Міністерство освіти і науки України
Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”
Навчально-науковий фізико технічний інститут
Кафедра інформаційної безпеки

Звіт

З практичного завдання №2
із дисципліни «Технологія блокчейн та розподілені системи»
Тема: «Реалізація смарт-контракту або анонімної криптовалюти»

Виконав:
Студент групи ФБ-41мн
Шерстюк А. В.
Варіант 13

Київ-2025

Dash

Встановлюємо dash

$ git clone https://github.com/dashpay/dash.git
$ cd dash
$ git checkout v20.x
$ ./autogen.sh
$ ./configure --enable-tests=no --disable-bench --enable-wallet --with-incompatible-bdb
$ make -j$(nproc)

Налаштування dash.conf

```
regtest=1
server=1
daemon=1
txindex=1
rpcuser=admin
rpcpassword=admin123
rpcallowip=127.0.0.1
datadir=/home/as/kpi/blockchain/regtest

[regtest]
rpcport=18443
port=18444
```

Запускаємо демон

$ ./dash/src/dashd -datadir=/home/as/kpi/blockchain/testnet

```
2025-05-26T20:33:16Z SPORK -- hash: d57979574f6994362bd890d25b28d52e30efbd4690def
       1 bestHeight: 0 peer=0 new
2025-05-26T20:33:18Z Synchronizing blockheaders, height: 24000 (~1.00%)
2025-05-26T20:33:20Z Synchronizing blockheaders, height: 26000 (~1.09%)
2025-05-26T20:33:22Z Synchronizing blockheaders, height: 28000 (~1.17%)
2025-05-26T20:33:25Z Synchronizing blockheaders, height: 30000 (~1.26%)
```

$ ./src/dash-cli -datadir=/home/as/kpi/blockchain/testnet getblockcount

```
as@as-win:~/kpi/blockchain/dash$ ./src/dash-cli -datadir=/home/as/kpi/blockchain/testnet getblockcount
0
```

Перевіряємо

$ ./src/dash-cli -datadir=/home/as/kpi/blockchain/testnet getblockchaininfo

```
as@as-win:~/kpi/blockchain/dash$ ./src/dash-cli -datadir=/home/as/kpi/blockchain/testnet getblockchaininfo
{
  "chain": "main",
  "blocks": 912,
  "headers": 2084000,
  "bestblockhash": "00000349d7f755d397cd39548116e31aa74be1a0d881375ea4f3ee541bcba851",
  "difficulty": 0.0009765625,
  "mediantime": 1390105768,
  "verificationprogress": 1.633393492574165e-05,
  "initialblockdownload": true,
  "chainwork": "0000000000000000000000000000000000000000000000000078407840",
  "size_on_disk": 216315,
```

Майнінг у тестовій мережі

./src/dash-cli -datadir=/home/as/kpi/blockchain/testnet createwallet "testwallet"

./src/dash-cli -datadir=/home/as/kpi/blockchain/testnet getnewaddress

```
make[1]: Leaving directory '/home/as/kpi/blockchain/dash'
as@as-win:~/kpi/blockchain/dash$ ./src/dash-cli -datadir=/home/as/kpi/blockchain/testnet createwallet "testwallet"
{
  "name": "testwallet",
  "warning": ""
}
as@as-win:~/kpi/blockchain/dash$ ./src/dash-cli -datadir=/home/as/kpi/blockchain/testnet getnewaddress
XtPUnSKwvQJW7tvff1xX21ogmzytGMjQLe
```

./src/dash-cli -datadir=/home/as/kpi/blockchain/testnet generatetoaddress 1 XtPUnSKwvQJW7tvff1xX21ogmzytGMjQLe

```
as@as-win:~/kpi/blockchain/dash$ ./src/dash-cli -datadir=/home/as/kpi/blockchain/testnet generatetoaddress 1 XtPUnSKwvQJW7tvff1xX21ogmzytGMjQLe
error code: -1
error message:
CreateNewBlock: TestBlockValidity failed: bad-fork-prior-to-checkpoint
```

CreateNewBlock: TestBlockValidity failed: bad-fork-prior-to-checkpoint
Dash не дозволяє самостійно майнити блоки на загальнодоступному testnet після певного моменту- це захист від форків. Далі будуть скріни з regtest.

Мінімум 101 блок, щоб винагорода стала доступною для витрат (100 підтверджень).

$ ./src/dash-cli -datadir=/home/as/kpi/blockchain/regtest generatetoaddress 101 $ADDR

```
as@as-win:~/kpi/blockchain/dash$ ./src/dash-cli -datadir=/home/as/kpi/blockchain/regtest generatetoaddress 101 $ADDR
[
  "217dc274a09d5ae05b40ce69cc11d7a757906767140d21f69243e29a4a8c5c7c",
  "0bf532a5a9e590bee68be97904fdbcae7a048ec199fd2e4a5a7961346cbdf418",
  "4ac1f44744d3cbc370e9933f3f38234bcef1ab21147be54bf3f91feb0e9f7b61",
  "3f371a74d05831a6622abae93034f82d5b8d066bd49d669c0d3e0d5f6ce7fa85",
  "7a17961aa28e0576eda9c5965016358ca73f9ec80782d8a1f63917a2aa9a9249",
  "7200410b42b1417b1c5581d49b21d6dffa8c146e6ea2cf0dd84e04c730dd9ef4",
```

```
as@as-win:~/kpi/blockchain/dash$ ./src/dash-cli -datadir=/home/as/kpi/blockchain/regtest getbalance
500.00000000
```

Логи зберігаються у файлі /home/as/kpi/blockchain/regtest/**debug.log**

$ less /home/as/kpi/blockchain/regtest/debug.log

```
2025-05-26T20:55:41Z Dash Core version v20.0.4 (release build)
2025-05-26T20:55:41Z Assuming ancestors of block 000000000000000c8b7a3bdcd8b9f516462122314529c8342244c685a4c899bf have valid signatures.
2025-05-26T20:55:41Z Setting nMinimumChainWork=0000000000000000000000000000000000000000000000008d970bc6cda0b02b30fc
2025-05-26T20:55:41Z fDisableGovernance 0
2025-05-26T20:55:41Z Using the 'x86_shani(1way,2way)' SHA256 implementation
2025-05-26T20:55:41Z Using RdSeed as additional entropy source
2025-05-26T20:55:41Z Using RdRand as an additional entropy source
2025-05-26T20:55:41Z Default data directory /home/as/.dashcore
2025-05-26T20:55:41Z Using data directory /home/as/kpi/blockchain/regtest
2025-05-26T20:55:41Z Config file: /home/as/kpi/blockchain/regtest/dash.conf
2025-05-26T20:55:41Z Config file arg: [regtest] daemon="1"
```

Dash використовує функцію PrivateSend, яка реалізує змішування транзакцій (CoinJoin-подібний підхід).

Поточна реалізація системи дозволяє збільшити анонімність транзакцій, шляхом об'єднання декількох входів від різних користувачів в одній транзакції з декількома виходами. Це приховує потоки руху коштів і обмежує можливості прямого відстеження транзакцій. Для відстеження таких транзакцій був запропонований механізм ієрархічного списку (заснованого на входах і виходах), проте практичних доказів можливості такого аналізу представлено не було.

Механізм PrivateSend включає ряд процедур:

- Попередня деномінація- платежі дробляться на однакові частини: 100, 10, 1, 0.1, що перешкоджає відстеження за індивідуальними сумами.
- Кожна частина проходить свої власні етапи анонімізації, при цьому
  - анонімізується не вся сума, а частини;
  - перемішуються виключно однакові за величиною частини;
  - на кожному етапі обирається нова перемішуюча мастернода.
- Перемішування відбувається завчасно. Після змішування сума повертається власнику на нові анонімні адреси і може використовуватися, коли буде потреба (немає потреби чекати змішування).

З ростом числа користувачів DASH стає важче позбутися невизначеності в структурах змішування

Деанонімізація можлива через:

- Аналіз блокчейну (Dash regtest має відкритий блокчейн)
  $ ./src/dash-cli -datadir=/home/as/kpi/blockchain/regtest listtransactions
  $ ./src/dash-cli -datadir=/home/as/kpi/blockchain/regtest getrawtransaction <txid> 1

```
as@as-win:~/kpi/blockchain/dash$ ./src/dash-cli -datadir=/home/as/kpi/blockchain/regtest listtransactions
[
  {
    "address": "ygobFhLG2k5S5yTnDWQyAAuDhsrGRa4Lje",
    "category": "immature",
    "amount": 500.00000000,
    "label": "",
    "vout": 0,
    "confirmations": 10,
    "instantlock": false,
    "instantlock_internal": false,
    "chainlock": false,
    "generated": true,
    "blockhash": "1777d7aafd8cf9ae6ffd27beaa86767c2e29c3c5ba1feee1ac7037aa897dab08",
    "blockheight": 92,
    "blockindex": 0,
    "blocktime": 1748294335,
    "txid": "6ebb2857a6e3fe27fff9a554082a63863d661ec96360a5ce5ff6f0370b07f348",
    "walletconflicts": [
    ],
    "time": 1748294319,
    "timereceived": 1748294319
  },
```

```
as@as-win:~/kpi/blockchain/dash$ ./src/dash-cli -datadir=/home/as/kpi/blockchain/regtest getrawtransacti
{
  "txid": "7dba9bd2459570b6b3709c26858089224f0afb055c8bc168711735f45bece831",
  "version": 2,
  "type": 0,
  "size": 89,
  "locktime": 0,
  "vin": [
    {
      "coinbase": "01650101",
      "sequence": 4294967295
    }
  ],
  "vout": [
    {
      "value": 500.00000000,
      "valueSat": 50000000000,
      "n": 0,
      "scriptPubKey": {
        "asm": "OP_DUP OP_HASH160 e0b52dc547f6aac250774acec66e49890b087bed OP_EQUALVERIFY OP_CHECKSIG",
        "hex": "76a914e0b52dc547f6aac250774acec66e49890b087bed88ac",
        "reqSigs": 1,
        "type": "pubkeyhash",
        "addresses": [
          "ygobFhLG2k5S5yTnDWQyAAuDhsrGRa4Lje"
        ]
      }
    }
  ],
  "hex": "020000000100000000000000000000000000000000000000000000000000000000000000000000ffffffff0401650101ff
",
  "blockhash": "5c3376c2b830aa8b36176804eb6f2b166653f1ddad58e44ae510629f8f9ded84",
  "height": 101,
  "confirmations": 1,
  "time": 1748294336,
  "blocktime": 1748294336,
  "instantlock": false,
  "instantlock_internal": false,
  "chainlock": false
}
```

- Виявлення змішування:
    - У змішаних транзакціях видно однакові номінали (наприклад, 0.1 DASH, 1 DASH).
    - Виходи мають однакову структуру → легко виділити з загальної маси.
    - Застосовується "cluster analysis":
    - Виокремлення однакових виходів у транзакціях.
    - Пошук збігів по часі та IP (якщо є доступ до нодів).
- BlockSci (неофіційна підтримка Dash).
- Власний парсер блоків через getblock, getrawtransaction, decoderawtransaction.

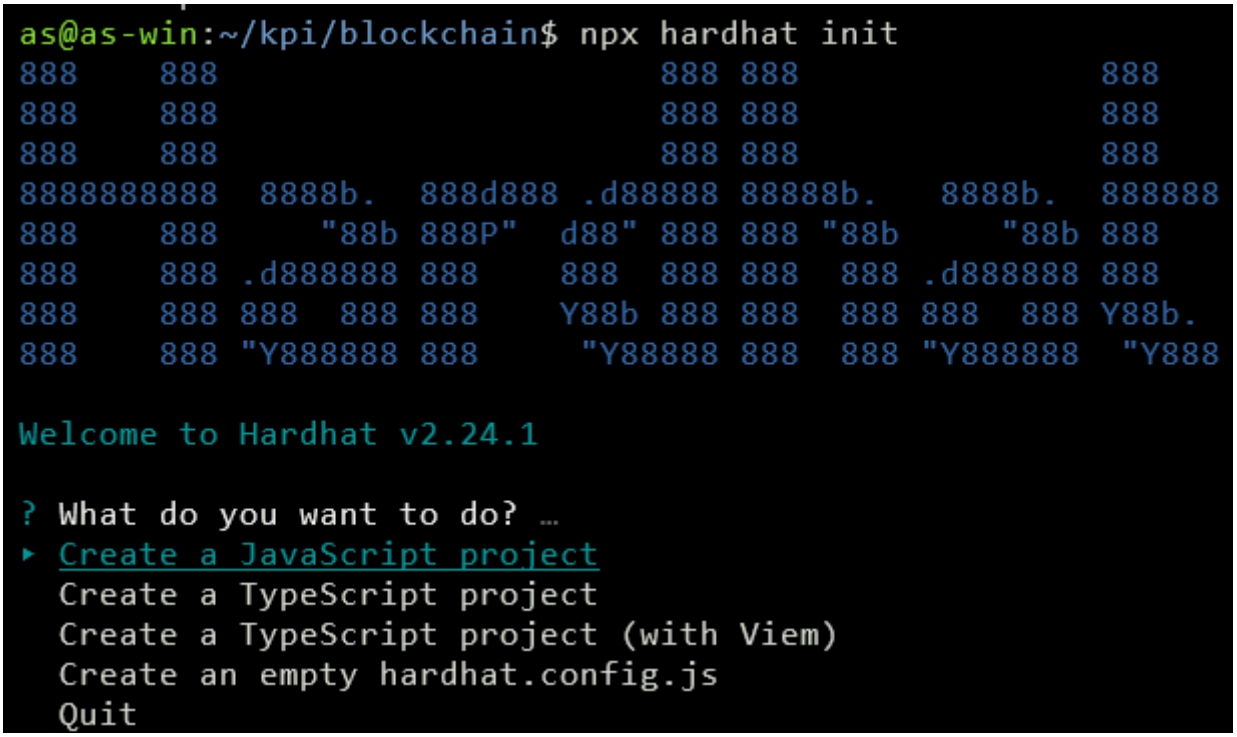
розгортання та запуск обраного смарт-контракту, підвищення ефективності роботи смарт-контракту з точки зору витрати гасу

Встановлюємо плагін для вс код (опціонально)

[Solidity - Visual Studio Marketplace]

Встановлюємо Hardhat
$ npm install --save-dev hardhat

Створюємо тестовий проект
$ npx hardhat init



Створюємо 2 контракти:

- неефективний

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.28;

contract UnoptimizedUserRegistry {
    // Неефективне зберігання даних
    mapping(address => string) public userNames;
    mapping(address => uint256) public userAges;
    mapping(address => bool) public isActive;
    mapping(address => uint256) public registrationTime;
    mapping(address => string) public userEmails;
    mapping(address => uint8) public userLevels;

    // Масиви для зберігання всіх користувачів
    address[] public allUsers;
```

```
    string[] public allNames;

    uint256 public totalUsers;
    uint256 public totalActiveUsers;

    event UserRegistered(address user, string name);
    event UserUpdated(address user);

    // Неефективна реєстрація користувача
    function registerUser(
        string memory _name,
        uint256 _age,
        string memory _email
    ) external {
        require(bytes(_name).length > 0, "Name cannot be empty");
        require(_age > 0, "Age must be positive");
        require(bytes(_email).length > 0, "Email cannot be empty");

        // Кілька записів у storage
        userNames[msg.sender] = _name;
        userAges[msg.sender] = _age;
        userEmails[msg.sender] = _email;
        isActive[msg.sender] = true;
        registrationTime[msg.sender] = block.timestamp;
        userLevels[msg.sender] = 1;

        // Неефективне додавання в масиви
        allUsers.push(msg.sender);
        allNames.push(_name);

        // Окремі оновлення лічильників
        totalUsers = totalUsers + 1;
        totalActiveUsers = totalActiveUsers + 1;

        emit UserRegistered(msg.sender, _name);
    }

    // Неефективне оновлення кількох полів
    function updateUser(
        string memory _name,
        uint256 _age,
        string memory _email,
        uint8 _level
    ) external {
        require(bytes(userNames[msg.sender]).length > 0, "User not registered");

        // Кілька записів у storage
```

```
        userNames[msg.sender] = _name;
        userAges[msg.sender] = _age;
        userEmails[msg.sender] = _email;
        userLevels[msg.sender] = _level;

        emit UserUpdated(msg.sender);
    }

    // Неефективний пошук серед усіх користувачів
    function findUserByName(string memory _name) external view returns (address) {
        for (uint256 i = 0; i < allUsers.length; i++) {
            if (keccak256(bytes(userNames[allUsers[i]])) == keccak256(bytes(_name)))
{
                return allUsers[i];
            }
        }
        return address(0);
    }

    // Неефективне отримання інформації про користувача
    function getUserInfo(address _user) external view returns (
        string memory name,
        uint256 age,
        string memory email,
        bool active,
        uint256 regTime,
        uint8 level
    ) {
        // Кілька читань із storage
        name = userNames[_user];
        age = userAges[_user];
        email = userEmails[_user];
        active = isActive[_user];
        regTime = registrationTime[_user];
        level = userLevels[_user];
    }

    // Неефективне пакетне оновлення
    function batchUpdateLevels(address[] memory _users, uint8[] memory _levels)
external {
        require(_users.length == _levels.length, "Arrays length mismatch");

        // Неоптимізований цикл
        for (uint256 i = 0; i < _users.length; i++) {
            require(bytes(userNames[_users[i]]).length > 0, "User not registered");
            userLevels[_users[i]] = _levels[i];
        }
```

```
    }

    // Неефективне деактивування користувача
    function deactivateUser(address _user) external {
        require(isActive[_user], "User already inactive");

        isActive[_user] = false;
        totalActiveUsers = totalActiveUsers - 1;
    }

    // Неефективний підрахунок активних користувачів певного рівня
    function countActiveUsersByLevel(uint8 _level) external view returns (uint256
count) {
        for (uint256 i = 0; i < allUsers.length; i++) {
            if (isActive[allUsers[i]] && userLevels[allUsers[i]] == _level) {
                count++;
            }
        }
    }
}
```

- оптимізований

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.28;

contract OptimizedUserRegistry {
    // Оптимізоване зберігання даних у структурі
    struct User {
        uint128 age;                // Упаковується в один слот із registrationTime
        uint128 registrationTime;
        uint8 level;               // Упаковується з isActive в один слот
        bool isActive;
        // bytes16 заповнення залишається в цьому слоті
    }

    // Основне сховище користувачів
    mapping(address => User) public users;
    mapping(address => bytes32) public userNames;    // bytes32 замість string
    mapping(address => bytes32) public userEmails;   // bytes32 замість string

    // Для пошуку за ім'ям — більш ефективний індекс
    mapping(bytes32 => address) public nameToUser;

    // Оптимізовані лічильники
    uint128 public totalUsers;
```

```
    uint128 public totalActiveUsers; // Упаковуються в один слот

    // Масив лише адрес (без дублювання даних)
    address[] public allUsers;

    event UserRegistered(address indexed user, bytes32 name);
    event UserUpdated(address indexed user);

    // Ефективна реєстрація користувача
    function registerUser(
        bytes32 _name,
        uint128 _age,
        bytes32 _email
    ) external {
        require(_name != bytes32(0), "Name cannot be empty");
        require(_age > 0, "Age must be positive");
        require(_email != bytes32(0), "Email cannot be empty");
        require(users[msg.sender].registrationTime == 0, "User already registered");
        require(nameToUser[_name] == address(0), "Name already taken");

        // Один запис структури
        users[msg.sender] = User({
            age: _age,
            registrationTime: uint128(block.timestamp),
            level: 1,
            isActive: true
        });

        userNames[msg.sender] = _name;
        userEmails[msg.sender] = _email;
        nameToUser[_name] = msg.sender;

        allUsers.push(msg.sender);

        // Оптимізоване оновлення лічильників
        unchecked {
            ++totalUsers;
            ++totalActiveUsers;
        }

        emit UserRegistered(msg.sender, _name);
    }

    // Ефективне оновлення даних користувача
    function updateUser(
        bytes32 _name,
        uint128 _age,
```

```
        bytes32 _email,
        uint8 _level
    ) external {
        User storage user = users[msg.sender];
        require(user.registrationTime != 0, "User not registered");

        // Оновлення імені тільки при зміні
        bytes32 oldName = userNames[msg.sender];
        if (oldName != _name) {
            require(nameToUser[_name] == address(0), "Name already taken");
            delete nameToUser[oldName];
            userNames[msg.sender] = _name;
            nameToUser[_name] = msg.sender;
        }

        // Одне оновлення структури
        user.age = _age;
        user.level = _level;
        userEmails[msg.sender] = _email;

        emit UserUpdated(msg.sender);
    }

    // Ефективний пошук за ім'ям через індекс
    function findUserByName(bytes32 _name) external view returns (address) {
        return nameToUser[_name];
    }

    // Ефективне отримання інформації про користувача
    function getUserInfo(address _user) external view returns (
        bytes32 name,
        uint128 age,
        bytes32 email,
        bool active,
        uint128 regTime,
        uint8 level
    ) {
        User memory user = users[_user]; // Одне читання структури зі storage
        return (
            userNames[_user],
            user.age,
            userEmails[_user],
            user.isActive,
            user.registrationTime,
            user.level
        );
    }
```

```
    // Оптимізоване пакетне оновлення
    function batchUpdateLevels(address[] calldata _users, uint8[] calldata _levels)
external {
        require(_users.length == _levels.length, "Arrays length mismatch");

        uint256 length = _users.length;

        // Оптимізований цикл
        for (uint256 i; i < length;) {
            address userAddr = _users[i];
            require(users[userAddr].registrationTime != 0, "User not registered");
            users[userAddr].level = _levels[i];

            unchecked { ++i; }
        }
    }

    // Ефективне деактивування користувача
    function deactivateUser(address _user) external {
        User storage user = users[_user];
        require(user.isActive, "User already inactive");

        user.isActive = false;
        unchecked { --totalActiveUsers; }
    }

    // Ефективний підрахунок із кешуванням довжини масиву
    function countActiveUsersByLevel(uint8 _level) external view returns (uint256
count) {
        uint256 length = allUsers.length;

        for (uint256 i; i < length;) {
            User memory user = users[allUsers[i]];
            if (user.isActive && user.level == _level) {
                unchecked { ++count; }
            }
            unchecked { ++i; }
        }
    }

    // Додаткова утилітарна функція для конвертації string у bytes32
    function stringToBytes32(string memory _str) external pure returns (bytes32) {
        bytes memory tempBytes = bytes(_str);
        require(tempBytes.length <= 32, "String too long");

        bytes32 result;
```

```
        assembly {
            result := mload(add(tempBytes, 32))
        }
        return result;
    }

    // Конвертація bytes32 назад у string
    function bytes32ToString(bytes32 _bytes) external pure returns (string memory) {
        uint256 length;
        for (uint256 i; i < 32;) {
            if (_bytes[i] != 0) {
                unchecked { ++length; }
            } else {
                break;
            }
            unchecked { ++i; }
        }

        bytes memory result = new bytes(length);
        for (uint256 i; i < length;) {
            result[i] = _bytes[i];
            unchecked { ++i; }
        }

        return string(result);
    }
}
```

Запускаємо ноду

$ npx hardhat node

Проводимо тестування

$ REPORT_GAS=true npx hardhat test

(index)	Values
Deploy (Unopt)	1106858
Deploy (Opt)	926228
Register (Unopt)	292683
Register (Opt)	201598
Update (Unopt)	46028
Update (Opt)	66820
BatchUpdate (Unopt)	39855
BatchUpdate (Opt)	39145