути реалізовані сценарії поведінки на випадок порушення протоколу (виборець не проголосував, проголосував неправильно, виборець не має права голосувати, виборець хоче проголосувати повторно, виборець хоче проголосувати замість іншого виборця та інші).

На основі змодельованого протоколу провести його дослідження (Аналіз повинен бути розгорнутим та враховувати всі можливі сценарії подій під час роботи протоколу голосування):

1. Перевірити чи можуть голосувати ті, хто не має на це права.
2. Перевірити чи може виборець голосувати кілька разів.
3. Чи може хтось (інший виборець, ВК, стороння людина) дізнатися за кого проголосували інші виборці?
4. Перевірити чи може інший виборець чи стороння людина проголосувати замість іншого зареєстрованого виборця.
5. Чи може хтось (інший виборець, ВК, стороння людина) таємно змінити голос в бюлетені?
6. Чи може виборець перевірити, що його голос врахований при підведенні кінцевих підсумків?

Виконання завдання:

Для виконання завдання було реалізовано клас VoterController, який зберігає всю логіку голосування з точки зору виборця. Він має методи для генерації пар rsa-ключів, голосування за кандидата, розшифрування зовнішнього шару, внутрішнього, верифікації підпису.

Цікавою деталлю реалізації є створення пар ключів rsa динамічно, на основі розрахунку теоретичного необхідного розміру ключа у бітах. Це пов’язано з необхідністю шифрувати великий об’єм даних у байтах, а публічний ключ rsa має обмеження на довжину повідомлення, яке може бути зашифровано.

Код класу наведено на рисунка 1.1 – 1.4.

```
class VoterController:

    _BITS_IN_BYTE_COUNT: int = 8
    _OFFSET_RSA_SIZE_BYTES: int = 11
    _BUFFER_NOISE_SIZE_BYTES: int = 5
    _ELGAMAL_SIGNATURE_PAYLOAD_LENGTH = 32
    _BUFFER_NOISE_CONTENT: str = string.ascii_letters + string.digits
    _OFFSET_RSA_SIZE_BITS: int = _OFFSET_RSA_SIZE_BYTES * _BITS_IN_BYTE_COUNT
    _BUFFER_NOISE_SIZE_BITS: int = _BUFFER_NOISE_SIZE_BYTES * _BITS_IN_BYTE_COUNT
    _BUFFER_PAYLOAD_SIZE_BITS = len(Candidate._next_id.to_bytes()) * _BITS_IN_BYTE_COUNT

    _next_id: int = 0
    _voters: list['VoterController'] = list()
    _shared_rsa_internal_public_keys: list[rsa.PublicKey] = list()
    _shared_rsa_external_public_keys: list[rsa.PublicKey] = list()

    def __init__(self) -> 'VoterController':
        VoterController._next_id += 1
        self._id = VoterController._next_id
        self._dumps: list[bytes] = list()
        VoterController._voters.append(self)
        self._elgamal_private_key, self._elgamal_public_key = elgamal.Elgamal.newkeys(VoterController._ELGAMAL_SIGNATURE_PAYLOAD_LENGTH)

    def get_id(self) -> int:
        return self._custom_id

    def get_rsa_public_key(self) -> rsa.PublicKey:
        return self._rsa_public_key

    def get_elgamal_public_key(self) -> elgamal.PublicKey:
        return self._elgamal_public_key
```

Рисунок 1.1 – Код класу VoterController

```
def generate_rsa_keys_pairs(self) -> None:
    reversed_id: int = VoterController._next_id - self._id + 1

    internal_keys_size_bits = VoterController._BUFFER_PAYLOAD_SIZE_BITS + VoterController._BUFFER_NOISE_SIZE_BITS + (VoterController._OFFSET_RSA_SIZE_BITS * reversed_id)
    (self._rsa_internal_public_key, self._rsa_internal_private_key) = rsa.newkeys(nbits = internal_keys_size_bits)    "newkeys": Unknown word.
    VoterController._shared_rsa_internal_public_keys.insert(0, self._rsa_internal_public_key)

    largest_keys_size_bits = VoterController._BUFFER_PAYLOAD_SIZE_BITS + VoterController._BUFFER_NOISE_SIZE_BITS + (VoterController._OFFSET_RSA_SIZE_BITS * VoterController.
    _next_id)
    external_keys_size_bits = largest_keys_size_bits + (VoterController._BUFFER_NOISE_SIZE_BITS * reversed_id) + (VoterController._OFFSET_RSA_SIZE_BITS * reversed_id)
    (self._rsa_external_public_key, self._rsa_external_private_key) = rsa.newkeys(nbits = external_keys_size_bits)    "newkeys": Unknown word.
    VoterController._shared_rsa_external_public_keys.insert(0, self._rsa_external_public_key)

def vote(self, candidate: Candidate) -> VoteOnion:
    if candidate is None:
        raise ValueError("candidate cannot be None")

    vote_onion = VoteOnion(candidate.get_id(), self._generate_random_buffer_payload())
    self._dumps.append(vote_onion.get_buffer())

    for rsa_public_key in VoterController._shared_rsa_internal_public_keys:
        vote_onion.push_layer(rsa_public_key)
        self._dumps.append(vote_onion.get_buffer())

    for rsa_public_key in VoterController._shared_rsa_external_public_keys:
        buffer_noise = self._generate_random_buffer_payload()
        vote_onion.push_layer(rsa_public_key, buffer_noise)
        self._dumps.append(vote_onion.get_buffer())

    return vote_onion

def _generate_random_buffer_payload(self) -> bytes:
    return (''.join(random.choice(VoterController._BUFFER_NOISE_CONTENT) for _ in range(VoterController._BUFFER_NOISE_SIZE_BYTES))).encode()
```

Рисунок 1.2 – Код класу VoterController

```python
def decrypt_external_part(self, votes: list[VoteOnion]) -> None:
    for vote in votes:
        vote.pop_layer(self._rsa_external_private_key)

    if not self._validate_votes(votes):
        raise VoteProtocolError("Fraud attempt has been detected.")

    random.shuffle(votes)
    print(f"voter #{self._id} has removed external encryption layer.")

def decrypt_internal_part(self, votes: list[VoteOnion]) -> None:
    for vote in votes:
        vote.pop_layer(self._rsa_internal_private_key)
        vote.set_signature(elgamal.Elgamal.sign(vote.get_buffer()[:VoterController._ELGAMAL_SIGNATURE_PAYLOAD_LENGTH], self._elgamal_private_key))

    if not self._validate_votes(votes):
        raise VoteProtocolError("Fraud attempt has been detected.")

    other_voters = [voter for voter in VoterController._voters if voter != self]

    for voter in other_voters:
        voter.verify_signatures(votes, self._elgamal_public_key)

    print(f"voter #{self._id} has removed internal encryption layer.")

def _validate_votes(self, votes: list[VoteOnion]) -> bool:
    if votes is None:
        raise ValueError("votes cannot be None")

    is_own_vote_inside: bool = False
    self._dumps = self._dumps[:random.choice(votes).get_layers_count()]

    if len(votes) != VoterController._next_id:
        return False

    for vote in votes:
        if self._dumps[-1] == vote.get_buffer():
            is_own_vote_inside = True

    return is_own_vote_inside
```

Рисунок 1.3 – Код класу VoterController

```python
def verify_signatures(self, votes: list[VoteOnion], elgamal_public_key: elgamal.PublicKey) -> None:
    if votes is None:
        raise ValueError("votes cannot be None")

    if elgamal_public_key is None:
        raise ValueError("elgamal_public_key cannot be None")

    for vote in votes:
        try:
            elgamal.Elgamal.verify(vote.get_signature(), elgamal_public_key)
        except:
            raise VoteProtocolError("Fraud attempt has been detected.")

    if not self._validate_votes(votes):
        raise VoteProtocolError("Fraud attempt has been detected.")

    print(f"voter #{self._id} has validated signatures.")
```

Рисунок 1.4 – Код класу VoterController

Бюлетень представлений за допомогою класу VoteOnion, який має методи для додавання шарів шифрування за допомогою відкритого rsa-ключа та зняття за допомогою закритого rsa-ключа з пари. Також до шару може додаватися буфер байтів для шуму з метою, щоб виборець міг згодом ідентифікувати свій бюлетень серед інших зашифрованих. Код цього класу можна побачити на рисунках 1.5 – 1.6.

```python
class VoteOnion:

    def __init__(self, candidate_id: int, initial_buffer_noise: bytes) -> 'VoteOnion':
        self._signature = None
        self._layers_count: int = 1
        self._layers_with_noise: list[int] = list()
        self._buffer_noise_size_bytes = len(initial_buffer_noise)
        self._buffer: bytes = candidate_id.to_bytes() + initial_buffer_noise

    def get_buffer(self) -> bytes:
        return self._buffer

    def get_layers_count(self) -> int:
        return self._layers_count

    def get_signature(self) -> bytes:
        return self._signature

    def set_signature(self, value: bytes) -> None:
        self._signature = value

    def pop_layer(self, rsa_private_key: rsa.PrivateKey) -> bytes:
        if rsa_private_key is None:
            raise ValueError("rsa_private_key cannot be None")

        self._layers_count -= 1
        self._buffer = rsa.decrypt(self._buffer, rsa_private_key)

        if self._layers_count in self._layers_with_noise:
            self._layers_with_noise.remove(self._layers_count)
            self._buffer = self._buffer[:-self._buffer_noise_size_bytes]

        return self._buffer
```

Рисунок 1.5 – Код класу VoteOnion

```python
    def push_layer(self, rsa_public_key: rsa.PublicKey, buffer_payload: bytes = None) -> None:
        if rsa_public_key is None:
            raise ValueError("rsa_public_key cannot be None")

        if buffer_payload is not None:
            self._buffer += buffer_payload
            self._layers_with_noise.append(self._layers_count)

        self._layers_count += 1
        self._buffer = rsa.encrypt(self._buffer, rsa_public_key)

    def remove_initial_noise(self) -> None:
        self._buffer = self._buffer[:-self._buffer_noise_size_bytes]
```

Рисунок 1.6 – Код класу VoteOnion

```python
def main() -> None:
    candidate_1 = Candidate()
    candidate_2 = Candidate()

    voter_0 = VoterController()     # A
    voter_1 = VoterController()     # B
    voter_2 = VoterController()     # C
    voter_3 = VoterController()     # D

    voter_0.generate_rsa_keys_pairs()
    voter_1.generate_rsa_keys_pairs()
    voter_2.generate_rsa_keys_pairs()
    voter_3.generate_rsa_keys_pairs()

    vote_0 = voter_0.vote(candidate_1)
    vote_1 = voter_1.vote(candidate_2)
    vote_2 = voter_2.vote(candidate_1)
    vote_3 = voter_3.vote(candidate_1)

    votes:list[VoteOnion] = [vote_0, vote_1, vote_2, vote_3]

    # External part
    voter_0.decrypt_external_part(votes)
    voter_1.decrypt_external_part(votes)
    # voter_1.decrypt_external_part([vote_0, vote_1, deepcopy(vote_1), vote_2, vote_3])
    voter_2.decrypt_external_part(votes)
    voter_3.decrypt_external_part(votes)

    # Internal part
    voter_0.decrypt_internal_part(votes)
    voter_1.decrypt_internal_part(votes)
    voter_2.decrypt_internal_part(votes)
    # voter_2.decrypt_internal_part([vote_1, vote_2, deepcopy(vote_2), vote_3])
    voter_3.decrypt_internal_part(votes)

    print_results([candidate_1, candidate_2], votes)
```

Рисунок 1.7 – Тестування протоколу голосування

```
voter #1 has removed external encryption layer.
voter #2 has removed external encryption layer.
voter #3 has removed external encryption layer.
voter #4 has removed external encryption layer.
voter #2 has validated signatures.
voter #3 has validated signatures.
voter #4 has validated signatures.
voter #1 has removed internal encryption layer.
voter #1 has validated signatures.
voter #3 has validated signatures.
voter #4 has validated signatures.
voter #2 has removed internal encryption layer.
voter #1 has validated signatures.
voter #2 has validated signatures.
voter #4 has validated signatures.
voter #3 has removed internal encryption layer.
voter #1 has validated signatures.
voter #2 has validated signatures.
voter #3 has validated signatures.
voter #4 has removed internal encryption layer.

Candidate #1: 3 vote(s)
Candidate #2: 1 vote(s)
WINNER is candidate #1 with 3 vote(s).
```

Рисунок 1.8 – Результат роботи

Дослідження протоколу:

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

   Оскільки вся відповідальність про проведення голосування лежить на виборцях, то всім відома загальна кількість виборців, їх публічні ключі, тому стороння людина не може проголосувати без відповідного дозволу.
2. Перевірити чи може виборець голосувати кілька разів.

   Ні, не може, оскільки кожен виборець перед зняттям шару шифрування та підписанням перевіряє кількість бюлетенів та наявність свого у наборі й має право завершити процедуру голосування у випадку порушення.
3. Чи може хтось (інший виборець, ВК, стороння людина) дізнатися за кого проголосували інші виборці?

   Ні, бо бюлетені перемішуються, шифруються багатьма шарами, додаються випадкові набори байтів для шуму (щоб кожен міг ідентифікувати свій бюлетень у наборі). Але останній виборець перед оголошенням результатів може дізнатися результат раніше за інших.
4. Перевірити чи може інший виборець чи стороння людина проголосувати замість іншого зареєстрованого виборця.

   Це можливо лише у випадку отримання всіх приватних ключів виборця. Хоча бюлетенів можна зробити необмежену кількість, але їх аномальну кількість помітять інші.
5. Чи може хтось (інший виборець, ВК, стороння людина) таємно змінити голос в бюлетені?

   Ні, бо бюлетені шифруються багатьма шарами шифрування, додаються випадкові набори байтів для шуму, якщо змінити цей буфер байтів, то інший виборець не зможе ідентифікувати свій бюлетень.
6. Чи може виборець перевірити, що його голос врахований при підведенні кінцевих підсумків?

   У цьому можна переконатися на кожному етапі, коли отримується набір бюлетенів для розшифрування та перевірки підпису іншого виборця.

Висновок:

У ході виконання лабораторної роботи було змодельовано протокол електронного голосування з перемішуванням, який забезпечує конфіденційність, цілісність і автентифікацію даних виборців. Використання шифрування RSA для захисту повідомлень та алгоритму Ель-Гамаля для створення електронного цифрового підпису забезпечило необхідний рівень безпеки та можливість проведення механізму виборів.

Дослідження роботи протоколу показало, що система успішно запобігає спробам порушення процедури голосування. Було перевірено різні сценарії порушення, такі як багаторазове голосування, а також спроби проголосувати замість іншого виборця. Усі ці сценарії підтвердили стійкість системи до шахраювання.

Даний протокол також забезпечує виборцям можливість переконатися в тому, що їхній голос враховано в кінцевих підсумках, що підвищує рівень довіри до процесу голосування.