Міністерство освіти і науки України
Харківський національний університет імені В. Н. Каразіна

ТЕХНОЛОГІЯ БЛОКЧЕЙН ТА КРИПТОВАЛЮТНІ ОПЕРАЦІЙНІ ПЛАТФОРМИ

Методичні вказівки
до лабораторних робіт з дисципліни для здобувачів вищої освіти
першого (бакалаврського) рівня за спеціальністю F3 «Комп’ютерні науки»

Електронний ресурс

Харків – 2026

УДК 004.75:004.056.55
Р 60

Рецензенти:

Л. В. Ковальчук – доктор технічних наук, професор, провідний науковий співробітник відділу математичного і економетричного моделювання Інституту проблем моделювання в енергетиці імені Г. Є. Пухова;

Р. В. Олійников – доктор технічних наук, професор, професор кафедри кібербезпеки інформаційних систем, мереж і технологій Харківського національного університету імені В. Н. Каразіна.

Затверджено до розміщення в мережі Інтернет рішенням Науково-методичної ради Харківського національного університету імені В. Н. Каразіна (протокол № 8 від 21 травня 2026 року)

Р 60 Технологія блокчейн та криптовалютні операційні платформи : методичні вказівки до лабораторних робіт з дисципліни для здобувачів вищої освіти першого (бакалаврського) рівня за спеціальністю F3 «Комп'ютерні науки» [Електронний ресурс] / уклад. М. Ю. Родінко. – Харків : ХНУ імені В. Н. Каразіна, 2026. – (PDF 51 с.)

Дані методичні вказівки до лабораторних робіт розроблені для того, щоб провести здобувачів вищої освіти через повний цикл вивчення технології: від аналізу готових мереж до розробки власних рішень. Кожна лабораторна робота містить теоретичний мінімум, перелік завдань для виконання та контрольні запитання для самоперевірки. Виконання цих робіт дозволить здобувачам вищої освіти не лише зрозуміти внутрішню логіку блокчейн-платформ, а й отримати практичний досвід роботи з інструментами, що використовуються в сучасній індустрії FinTech та Web3.

УДК 004.75:004.056.55

ЗМІСТ

ВСТУП

Швидкий розвиток цифрової економіки та криптографічних технологій зумовив появу блокчейну як однієї з найбільш інноваційних технологій XXI століття. Блокчейн – це не лише основа для криптовалют, а й потужна децентралізована інфраструктура для побудови довірчих відносин, автоматизації процесів за допомогою смарт-контрактів та створення прозорих систем управління даними.

Метою дисципліни «Технологія блокчейн та криптовалютні операційні платформи» є формування у студентів спеціальності F3 теоретичних знань та практичних навичок щодо принципів побудови, функціонування та безпеки децентралізованих систем.

Дані методичні вказівки до лабораторних робіт розроблені для того, щоб провести студента через повний цикл вивчення технології: від аналізу готових мереж до розробки власних рішень. Лабораторний практикум охоплює такі ключові аспекти:

1. Аналіз структури даних: вивчення архітектури блоків, транзакцій та механізмів UTXO на прикладі мережі Bitcoin.
2. Алгоритми консенсусу: практичне дослідження принципів Proof-of-Work, Proof-of-Stake та BFT-протоколів, які забезпечують цілісність розподіленого реєстру.
3. Криптографічна стійкість: оцінка ймовірності атаки подвійної витрати) для розуміння меж безпеки мережі.
4. Програмування смарт-контрактів: опанування мови Solidity та середовища розгортання для створення децентралізованих додатків (dApps) у мережі Ethereum.
5. Екосистеми та управління: ознайомлення з сучасними платформами (як-от Cardano) та механізмами децентралізованого голосування і стейкінгу.

Кожна лабораторна робота містить теоретичний мінімум, перелік завдань для виконання та контрольні запитання для самоперевірки. Виконання цих робіт дозволить студентам не лише зрозуміти внутрішню логіку блокчейн-платформ, а й отримати практичний досвід роботи з інструментами, що використовуються в сучасній індустрії FinTech та Web3.

ЛАБОРАТОРНА РОБОТА №1. Аналіз блокчейну криптовалютної платформи Bitcoin

1.1. Мета роботи

Ознайомитись на практиці з мережею Bitcoin за допомогою відповідних ресурсів, що дозволяють оглядати блокчейни криптовалютних платформ (blockchain explorers).

1.2. Теоретичні відомості

Структура блокчейну як геш-ланцюжка. Блокчейн Bitcoin [1-7] – це розподілена децентралізована база даних, що представляє собою послідовність блоків, зв'язаних між собою за допомогою криптографічних геш-функцій. Кожен блок містить заголовок (*Header*) та список транзакцій.

Ключовим елементом зв'язку є поле *Previous Block Hash* у заголовку поточного блоку, яке містить результат обчислення функції SHA-256(SHA-256()) від заголовка попереднього блоку. Це створює ефект лавини: зміна одного біта в блоці N вимагає перерахунку всіх наступних блоків, що забезпечує незмінність даних (Immutability).

Модель транзакцій UTXO (Unspent Transaction Output). На відміну від традиційних банківських систем, Bitcoin не використовує поняття «баланс рахунку». Замість цього використовується модель UTXO (невитрачені виходи транзакцій).

- **Input (вхід):** посилання на вихід попередньої транзакції, де зберігаються кошти.
- **Output (вихід):** сума та «замок» (зазвичай скрипт P2PKH), який визначає умови витрачання коштів (володіння приватним ключем).

Загальний баланс адреси – це сума всіх UTXO, які доступні для розблокування підписом власника цієї адреси.

Дерево Меркла (Merkle Tree). Для оптимізації перевірки цілісності

транзакцій усередині блоку використовується бінарне дерево гешів – Дерево Меркла.

— Усі транзакції гешуються попарно, доки не залишиться один геш – Merkle Root, який записується в заголовок блоку.
— Це дозволяє реалізувати механізм SPV (Simplified Payment Verification): для підтвердження наявності транзакції в блоці достатньо мати лише заголовок блоку та «шлях Меркла» ($\log_2 N$ гешів), а не весь блок.

Алгоритм консенсусу Proof-of-Work (PoW). Процес додавання блоку (майнінг) полягає у розв'язанні криптографічної задачі: знайти таке значення *Nonce*, щоб подвійний геш заголовка блоку був меншим за певне цільове значення (*Target*):

$$\text{SHA256}(\text{SHA256}(\textit{Header})) < \textit{Target}.$$

Складність (*Difficulty*) автоматично коригується кожні 2016 блоків (~2 тижні), щоб підтримувати середній час генерації блоку на рівні 10 хвилин.

Формати адрес та криптографічні стандарти. Bitcoin використовує еліптичну криптографію (ECDSA) на кривій secp256k1.

— **Приватний ключ:** випадкове число 256 біт.
— **Публічний ключ:** точка на кривій, отримана множенням базової точки на приватний ключ.
— **Адреса:** результат обробки публічного ключа функціями SHA-256 та RIPEMD-160 (формати Legacy, SegWit/Bech32).

1.3. Хід роботи

Завдання 1. Ознайомитись із архітектурою блокчейну і принципами роботи алгоритму консенсусу Proof-of-Work.

Завдання 2. Ознайомитись із ресурсами, що дозволяють оглядати

блокчейни (blockchain explorers):

— https://explorer.bitquery.io/;

— https://live.blockcypher.com/btc/;

— https://btcscan.org/.

Додатково, можна виконати самостійний пошук подібних ресурсів і ознайомитись із ними.

Завдання 3. За допомогою одного або декількох ресурсів проаналізувати блокчейн криптовалюти Bitcoin, а саме структуру блоків. За результатами аналізу зробити опис атрибутів блоку.

Завдання 4. Проаналізувати, як із часом змінюється складність видобутку блоку (difficulty). У Bitcoin складність змінюється кожні 2016 блоків, або раз на два тижні. На прикладі 12 довільно обраних блоків, кожен з яких згенерований у різному місяці, дослідити, як змінювалась складність протягом одного конкретного року. На прикладі 5 випадково обраних блоків, вироблених у різні роки, дослідити, як змінювалась складність протягом 5 обраних років. Побудувати відповідні графіки і зробити висновки.

1.4. Контрольні питання

1. Які особливості відрізняють бази даних від блокчейну?
2. Поясніть, як працює блокчейн, опишіть його основні характеристики.
3. Поясніть процес майнінгу в Bitcoin: що це за тип алгоритму консенсусу і як він працює?
4. Опишіть структуру блоків в блокчейні Bitcoin. Що таке Merkle root?
5. Що таке складність видобутку блоку (difficulty)? Що таке target? Як ці параметри визначаються в Bitcoin?

ЛАБОРАТОРНА РОБОТА №2. Аналіз структури транзакцій Bitcoin. Програмна реалізація примітивів блок і транзакція

2.1. Мета роботи

Ознайомитись зі структурою транзакцій Bitcoin. Розробити програмну реалізацію примітивів блок і транзакція мовою програмування Java (або іншою на вибір за бажанням).

2.2. Теоретичні відомості

Bitcoin – перша децентралізована платіжна система на основі технології блокчейн. Запропонована у 2009 році Сатоші Накамото. Утримує першу сходинку серед всіх криптовалют за рівнем капіталізації.

На рис. 1 наведено структуру блокчейну мережі Bitcoin.

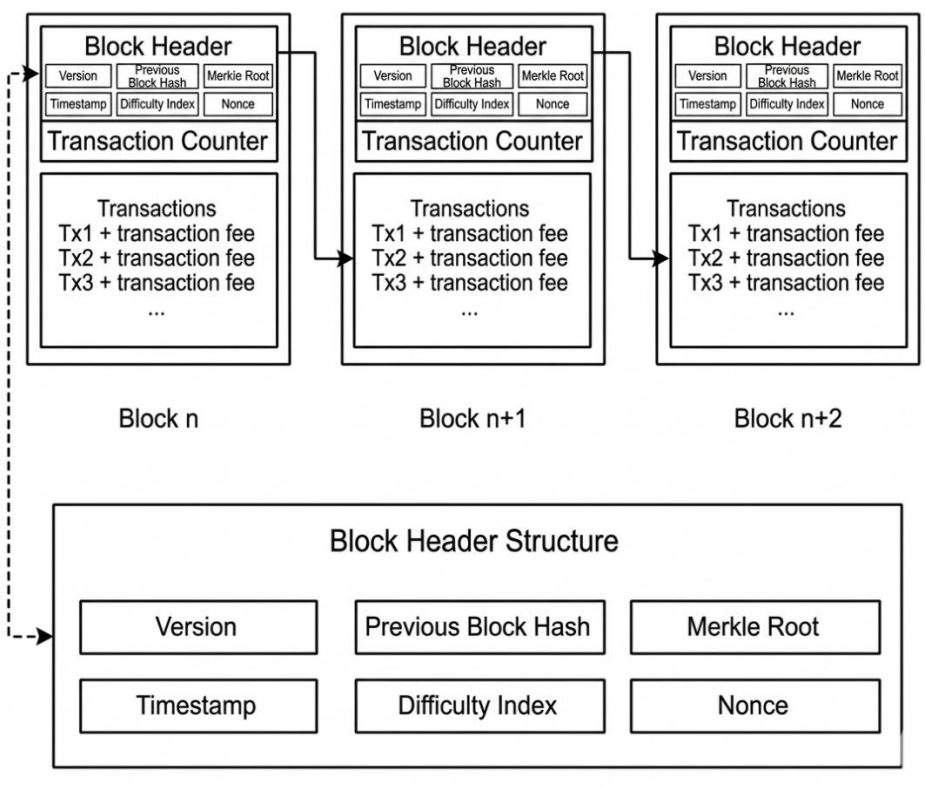

Рисунок 1 – Структура блокчейну Bitcoin

Структура транзакції. Транзакція – це структура даних, яка кодує передачу монет від джерела коштів, що називається входом, до місця

призначення, яке називається виходом (див. рис. 2). Вхідні та вихідні дані транзакцій не пов’язані з обліковими записами чи ідентифікаторами. Натомість, вони інтерпретуються як суми біткойнів, заблоковані приватним ключем, який може відкрити лише власник або особа, яка знає ключ.

Основою транзакції Bitcoin є невитрачений вихід транзакції або UTXO. UTXO – це неподільні шматки біткойн-валюти, заблоковані і прив’язані до певного власника, записані в блокчейні та розпізнані як валютні одиниці всією мережею. Мережа відстежує всі доступні (невитрачені) UTXO, які наразі нараховуються мільйонами. Щоразу, коли користувач отримує біткойн, ця сума записується в блокчейні як UTXO. Таким чином, біткойни користувача можуть бути розкидані як UTXO серед сотень транзакцій і сотень блоків (див. рис. 3).

По суті, не існує такого поняття, як збережений баланс біткойн-адреси чи рахунку; є лише розрізнені UTXO, що належать до певних власників. Концепція біткойн-балансу користувача – це похідна конструкція, створена програмою гаманця. Гаманець обчислює баланс користувача, скануючи блокчейн і агрегуючи всі UTXO, що належать цьому користувачеві.

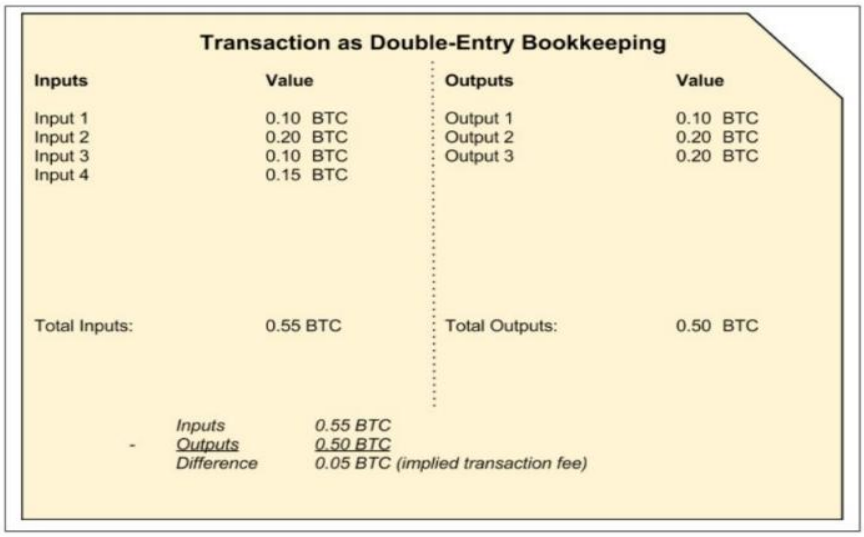

Рисунок 2 – Приклад транзакції Bitcoin [7]

Рисунок 3 – Приклад ланцюга транзакцій в Bitcoin [7]

Біткойн-гаманець містить набір пар ключів, кожна з яких складається з приватного та відкритого ключів (див. рис. 4).

Приватний ключ (k) – це число, зазвичай вибране випадковим чином. На основі приватного ключа генерується відкритий ключ (K).

На основі відкритого ключа (K) із використанням односторонньої криптографічної геш-функції створюється біткойн-адреса (A).

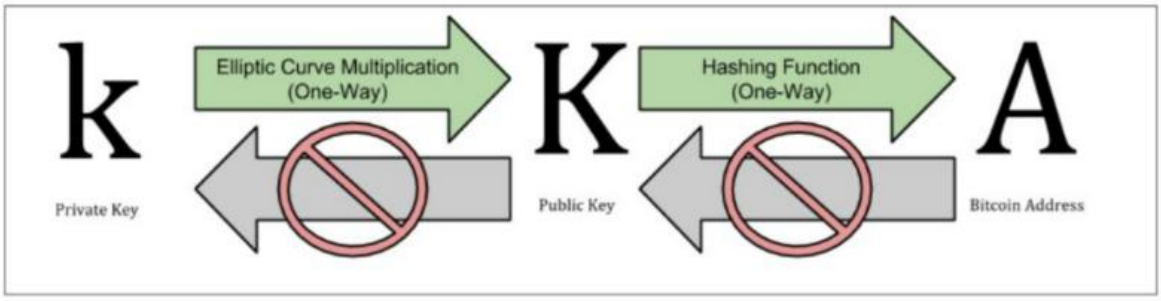

Рисунок 4 – Зв’язок між парою ключів та адресою [7]

Нижче наведено випадково згенерований приватний ключ у шістнадцятковому форматі (256 двійкових цифр, показаних як 64 шістнадцяткові цифри, кожна з яких складається з 4 бітів).

Випадково згенерований приватний ключ (k):

1E99423A4ED27608A15A2616A2B0E9E52CED330AC530EDCC32C8FFC6A526AEDD.

Відкритий ключ (точка еліптичної кривої), обчислюється як

$$K = k \times G,$$

де G - базова точка еліптичної кривої (точка на кривій, яка породжує підгрупу великого простого порядку n);

операція × - скалярне множення на кривій;

k - приватний ключ (випадково обране ціле число з діапазону [1; n-1]).

Пара ключів: (k, K), де $K = (x, y)$.

Біткойн-адреса. Починаючи з відкритого ключа K, ми обчислюємо геш SHA256, а потім обчислюємо геш RIPEMD160 за результатом, створюючи 160-бітне (20-байтове) число: $A = RIPEMD160(SHA256(K))$.

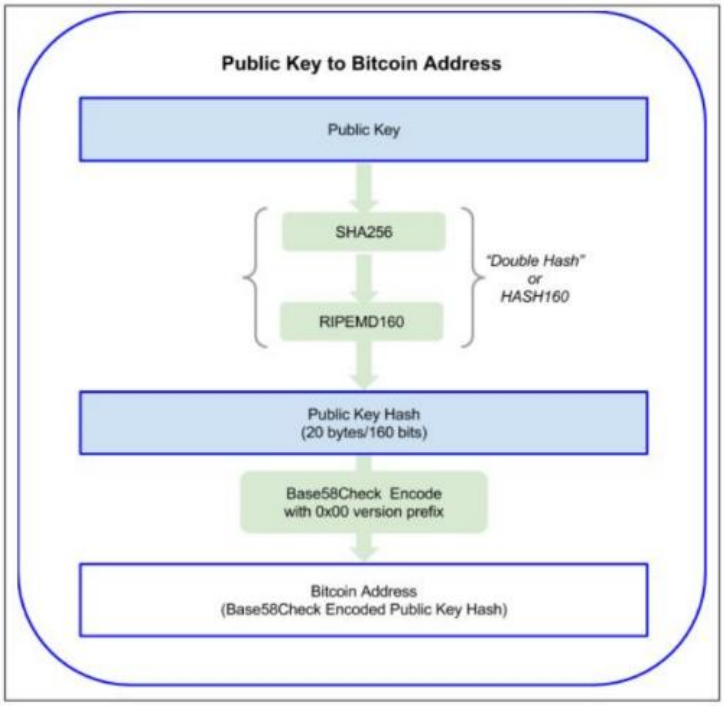

Рисунок 5 – Обчислення значення адреси [7]

2.3. Необхідні програмні засоби

Java Cryptography API надається тим, що офіційно називається Java Cryptography Extension (JCE).

The Java Cryptography Architecture (JCA) [8] – це назва внутрішнього дизайну Java Cryptography API.

До основних пакетів Java Cryptography API належать наступні:

— java.security;

— java.security.cert;

— java.security.spec;

— java.security.interfaces;

— javax.crypto [9];

— javax.crypto.spec;

— javax.crypto.interfaces.

Алгоритми, що підтримуються, визначені у [10].

Основний клас, який знадобиться для реалізації гешування:

— MessageDigest.

Приклад реалізації гешування рядка:

```
MessageDigest digest = MessageDigest.getInstance("SHA-256");
byte[] encodedhash = digest.digest(
originalString.getBytes(StandardCharsets.UTF_8));
```

Зверніть увагу, що метод гешування приймає на вхід масив байтів.

Реалізація застосування цифрового підпису за допомогою алгоритму ECDSA з бібліотеки Java.

1. Генерація ключової пари.

```
ECGenParameterSpec ecSpec = new ECGenParameterSpec("secp256k1");
KeyPairGenerator g = KeyPairGenerator.getInstance("EC");
g.initialize(ecSpec, new SecureRandom());
KeyPair keypair = g.generateKeyPair();
PublicKey publicKey = keypair.getPublic();
PrivateKey privateKey = keypair.getPrivate();
```

Підписування і перевірку можна реалізувати із використанням формату JSON для представлення/передачі даних підпису.

2. Підписування.

```
Signature ecdsaSign = Signature.getInstance("SHA256withECDSA");
ecdsaSign.initSign(privateKey);
ecdsaSign.update(plaintext.getBytes("UTF-8"));
byte[] signature = ecdsaSign.sign();

String pub = Base64.getEncoder().encodeToString(publicKey.getEncoded());
String sig = Base64.getEncoder().encodeToString(signature);

JSONObject obj = new JSONObject();
    obj.put("publicKey", pub);
    obj.put("signature", sig);
    obj.put("message", plaintext);
    obj.put("algorithm", "SHA256withECDSA");
```

Формат підписаних даних:


```
{
    "publicKey":
"MFYwEAYHKoZIzj0CAQYFK4EEAAoDQgAEMEV3EPREEDc0t4MPeuYgreLMHMVfD7iYJ2Cnkd
0ucwf3GYVySvYTttMVMNMEKF554NYmdrOlqwo2s8J2tKt/oQ==",
    "message": "Hello",
    "signature":
"MEUCIQCsuI4OcBAyA163kiWji1lb7xAtC8S0znf62EpdA+U4zQIgBcLbXtcuxXHcwQ9/Dm
iVfoiigKnefeYgpVXZzjIuYn8=",
    "algorithm": "SHA256withECDSA"
}
```


3. Перевірка.

```
Signature ecdsaVerify = Signature.getInstance(obj.getString("algorithm"));
KeyFactory kf = KeyFactory.getInstance("EC");

EncodedKeySpec publicKeySpec = new
X509EncodedKeySpec(Base64.getDecoder().decode(obj.getString("publicKey")));

KeyFactory keyFactory = KeyFactory.getInstance("EC");
PublicKey publicKey = keyFactory.generatePublic(publicKeySpec);

ecdsaVerify.initVerify(publicKey);
ecdsaVerify.update(obj.getString("message").getBytes("UTF-8"));
boolean result =
ecdsaVerify.verify(Base64.getDecoder().decode(obj.getString("signature")));
```

2.4. Хід роботи

Завдання 1. Розробити програмний примітив, що представляє **транзакцію**, тобто, клас *Transaction*, який містить приблизно наступні поля:

— вхідна адреса відправника (input);

— вихідна адреса(-и) отримувача(-ів) (output(s));

— сума переказу транзакції (amount);

— часова мітка створення транзакції (txTimestamp);

— геш транзакції (txHash);

— цифровий підпис.

Додати відповідні конструктори, сетери, гетера та toString(), метод для верифікації геш-значення та цифрового підпису транзакції.

Завдання 2. Розробити програмний примітив, що представляє **блок**, тобто, клас *Block*, який містить приблизно наступні поля (див. рис. 6):

— версія протоколу (version);

— геш-значення заголовку попереднього блока (prevHash);

— мітка часу (timestamp);

— складність видобутку блока, тобто, майнінгу (difficulty target);

— випадкове значення (nonce);

— корінь дерева Меркла (MerkleRoot);

— транзакції у вигляді дерева Меркла (геш-дерева);

— цифровий підпис.

Додати відповідні конструктори, сетери, гетера та toString().

Окрім, цього необхідно створити метод для генерації геш-значення заголовку блока за усіма його атрибутами, метод для верифікації блока (перевірки коректності геш-значення заголовку блока, кореня дерева та цифрового підпису).

За необхідністю, Ви можете додавати й інші методи, які допоможуть покращити реалізацію.

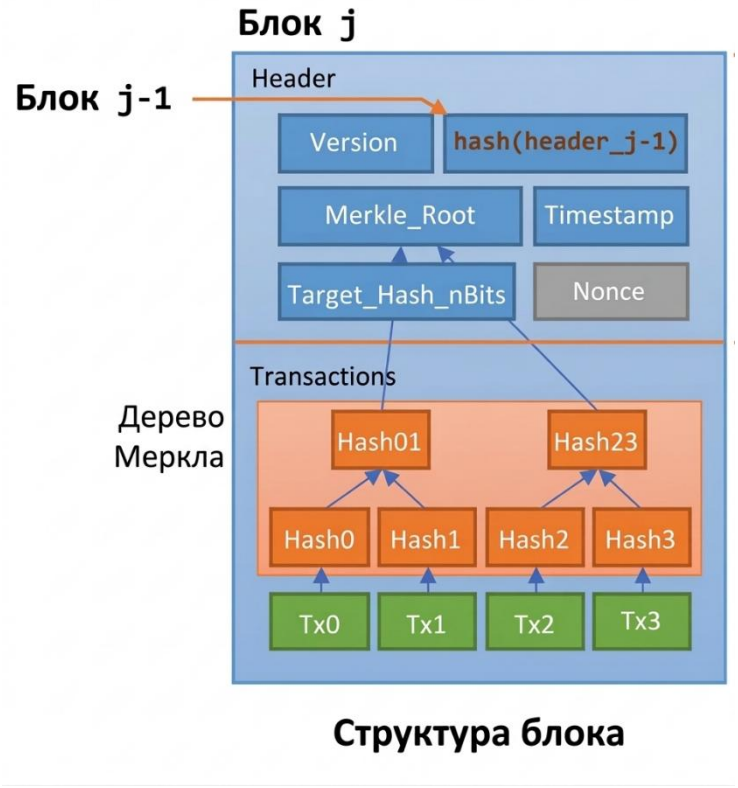

Рисунок 6 – Структура блока Bitcoin

Побудова заголовку блока.

Previous Block Hash: посилання на геш заголовку попереднього блоку в блокчейні;

Merkle Root: комбінований геш усіх транзакцій у блоці;

Timestamp: поточний час;

Difficulty Target: відображення того, наскільки важко знайти правильний геш для блоку, наприклад:

0000000000000000057fcc708cf0130d95e27c5819203e9f967ac56e4df598ee;

Nonce: початкове значення 0, яке змінюватиметься в процесі видобутку.

Гешування заголовку блока за допомогою SHA-256.

Після створення заголовку блоку використовують алгоритм гешування SHA-256, щоб створити вихідні дані фіксованого розміру (256 біт) – геш.

Завдання 3. Розробити клас *Blockchain*, основним полем якого є список блоків (наприклад, *List<Block>*). Додати конструктор, методи для додавання нового блока, отримання останнього блока в ланцюгу (або заданого). За потреби можна додавати й інші методи.

Завдання 4. Провести тестування створених класів. Створити декілька блоків, кожен з яких містить по декілька транзакцій. Протестувати методи верифікації транзакцій та блоків, відповідних геш-значень та цифрових підписів.

2.5. Контрольні питання

1. Що таке модель UTXO (Unspent Transaction Output)?
2. Чому в Bitcoin немає поняття «баланс рахунку» на рівні протоколу?
3. Як обчислюється загальний баланс гаманця користувача?
4. Яка роль входів (Inputs) та виходів (Outputs) у транзакції?
5. Як розраховується комісія (fee) за транзакцію?
6. Що таке Coinbase-транзакція?
7. Які обов’язкові поля містить заголовок блоку (Block Header)?
8. Дерево Меркла (Merkle Tree): як Merkle root дозволяє перевірити цілісність усіх транзакцій у блоці?
9. У чому перевага зберігання лише кореня Меркла в заголовку блоку замість повного списку транзакцій?

ЛАБОРАТОРНА РОБОТА №3. Програмна реалізація алгоритму Proof-of-Work та прототипу блокчейн мережі

3.1. Мета роботи

Розробити програмну реалізацію алгоритму майнінгу Proof-of-Work мовою програмування Java (або іншою на вибір за бажанням), використовуючи результати, отримані у попередній роботі. Здійснити створення та запуск тестової блокчейн мережі.

3.2. Теоретичні відомості

Алгоритм майнінгу

— **Перевірка транзакцій**: майнери збирають незавершені мережеві транзакції з mempool, щоб сформувати новий блок.

— **Створення нового блоку**: ці транзакції формуються у блок, очікуючи на підтвердження. Через обмежений простір, доступний у кожному новому блоці, майнери, як правило, віддають пріоритет транзакціям із високою комісією.

— **Обчислення кореня Меркла**: транзакції в новому блоці гешуються разом попарно, а потім ці геші гешуються разом і так далі, поки не буде знайдено єдиний геш для всіх транзакцій, відомий як корінь Меркла.

— **Розв’язування головоломки «Доказ роботи»**: найбільш трудомістким кроком є вирішення головоломки «Доказ роботи» (PoW). Це передбачає пошук nonce (змінного числа, яке майнери можуть змінювати), яке, додане до блоку та передане через геш-функцію, створює геш, який відповідає цільовій складності мережі.

— **Пошук Nonce**: майнери використовують спеціалізоване обладнання, щоб швидко вгадувати багато nonce значень у пошуках правильного,

яке розв’язує головоломку. Цей процес потребує значної обчислювальної потужності та електроенергії, оскільки ймовірність знайти правильний nonce за будь-якої спроби надзвичайно мала.

— **Перевірка мережею**: коли майнер вирішує головоломку, новий блок перевіряється іншими вузлами мережі. Блок приймається, якщо він відповідає необхідним умовам (транзакції дійсні, а головоломку PoW розв’язано правильно).

— **Додавання блоку до ланцюга блоків**: після перевірки новий блок додається до ланцюга блоків. Це оновлення поширюється по всій мережі, а транзакції, що містяться в блоці, вважаються підтвердженими.

— **Винагорода майнера**: успішний майнер отримує нещодавно створені біткойни (винагорода за блок) і комісію за транзакції з транзакцій блоку.

— **Повторення процесу**: майнери безперервно змагаються за вирішення нових головоломок, обробку нових транзакцій і захист мережі.

3.3. Хід роботи

3.3.1. Реалізація алгоритму майнінгу

До класів, що реалізують транзакцію, блок та блокчейн додати реалізацію класу майнерського вузла (*Node*). Основним полем цього класу буде об’єкт типу *Blockchain* (локальна копія блокчейну), а основним методом – *mineBlock*() для створення блоків (фактично, реалізація алгоритму майнінгу.

Створення потенційного блоку

— ***Побудова заголовка блоку.***

Заголовок блоку є важливим компонентом блоку-кандидата та містить:

Previous Block Hash: посилання на геш попереднього блоку в блокчейні;

Merkle Root: комбінований геш усіх транзакцій у блоці;

Timestamp: поточний час;

Difficulty Target: відображення того, наскільки важко знайти правильний геш для блоку, наприклад:

0000000000000000057fcc708cf0130d95e27c5819203e9f967ac56e4df598ee;

Nonce: початкове значення 0, яке змінюватиметься в процесі видобутку.

— ***Гешування заголовка блоку за допомогою SHA-256.***

Після створення заголовка блоку майнери використовують алгоритм гешування SHA-256, щоб створити вихідні дані фіксованого розміру (256 біт) – геш.

— ***Порівняння отриманого значення з Difficulty Target.***

Потім отриманий геш порівнюється з поточною цільовою складністю. Якщо геш відповідає критеріям (тобто має необхідну кількість початкових нулів), то блок дійсний. В іншому разі, майнеру доведеться змінити nonce і повторити спробу.

Налаштування Nonce

Nonce в заголовку блоку змінюють, збільшуючи його на одиницю для кожної нової спроби гешування. Змінюючи nonce, результуючий геш різко змінюється завдяки криптографічним властивостям алгоритму SHA-256.

Обрання значення Difficulty Target

Кількість нулів на початку геш-значення необхідно обрати із таким розрахунком, щоб необхідний геш міг бути знайдений на Вашому комп’ютері за прийнятний час (наприклад, декілька секунд). Можна

проаналізувати, як змінюватиметься час видобутку блоку при збільшенні складності.

3.3.2. Імітація роботи мережі та тестування

Імітувати роботу мережі будемо в окремому класі у методі main().

Варіант 1

Створити декілька вузлів (екземплярів класу *Node*), які послідовно будуть створювати блоки та «розсилати» їх всім іншим нодам (цю операцію імітуємо), які будуть верифікувати їх та додавати кожен у свій локальний блокчейн.

Варіант 2

Створити декілька вузлів (екземплярів класу *Node*), які будуть паралельно майнити блоки *в окремих потоках виконання* (конкурентно) та «розсилати» їх всім іншим нодам (цю операцію імітуємо), які будуть верифікувати їх та додавати кожен у свій локальний блокчейн.

Варіант 3*

Реалізувати прототип P2P мережі із використанням сокетів.

Також, перед цим необхідно згенерувати достатню кількість транзакцій, які майнери зможуть додавати в блоки. Вхідні та вихідні адреси та суми переказів (input, output, amount) можна генерувати випадковим чином.

3.4. Контрольні питання

1. Що таке «Target» (цільове значення) і як воно пов'язане зі складністю (Difficulty)?
2. Яким чином зміна лише одного поля nonce впливає на весь геш блоку?

3. Що робити майнеру, якщо він перебрав усі 2^{32} комбінацій nonce, але не знайшов потрібний геш?
4. Чи можна передбачити, скільки спроб знадобиться для знаходження блоку?
5. Навіщо мережі потрібно коригувати складність кожні 2016 блоків (у Bitcoin)?
6. Що станеться, якщо до мережі раптово приєднається величезна кількість нових майнерів?
7. Як вузол вирішує, яку гілку блокчейну вважати істинною, якщо він отримав два різні блоки одночасно?
8. Що таке «вага» ланцюга (Chainwork) і чому вона важливіша за просто кількість блоків?
9. Які перевірки повинен виконати вузол, перш ніж додати отриманий блок до свого локального ланцюга?

ЛАБОРАТОРНА РОБОТА №4. Дослідження атаки подвійної витрати

4.1. Мета роботи

Дослідити ймовірність атаки подвійної витрати (double spend attack) в залежності від кількості зловмисних вузлів та часу розповсюдження блока для протоколу PoW [1, 5, 11-12].

4.2. Теоретичні відомості

4.2.1. Модель Накамото (для Bitcoin) [1]

Параметри:

— p – ймовірність того, що чесні майнери створять наступний блок раніше, ніж зловмисники (тобто швидше);

— $q = 1 - p$ – ймовірність того, що зловмисники створять наступний блок раніше, ніж чесні майнери (тобто швидше);

— q_z – ймовірність того, що зловмисники коли-небудь наздоженуть (перевершать) z блоків позаду.

$$q_z = \begin{cases} 1 & if\ p \le q \\ (q/p)^z & if\ p > q \end{cases}$$

Потенційний прогрес зловмисника буде розподілом Пуассона з очікуваним значенням:

$$\lambda = z\frac{q}{p}$$

Ймовірність успіху атаки подвійної витрати після z блоків підтвердження становить:

$$1 - \sum_{k=0}^{z} \frac{\lambda^k e^{-\lambda}}{k!}\left(1 - (q/p)^{(z-k)}\right) \quad (1)$$

4.2.2. Модель з ненульовою затримкою [12]

Розглянемо наступну модель атаки.

Припущення:

— час є безперервним параметром;

— час синхронізації між чесними майнерами встановлюється як задане довільне значення;

— час синхронізації для зловмисників дорівнює нулю (робиться припущення на користь зловмисника);

— швидкість генерації блоків є довільним значенням (як для чесних майнерів, так і для зловмисників);

— частка геш-потужності зловмисників є довільною.

Параметри:

— T_H – це випадкова величина, яка вимірює час, необхідний для видобутку блока для чесних майнерів;

— T'_H – це випадкова величина, яка вимірює час, необхідний для видобутку та розповсюдження блока для чесних майнерів;

— T_M – це випадкова величина, яка вимірює час, необхідний для видобутку блока для зловмисних майнерів;

— T'_M – це випадкова величина, яка вимірює час, необхідний для видобутку та розповсюдження блока для зловмисних майнерів;

— α_H – інтенсивність генерації блоків для чесних майнерів;

— α_M – інтенсивність генерації блоків для зловмисних майнерів;

— $\alpha = \alpha_H + \alpha_M$ – загальна інтенсивність генерації блоків;

— D_H – час розповсюдження блока для чесних майнерів, $D_H > 0$;

— D_M – час розповсюдження блока для зловмисних майнерів, $D_M = 0$;

— p_H – ймовірність того, що чесні майнери створять наступний блок раніше, ніж зловмисники (тобто швидше);

— $p_M = 1 - p_H$ – ймовірність того, що зловмисники створять наступний

блок раніше, ніж чесні майнери (тобто швидше);

— p'_H – ймовірність того, що чесні майнери згенерують та поширять наступний блок для всіх (принаймні для всіх чесних) вузлів до того, як зловмисники згенерують свій наступний блок;

— $p'_M = 1 - p'_H$ – ймовірність альтернативної події.

Ймовірності p_H та p_M можна представити як:

$$p_H = \frac{\alpha_H}{\alpha_H + \alpha_M}, p_M = \frac{\alpha_M}{\alpha_H + \alpha_M}. \quad (2)$$

При розрахунках задати p_M (напр., 0.1), далі знайти $p_H = 1 - p_M$.

Взяти $\alpha_H + \alpha_M = 1/600$ сек = 0.00167.

Знайти α_H як $\alpha_H = (\alpha_H + \alpha_M) * p_H$. Аналогічно знайти p_M.

Ймовірності p'_H та p'_M знаходяться як (D_H задаєте самі):

$$p'_M = 1 - e^{-\alpha_M D_H} \cdot p_H;\ p'_H = e^{-\alpha_M D_H} \cdot p_H. \quad (3)$$

Ймовірність того, що зловмисники згенерують k блоків до того часу, як чесні майнери згенерують z блоків, обчислюється як:

$$P_z(k) = \frac{p_H}{(z-1)!} \cdot \frac{e^{-\alpha_M z D_H} \cdot (\alpha_M z D_H)^k}{k!} \cdot \sum_{i=0}^{k} \frac{(z-i+1)! \cdot C_k^i}{(\alpha z D_H)^i}, \text{for } z \in \mathbb{N}.$$

Ймовірність успіху атаки подвійної витрати після z блоків підтвердження становить:

$$P(z) = \begin{cases} 1, & \text{if } p'_M \geq p'_H; \\ 1 - \sum_{k=0}^{z} P_z(k) \left(1 - \left(\frac{p'_M}{p'_H}\right)^{z-k}\right), & \text{else.} \end{cases} \quad (4)$$

Тобто, перебираєте z до тих пір, поки ймовірність не стане менше 0.001.

Число комбінацій обчислюється як:

$$\binom{n}{k} = C_n^k = \frac{n(n-1)\dots(n-k+1)}{k(k-1)\dots 1}$$

4.3. Хід роботи

Завдання 1. ***Для моделі Накамото*** обчислити мінімальну кількість блоків підтвердження, яка гарантуватиме ймовірність успіху атаки подвійної витрати менше визначеного порогового значення (напр., 10^{-3}), для різних значень q. Для цього використати наступний алгоритм.

1. Задати q та обчислити $p = 1 - q$, де $q \in \{0.1;\ 0.45\}$ з кроком 0.05.
2. Ітеративно перебираючи z, знайти таке його значення, при якому ймовірність атаки буде менше 1) 10^{-3} 2) 10^{-4} 3) 10^{-5}. Для обчислення ймовірності використовувати (1).

Алгоритм для обчислення кількості блоків підтвердження, яка гарантуватиме ймовірність успіху атаки подвійної витрати менше 10^{-3} для фіксованого значення q, виглядатиме наступним чином [1]:

```
double AttackerSuccessProbability(double q)
{
  double p = 1.0 - q;
  int z = 0;
  while(true){
    double lambda = z * (q / p);
    double sum = 1.0;
    int i, k;
    for (k = 0; k <= z; k++)
    {
      double poisson = exp(-lambda);
      for (i = 1; i <= k; i++)
        poisson *= lambda / i;
      sum -= poisson * (1 - pow(q / p, z - k));
    }
    if(abs(sum) > 0.001)
     z++;
    else return z;
  }
}
```

За результатами побудувати графіки залежності кількості блоків підтвердження (вісь y) від частки зловмисників (вісь x) для різних порогів ймовірності.

Завдання 2*. ***Для вдосконаленої моделі*** обчислити мінімальну кількість блоків підтвердження, яка гарантуватиме ймовірність успіху атаки подвійної витрати менше визначеного порогового значення (напр., 10^{-3}), для різних значень p_M. Значення p_M взяти аналогічним значенню q з попередньої задачі. Значення затримки D_H узяти рівним 0, 15, 30, 60 секунд.

4.4. Контрольні питання

1. Чому проблему подвійної витрати легко вирішити в централізованих системах і чому вона є викликом для P2P-мереж?
2. Яка роль механізму підтверджень (Confirmations) у запобіганні цій атаці?
3. Чому стандартною рекомендацією для великих транзакцій є очікування саме 6 підтверджень?
4. Як кожне нове підтвердження експоненціально знижує ймовірність успіху зловмисника?
5. Як зловмисник використовує мережеві затримки, щоб надіслати дві різні транзакції з одним і тим самим UTXO одночасно?
6. Опишіть сценарій, за якого майнер включає транзакцію у вже знайдений, але ще не опублікований блок.
7. Як залежить успіх атаки від частки потужності зловмисника (q) та кількості підтверджень (z), які чекає отримувач?
8. При якому порозі потужності мережі атака стає практично неминучою?

ЛАБОРАТОРНА РОБОТА №5. Дослідження властивостей протоколу консенсусу BFT

5.1. Мета роботи

Дослідити ефективність протоколу BFT в залежності від кількості учасників при фіксованій затримці розповсюдження повідомлення.

5.2. Теоретичні відомості

Byzantine Fault Tolerance (BFT) – це властивість системи залишатися працездатною та досягати консенсусу навіть у випадках, коли частина її вузлів діє зловмисно: надсилає суперечливі дані, ігнорує повідомлення або намагається зламати протокол.

Алгоритм PBFT (Practical Byzantine Fault Tolerance) [13,14], запропонований Мігелем Кастро та Барбарою Лісков у 1999 році, став першим BFT-рішенням, здатним працювати в реальних системах.

Основні умови роботи:

— для того, щоб система витримала f зловмисних вузлів, загальна кількість вузлів у мережі n має бути не меншою за $3f + 1$;

— один вузол обирається первинним (Primary/Leader), інші – репліками (Backups).

Консенсус у PBFT проходить через п’ять основних етапів (див. рис. 7).

1. **Request.** Клієнт надсилає запит на виконання операції первинному вузлу.
2. **Pre-Prepare.** Первинний вузол присвоює запиту порядковий номер і розсилає його всім іншим реплікам.
3. **Prepare.** Кожна репліка перевіряє запит і розсилає повідомлення «Prepare» всім іншим вузлам. Це підтверджує, що вузол отримав запит і згоден із його номером.

4. **Commit.** Коли вузол отримує $2f + 1$ підтвердження «Prepare», він розсилає повідомлення «Commit». Це означає, що більшість вузлів погодилася на виконання.
5. **Reply.** Після отримання $2f + 1$ повідомлень «Commit» вузол виконує запит і надсилає відповідь клієнту. Клієнт чекає на $f + 1$ однакову відповідь.

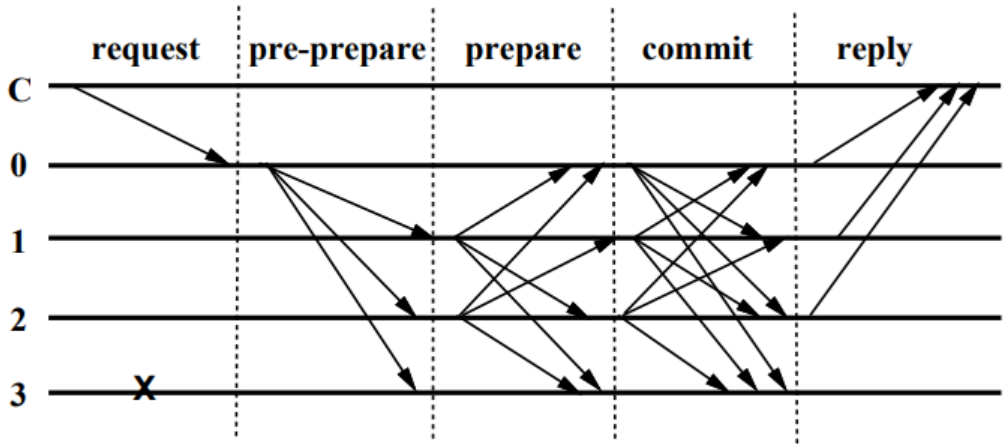

Рисунок 7 – Процес досягнення консенсусу PBFT [13]

До **переваг** протоколу слід віднести наступні.

— Енергоефективність: на відміну від Proof-of-Work, не потребує складних обчислень.

— Абсолютна фіналізація (finality): як тільки транзакція підтверджена, вона є остаточною і не може бути скасована, отже, немає «розгалужень» ланцюга.

Основним **недоліком** протоколу є низький рівень масштабованості. Через необхідність кожного вузла спілкуватися з кожним ($O(n^2)$ повідомлень), продуктивність різко падає при збільшенні кількості вузлів понад кілька сотень. Крім того, слід відзначити і чутливість до Primary. Якщо лідер стає зловмисним, система має ініціювати процедуру View Change для переобрання лідера, що уповільнює роботу.

Сьогодні ідеї PBFT лежать в основі консенсусів багатьох сучасних блокчейнів, таких як Hyperledger Fabric, Tendermint та модифікації для Ethereum (у частині фіналізації блоків).

Спрощений BFT

1. **Реплікація стану.** Кожен вузол мережі підтримує актуальну копію розподіленого реєстру та бере участь у верифікації нових даних.

2. **Децентралізована верифікація.** Сформований блок транслюється всім учасникам мережі (broadcast), які автономно перевіряють коректність цифрових підписів та валідність включених транзакцій.

3. **Етап голосування.** За результатами перевірки вузли ініціюють процес голосування щодо включення блоку до основного ланцюга.

4. **Досягнення кворуму.** Остаточне фіналізування блоку відбувається лише за умови отримання понад 2/3 підтверджень (голосів). Такий поріг візантійської відмовостійкості гарантує цілісність системи за наявності компрометованих або несправних вузлів.

5.3. Хід роботи

1. Реалізувати спрощений BFT-протокол. Для цього використати примітиви, розроблені у роботі 1, підкоригувавши їх під новий протокол. Протокол BFT реалізувати наступним чином:
   - — створити *n* екземплярів класу Node;
   - — перший вузол обрати лідером: він повинен створити блок з транзакціями і розіслати його всім іншим вузлам (процедуру розсилки імітуємо, як і в л/р №1; *за бажанням можна реалізувати за допомогою сокетів*);
   - — кожен з *n*-1 вузлів перевіряє отриманий блок на валідність;
   - — якщо блок валідний, вузол повинен передати всім іншим вузлам повідомлення «Valid»; в методі передачі повідомлення

встановити фіксований час затримки (за допомогою Thread.sleep());

— кожен вузол, що отримав від інших вузлів не менше ⅔ інших вузлів повідомлення «Valid», додає блок до своєї копії блокчейну.

2. Проаналізувати, за який час протокол успішно завершить роботу щодо узгодження одного блоку за різної кількості учасників *n* (наприклад, 10, 50, 100, 500, 1000; *при використанні сокетів можна взяти значення 5, 10, 15, 20, 30*).
3. Побудувати графік залежності часу виконання протоколу від кількості учасників, зробити висновок.

5.4. Контрольні питання

1. У чому полягає суть «Задачі візантійських генералів»?
2. Чому для досягнення консенсусу при наявності f зловмисників необхідно мати мінімум $3f + 1$ вузол?
3. Як протокол гарантує, що два чесних вузли не приймуть різні рішення (Safety)?
4. Як змінюється кількість повідомлень у мережі при збільшенні кількості вузлів n?
5. Чому класичний BFT важко використовувати у публічних мережах з тисячами учасників, на відміну від PoW?
6. Що означає поняття «Finality» (завершеність)? Чому в BFT вона абсолютна, а в Bitcoin – лише імовірнісна?
7. Які обмеження накладає BFT на членство в мережі (Permissioned vs Permissionless)?

ЛАБОРАТОРНА РОБОТА №6. Створення та розгортання смарт-контракту

6.1. Мета роботи

Створити простий смарт-контракт мовою Solidity. Розгорнути та верифікувати контракт із використанням тестової мережі Sepolia.

6.2. Теоретичні відомості

Ethereum [15,16] – це децентралізована блокчейн-платформа з відкритим вихідним кодом, яка дозволяє створювати та запускати смартконтракти та децентралізовані додатки (dApps) без посередників.

На відміну від Bitcoin, який був розроблений переважно як платіжна система, Ethereum функціонує як «світовий комп'ютер».

1. Концепція смарт-контракту

Смарт-контракт – це програмний код, який зберігається в блокчейні та автоматично виконується при виконанні певних умов.

- **Детермінізм:** результат виконання контракту однаковий для всіх вузлів мережі при одних і тих самих вхідних даних.
- **Автономність:** після деплою контракт не потребує посередників для роботи.

2. Віртуальна машина (EVM)

Ethereum Virtual Machine (EVM) – це середовище виконання для смарт-контрактів. Код, написаний високорівневою мовою (наприклад, Solidity), компілюється у **байт-код** (Bytecode), який розуміє EVM.

- **Сховище (Storage):** постійна пам'ять, дані в якій записуються в блокчейн (коштує дорого).
- **Пам'ять (Memory):** тимчасове сховище, дані в якому існують лише під час виконання транзакції (коштує дешевше).

3. Економіка газу (Gas)

Газ – це одиниця вимірювання обчислювальної роботи. Кожна операція (додавання, запис у пам’ять) має свою вартість у газі.

— **Gas Limit:** максимальна кількість газу, яку ви готові витратити на транзакцію.

— **Gas Price:** ціна, яку ви готові заплатити за одиницю газу (зазвичай у Gwei).

4. Життєвий цикл розгортання

Розгортання (Deployment) – це процес створення спеціальної транзакції, де поле *to* порожнє, а поле *data* містить компільований байт-код контракту.

1. **Написання коду:** створення файлу .sol (Solidity).
2. **Компіляція:** отримання байт-коду та ABI (Application Binary Interface).
3. **Підпис транзакції:** відправка байт-коду в мережу за допомогою приватного ключа.
4. **Майнінг/Валідація:** включення транзакції в блок. Після цього контракт отримує унікальну адресу.

5. ABI (Application Binary Interface)

ABI – це інтерфейс у форматі JSON, який описує функції та параметри контракту. Без ABI зовнішні програми (наприклад, ваш вебсайт або скрипт) не знатимуть, як правильно «звернутися» до функцій контракту, оскільки в блокчейні вони представлені лише набором байтів.

6. Тестові мережі (Testnets)

Оскільки розгортання в основній мережі (Mainnet) коштує реальних грошей, розробники використовують тестові мережі (наприклад, Sepolia). Вони імітують роботу блокчейну, але використовують безкоштовні токени, які можна отримати через «крани» (Faucets).

Основні інструменти (Stack)

— **Solidity [17]:** основна мова програмування.

— **Remix IDE [18]:** браузерне середовище для швидкого написання та тестування.

— **MetaMask:** криптовалютний гаманець, що виступає як підпис та міст до блокчейну.

6.3. Хід роботи

1. Створюємо акаунт Metamask. Для цього заходимо сюди і встановлюємо відповідне розширення браузера: https://chromewebstore.google.com/detail/metamask/nkbihfbeogaeaoehlefnkodbefgpgknn?hl=en

2. Тепер необхідно отримати монети для віртуальної мережі.

   В інтерфейсі Metamask у верхньому лівому куті серед запропонованих мереж обираємо тестову мережу Sepolia і копіюємо адресу нашого акаунту (див. рис. 8).

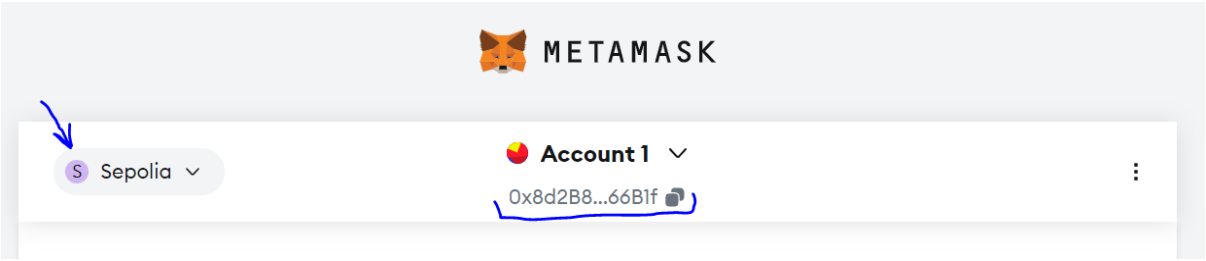

Рисунок 8 – Обрання тестової мережі

Заходимо на ресурс для отримання монет: https://cloud.google.com/application/web3/faucet/ethereum/sepolia

Вводимо адресу і отримуємо 0.05 Sepolia ETH на свій баланс (див. рис. 9).

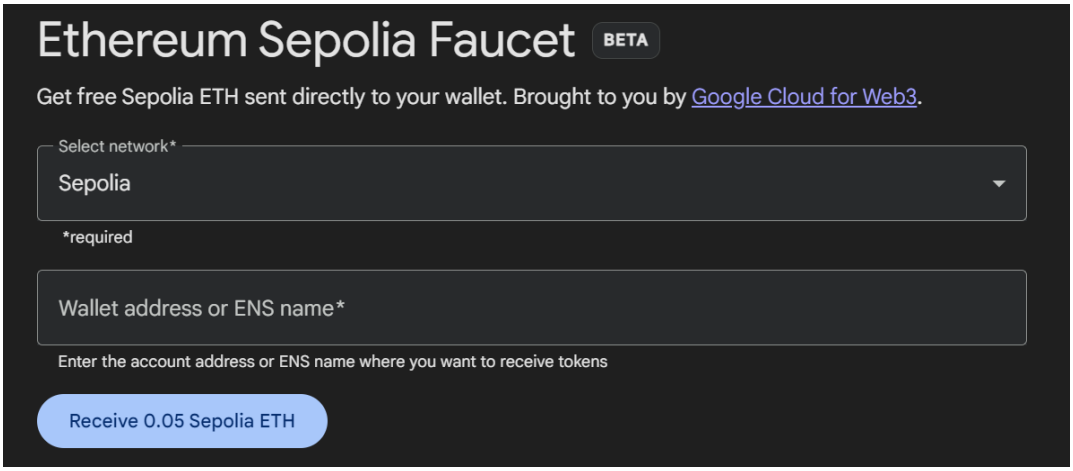

Рисунок 9 – Отримання віртуальних монет

Перевіряємо баланс акаунту Metamask (див. рис. 10).

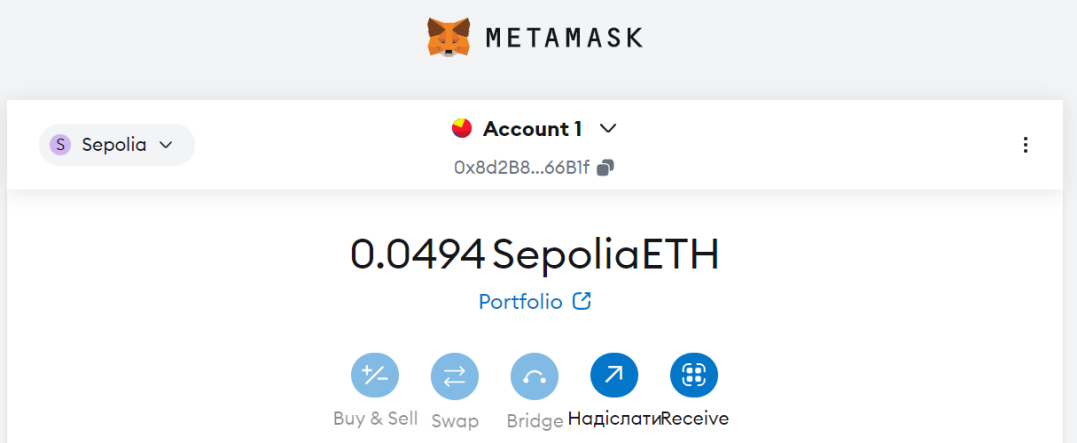

Рисунок 10 – Перевірка балансу

3. Заходимо в Remix та створюємо новий файл MyFirstContract.sol з наступним кодом смарт-контракту (див. рис. 11).

```
pragma solidity 0.8.19;
contract GreetingContract {
string public greet;
constructor(string memory name) {
require(bytes(name).length > 0, "name can't be empty");
greet = string.concat("Hello, my name is ", name);
}
}
```

Рисунок 11 – Код смарт-контракту

4. Компілюємо файл, використовуючи версію компілятору Solidity 0.8.19 (див. рис. 12).

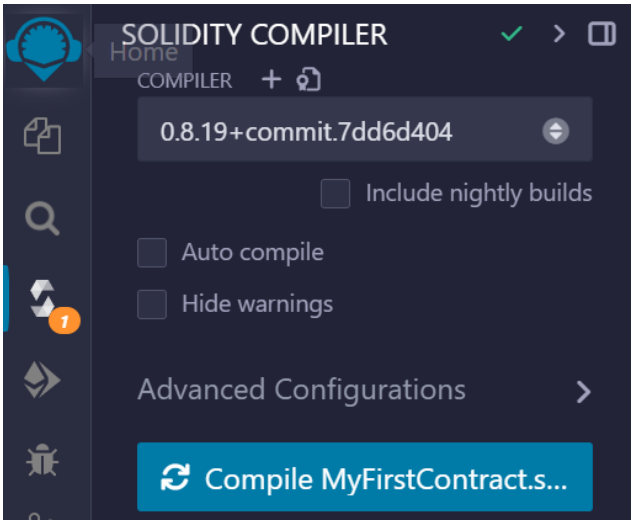

Рисунок 12 – Компіляція смарт-контракту

5. Підключаємо обліковий запис Metamask до Remix, обравши відповідне середовище і ввівши значення адреси (див. рис. 13).

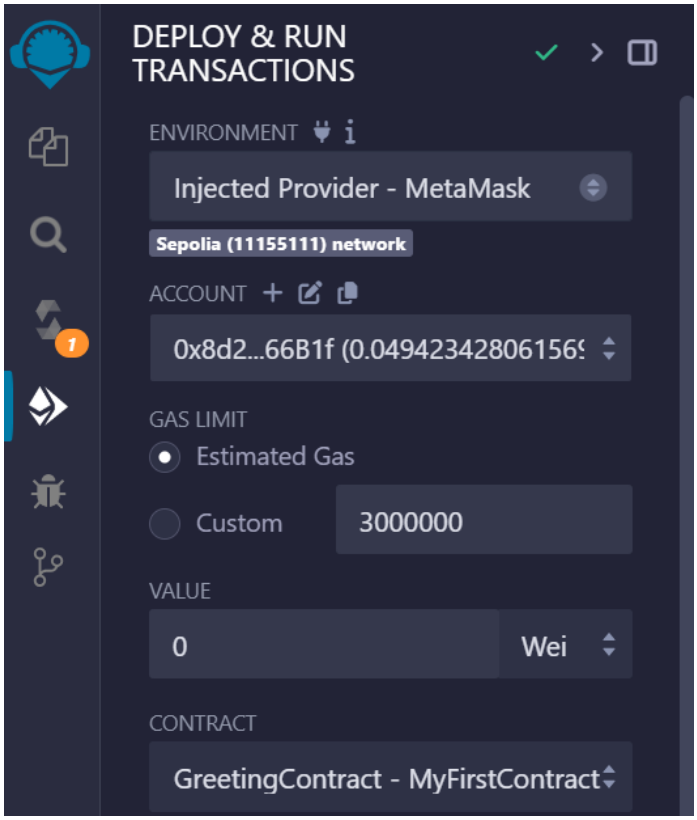

Рисунок 13 – Підключення гаманця

6. Розгортаємо контракт у Sepolia, надавши своє ім’я конструктору смарт-контракту (див. рис. 14).

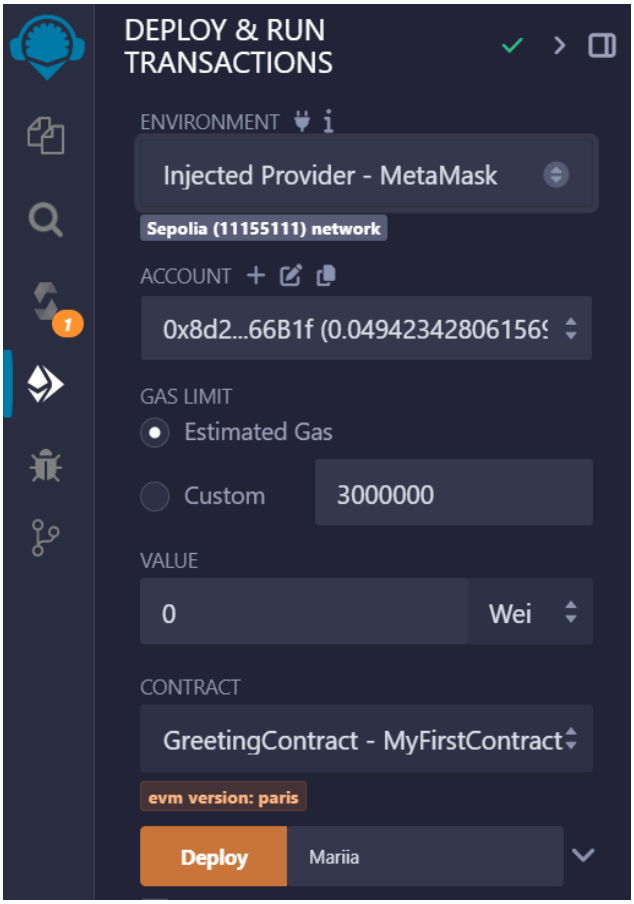

Рисунок 14 – Введення параметру

7. Дивимось деталі розгортання за допомогою Debug (див. рис. 15).

Рисунок 15 – Деталі розгортання смарт-контракту

8. Перевіряємо роботу контракту в Remix, натиснувши на кнопку greet (див. рис. 16).

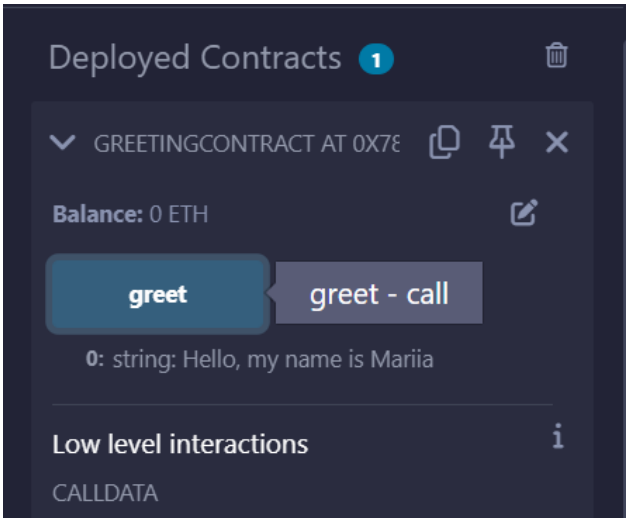

Рисунок 16 – Перевірка роботи

9. Знаходимо свій контракт за адресою на https://sepolia.etherscan.io/, далі натискаємо Contract -> Verify and Publish.

   Адресу контракту дивимось тут (рис. 17).

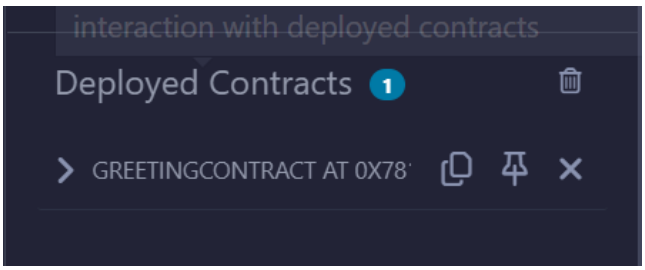

Рисунок 17 – Адреса контракту

Знаходимо контракт за адресою на Etherscan (див. рис. 18).

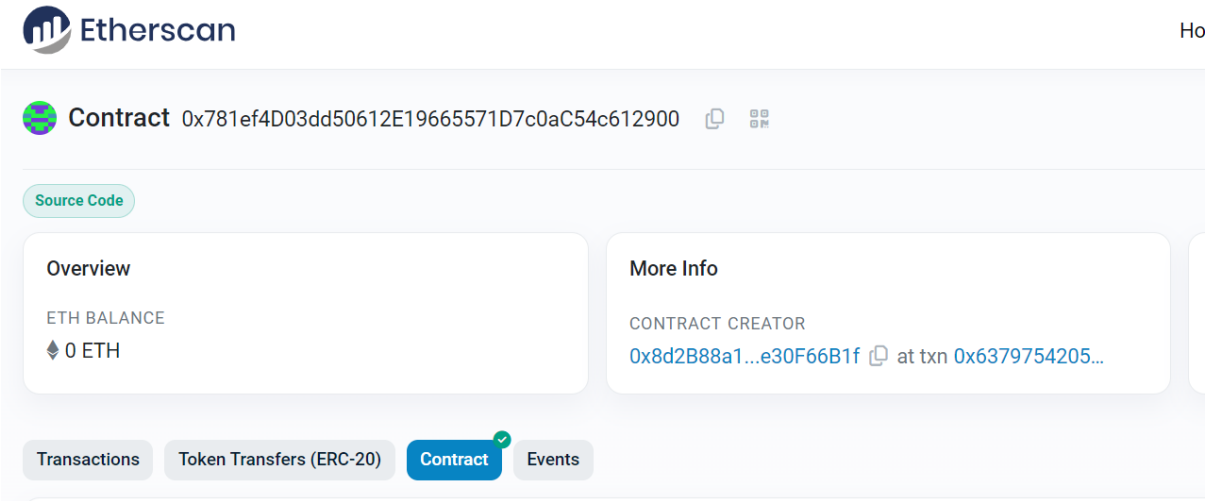

Рисунок 18 – Контракт на Etherscan

Під час верифікації вводимо наступні параметри:

- Тип компілятора - Solidity (single file);
- Тип ліцензії - No License;
- Аргумент конструктора, кодований в ABI, залишаємо той, що запропоновано за замовчуванням.

(Можна також використати https://abi.hashex.org/.)

10. Знову відкриваємо сторінку https://sepolia.etherscan.io/ -> Read Contract -> greet.

Отримуємо рядок привітання із вказаним іменем (див. рис. 19).

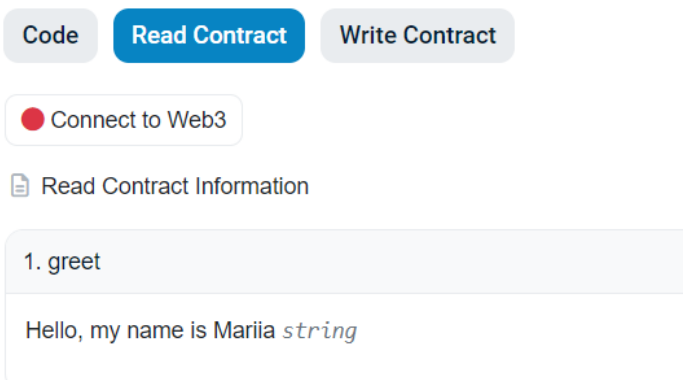

Рисунок 19 – Читання контракту

Конракт розгорнуто.

Звіт повинен містити скріншоти усіх кроків виконання роботи, а також посилання на розгорнутий і верифікований контракт на https://sepolia.etherscan.io/.

6.4. Контрольні питання

1. Чим відрізняється статус коду контракту до розгортання та після нього?
2. Чи можна змінити логіку контракту (код) після того, як він потрапив у блокчейн?
3. Що таке байт-код (Bytecode) і для чого потрібна віртуальна машина (наприклад, EVM)?
4. Яку роль відіграє JSON-файл ABI при взаємодії фронтенду або скриптів із розгорнутим контрактом?
5. Чому розгортання контракту зазвичай коштує значно дорожче, ніж звичайна транзакція переказу коштів?Як передати початкові параметри (наприклад, ім'я токена або адресу власника) під час створення контракту?
6. Де взяти «тестові» монети і чи мають вони реальну ринкову вартість?
7. Що таке «верифікація контракту» на ресурсах типу Etherscan і навіщо вона потрібна користувачам?

ЛАБОРАТОРНА РОБОТА №7. Дослідження криптовалютної платформи Cardano

7.1. Мета роботи

7.1.1. Ознайомитися з основами роботи платформи Cardano

Вивчити її особливості, технології, які використовуються для забезпечення безпеки та децентралізації, а також виконати практичні завдання для розуміння роботи з мережею Cardano.

7.1.2. Ознайомитися з проєктом Catalyst

Дослідити процес децентралізованого управління в екосистемі Cardano, зрозуміти, як працює система голосування та розподілу фінансування, а також виконати практичні завдання для отримання навичок роботи з платформою Catalyst.

7.2.Теоретичні відомості

Cardano [19] – це блокчейн-платформа третього покоління, яка вирізняється науковим підходом, суворою перевіркою коду (формальні методи) та унікальною архітектурою. Вона була створена для подолання «трилеми блокчейну» (безпека, масштабованість, децентралізація).

Ключові аспекти архітектури та технологій Cardano полягають в наступному.

1. Дворівнева архітектура

Cardano розділяє функціонал мережі на два рівні, що дозволяє легше масштабувати систему та оновлювати її без розгалужень (hard forks).

CSL (Cardano Settlement Layer): «Рівень розрахунків», де фіксуються транзакції з монетою ADA. Це фундамент, який відповідає за передачу вартості.

CCL (Cardano Computation Layer): «Рівень обчислень», де виконуються смарт-контракти та працюють децентралізовані додатки (dApps).

2. Модель eUTXO (Extended Unspent Transaction Output)

На відміну від Ethereum (який використовує модель «облікових записів» – account-based), Cardano використовує розширену модель UTXO.

- Кожна транзакція споживає виходи попередніх транзакцій (UTXO) і створює нові. Це схоже на готівку: ви не міняєте баланс рахунку, а «передаєте монети».
- **Переваги.** Вища детермінованість (результат транзакції можна передбачити заздалегідь), паралелізм (транзакції не блокують одна одну) та вищий рівень безпеки.
- **eUTXO.** Модель «Extended» додає можливість зберігати не лише значення активу, а й складні дані (datum) та логіку (script), що дозволяє реалізовувати смарт-контракти.

3. Протокол консенсусу Ouroboros

Ouroboros – це перший у світі протокол Proof-of-Stake (PoS), безпека якого доведена математично.

- **Механіка.** Час у мережі поділений на **епохи** та **слоти**. У кожному слоті обирається «лідер», який створює блок.
- **Енергоефективність.** Оскільки це PoS, мережа не потребує енергозатратного майнінгу, як у Bitcoin (PoW).
- **Децентралізація.** Система винагород стимулює користувачів делегувати ADA у стейк-пули, що забезпечує розподілене управління безпекою.

4. Смарт-контракти: Plutus та Marlowe

Cardano пропонує інструменти для розробників різного рівня.

Plutus. Мова програмування для смарт-контрактів на базі **Haskell**. Вона забезпечує високу безпеку завдяки функціональному програмуванню та можливості формальної верифікації коду (доведення коректності роботи контракту перед запуском).

Marlowe. Спеціалізована мова для фінансових контрактів. Вона має візуальний інтерфейс, що дозволяє навіть людям без глибоких навичок програмування створювати безпечні фінансові угоди.

Project Catalyst [20] – це децентралізований інкубатор та грантовий фонд екосистеми Cardano. Якщо коротко, це механізм, де сама спільнота вирішує, на які проєкти витрачати кошти з державної скарбниці (Treasury) блокчейну.

Процес проходить циклічними раундами (Funds). Кожен раунд має 4 основні етапи.

1. **Подання ідей.** Будь-хто (розробник, маркетолог, викладач) може запропонувати проєкт, який принесе користь Cardano.
2. **Обговорення та рев'ю.** Спільнота та спеціальні експерти (Community Reviewers) оцінюють пропозиції на життєздатність та корисність.
3. **Голосування.** Власники монет ADA голосують через спеціальний застосунок. Вага голосу пропорційна кількості монет на гаманці.
4. **Фінансування.** Проєкти-переможці отримують гранти у ADA, але не всі гроші одразу, а частинами – після виконання певних етапів (Milestones).

2. Масштаб (станом на 2026 рік)

Catalyst став одним із найбільших децентралізованих фондів у світі.

— Виплачено понад 150 мільйонів доларів (в еквіваленті) на розвиток екосистеми.

— Профінансовано більше **2 200 проєктів** (від гаманців та DEX до освітніх курсів та благодійності).

7.3. Хід роботи

7.3.1. Теоретичні завдання

1. **Вивчення платформи Cardano**
   — Ознайомитися з особливостями блокчейну Cardano: архітектура мережі, рівні (рівень розрахунків та рівень обчислень).
   — Дослідити відмінності між Cardano та іншими платформами (Bitcoin, Ethereum).
2. **Механізм консенсусу Ouroboros**
   — Розглянути механізм консенсусу Proof of Stake (PoS), на якому побудовано Cardano.
   — Вивчити принцип роботи Ouroboros – унікальної PoS-системи, що використовується у Cardano, та її роль у забезпеченні безпеки мережі.
3. **Токени ADA**
   — Дослідити особливості внутрішнього токена ADA, його роль у мережі, способи використання та можливості стейкінгу.
4. **Що таке Catalyst?**
   — Дослідити концепцію проєкту Catalyst як системи децентралізованого управління в Cardano.
   — Розглянути основні цілі Catalyst: підтримка інноваційних проєктів, стимулювання розвитку екосистеми, залучення спільноти.
5. **Фінансування та механізм голосування**
   — Вивчити, як відбувається розподіл фінансування на проєкти, що подають заявки на Catalyst.

— Розглянути роль механізму голосування для прийняття рішень спільнотою Cardano.
— Дослідити, як участь у голосуванні допомагає ADA-холдеру впливати на розвиток екосистеми.

6. **Процес подання проєктів**
   — Ознайомитися з процесом подання проєктів на фінансування в Catalyst, від заявки до оцінювання.
   — Дізнатися про основні критерії, за якими оцінюють заявки: технічна обґрунтованість, користь для екосистеми, перспективи розвитку тощо.
7. **Фази проєкту Catalyst**
   — Розглянути основні фази, через які проходить кожен раунд Catalyst (подача, голосування, оцінка проєктів, реалізація).
   — Ознайомитися з ролями учасників Catalyst (заявники, голосуючі, радники).

7.3.2. Практична частина

Завдання 1. Вивчення транзакцій у мережі Cardano

— Ознайомитися з транзакціями в мережі Cardano. Вибрати будь-який блокчейн-експлорер (наприклад, Cardano Explorer).
— Знайти й проаналізувати одну з останніх транзакцій: дослідити її ID, кількість переказаних монет, час проведення, адреси відправника і отримувача.
— Записати інформацію про транзакцію та зробити висновок про її особливості.

Завдання 2. Стейкінг у Cardano

— Вивчити процес стейкінгу в Cardano та знайти пули стейкінгу за допомогою https://adastat.net/uk/pools або інших ресурсів.
— Проаналізувати характеристики існуючих стейк пулів.
— Дослідити, як нараховуються винагороди за стейкінг.

Завдання 3. Ознайомлення з інтерфейсом Catalyst (додаток Catalyst Voting)

— Завантажити додаток Catalyst Voting (доступний для iOS та Android).
— Зареєструватися та ознайомитися з основними функціями програми: перегляд проєктів, участь у голосуванні, огляд результатів попередніх раундів.
— Ознайомитись з результатами голосування по 13-му раунду (Fund13).

Завдання 4. Огляд попередніх проєктів Catalyst

Використовуючи додаток Catalyst Voting або доступні джерела (наприклад, https://projectcatalyst.io/), ознайомитися із одним із успішних проєктів, що отримав фінансування у попередньому раунді. Описати цей проєкт, його цілі та результати.

Звіт повинен містити:

— Скріншот експлореру з описом інформації про транзакцію.
— Опис характеристик існуючих стейк пулів.
— Опис механізму нарахування винагороди за стейкінг.
— Скріншоти інтерфейсу Catalyst Voting з поясненнями.
— Висновок щодо результатів дослідження вибраного успішного проєкту.

— Загальні враження від роботи з платформою Catalyst, її роль у розвитку екосистеми Cardano, перспективи децентралізованого управління та залучення спільноти до розвитку блокчейну.
— Висновки про результати роботи з платформою Cardano, її переваги, можливі недоліки, та перспективи використання у майбутньому.

7.4. Контрольні питання

1. Що таке Epochs (епохи) та Slots (слоти) у контексті часу в Cardano?
2. Яка роль «Slot Leaders» (лідерів слотів) і як забезпечується безпека вибору наступного валідатора, щоб уникнути атак?
3. Як працює механізм винагород у Cardano і чому користувачам не потрібно «замикати» (lock) свої монети ADA при делегуванні?
4. Що таке параметр насичення пулу (Saturation) і як він запобігає централізації мережі?
5. Як реалізована модель децентралізованого управління (Governance) у Cardano?
6. Звідки беруться кошти у казначействі для фінансування нових проектів спільноти?

ПЕРЕЛІК ДЖЕРЕЛ

1. Nakamoto S. Bitcoin: A Peer-to-Peer Electronic Cash System. 2008. 9 p. URL: https://bitcoin.org/bitcoin.pdf (дата звернення: 05.03.2026).
2. Блокчейн і децентралізовані системи : навч. посіб. для студентів закл. вищ. освіти : у 3 ч. Ч. 1 / П. Кравченко, Б. Скрябін, О. Дубініна. Харків : ПРОМАРТ, 2021. 458 с.
3. Блокчейн і децентралізовані системи : навч. посіб. для студентів закл. вищ. освіти : у 3 ч. Ч. 2 / П. Кравченко та ін. Харків : ПРОМАРТ, 2021. 420 с.
4. Блокчейн і децентралізовані системи : навч. посіб. для студентів закл. вищ. освіти : у 3 ч. Ч. 3 / П. Кравченко та ін. Харків : ПРОМАРТ, 2021. 329 с.
5. Ковальчук Л. В., Кудін А. М., Кучинська Н. В. Вступ до технології блокчейн та криптовалют. Частина 1. Теоретичні засади функціонування блокчейн-технологій : навч. посіб. [Електронне видання]. Київ : КПІ ім. Ігоря Сікорського, 2022. 141 с.
6. Bashir I. Mastering Blockchain. Packt Publishing Ltd, 2017. 532 p.
7. Antonopoulos A. M. Mastering Bitcoin: Unlocking Digital Cryptocurrencies. O'Reilly Media, Inc., 2014. 282 p.
8. Java Cryptography Architecture (JCA) Reference Guide / Oracle. URL: https://docs.oracle.com/javase/8/docs/technotes/guides/security/crypto/CryptoSpec.html (дата звернення: 05.03.2026).
9. Package javax.crypto / Oracle. URL: https://docs.oracle.com/en/java/javase/11/docs/api/java.base/javax/crypto/package-summary.html (дата звернення: 05.03.2026).
10. Java Security Standard Algorithm Names / Oracle. URL: https://docs.oracle.com/javase/9/docs/specs/security/standard-names.html#standard-names (дата звернення: 05.03.2026).

11. Decreasing security threshold against double spend attack in networks with slow synchronization / L. Kovalchuk et al. *Computer Communications*. 2020. Vol. 154. P. 75–81.
12. Probability of double spend attack for network with non-zero synchronization time / L. Kovalchuk et al. *Proceedings of the 21th Central European Conference on Cryptology (CECC)*. 2021. P. 52–54.
13. Castro M., Liskov B. Practical Byzantine Fault Tolerance. *OSDI*. 1999. Vol. 99. P. 173–186.
14. Castro M., Liskov B. Practical Byzantine Fault Tolerance and Proactive Recovery. *ACM Transactions on Computer Systems (TOCS)*. 2002. Vol. 20, no. 4. P. 398–461.
15. Buterin V. Ethereum White Paper. *GitHub repository*. 2013. Vol. 1 (22-23). P. 5–7.
16. Офіційний сайт Ethereum : вебсайт. URL: https://ethereum.org/uk/ (дата звернення: 05.03.2026).
17. Офіційний сайт Solidity : вебсайт. URL: https://soliditylang.org/ (дата звернення: 05.03.2026).
18. Ethereum IDE : вебсайт. URL: https://remix.ethereum.org/ (дата звернення: 05.03.2026).
19. Cardano Docs : вебсайт. URL: https://docs.cardano.org/ (дата звернення: 05.03.2026).
20. Project Catalyst : вебсайт. URL: https://projectcatalyst.io/ (дата звернення: 05.03.2026).

Електронне навчальне видання комбінованого використання
Можна використовувати в локальному та мережному режимі

Родінко Марія Юріївна

ТЕХНОЛОГІЯ БЛОКЧЕЙН ТА КРИПТОВАЛЮТНІ ОПЕРАЦІЙНІ ПЛАТФОРМИ

Методичні вказівки
до лабораторних робіт з дисципліни для здобувачів вищої освіти
першого (бакалаврського) рівня за спеціальністю F3 «Комп'ютерні науки»

В авторській редакції

Підписано до розміщення 21.05.2026. Гарнітура Times New Roman.
Ум. друк. арк. 2,98. Обсяг 1,453 Мб. Зам. № 270/26.

Харківський національний університет імені В. Н. Каразіна,
61022, м. Харків, майдан Свободи, 4.
Свідоцтво суб'єкта видавничої справи ДК № 3367 від 13.01.2009
Видавництво ХНУ імені В. Н. Каразіна