# Біткоїн та криптовалютні технології
# Лекція 5: транзакції

Юрій Жикін

14 жовтня, 2022

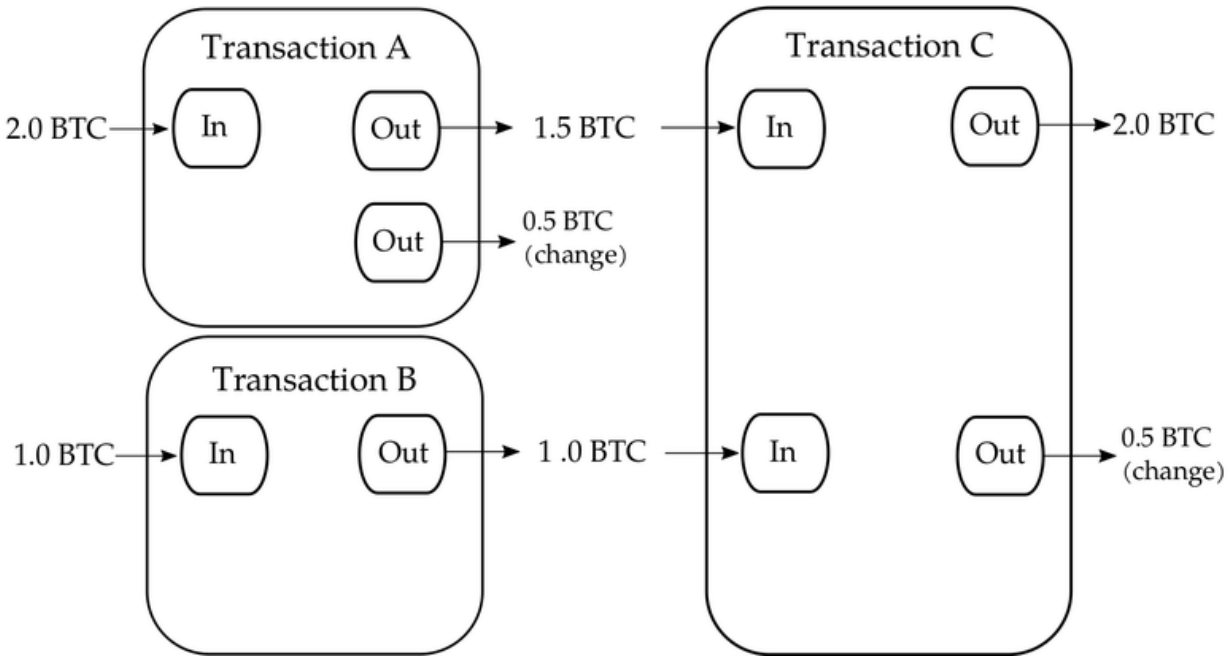
# Структура транзакції

- **Транзакція**
  - **версія**
  - **входи** (список структур **Транзакційний вхід**)
  - **виходи** (список структур **Транзакційний вихід**)
  - **свідки** (список структур **Свідок**)
  - **час блокування**
- **Транзакійний вхід**
  - **ідентифікатор попередньої транзакції**
  - **індекс попереднього виходу**
  - **програма-відмикання**
- **Транзакційний вихід**
  - **кількість**
  - **програма-замикання**
- **Свідок**
  - **елементи свідка** (список елементів свідка)

# Передача власності над біткоїном 1/2

- Невикористані транзакційні виходи (**UTXO**) - це записи про володіння “шматками” біткоїна: певними кількостями **сатоші**, прив’язаними до власника за допомогою програми-замикання.
- Транзакції передають власність над “шматками” біткоїна, *знищуючи* ці “шматки” (посилаючись на відповідні *невикористані виходи* через *входи*, що відмикають програми-замикання), та створюючи нові *невикористані виходи*.
- Сукупність всіх *невикористаних транзакційних виходів*, які існують в даний момент часу - це весь біткоїн, який існує в системі.

# Передача власності над біткоїном 2/2



# Біткоїн Скрипт

- **Біткоїн Скрипт**, або просто **Скрипт** - це **стекова Forth-подібна Тюрінг-неповна** мова програмування для формулювання логіки замикання/відмикання транзакційних виходів.
- Скрипт дозволяє формулювати довільні умови використання кожного окремого “шматка” біткоїна.
- Завдяки системі *“доказу виконаної роботи”*, Біткоїн є першою **децентралізованою** грошовою системою, але завдяки системі *скриптування*, Біткоїн також є першою **програмованою** грошовою системою.

# Стекові мови програмування

- **Стекове програмування** - це парадигма програмування, яка базується на моделі **стекової машини** для передачі параметрів.
- Приклад:

1. Програма
```
3 5 add 3 mul;
```
   Дані
```
;
```
2. Програма
```
5 add 3 mul;
```
   Дані
```
3;
```
3. Програма
```
add 3 mul;
```
   Дані
```
5 3;
```
4. Програма
```
3 mul;
```
   Дані
```
8;
```
5. Програма
```
mul;
```
   Дані
```
3 8;
```
6. Програма
```
;
```
   Дані
```
24;
```

# Тюрінг-неповнота

- *Скрипт* - це *навмисно* Тюрінг-неповна мова програмування.
- У Скрипті відсутній один з основних інструментів сучасних мов програмування: **цикл**.
- Скрипти у транзакціях виконуються кожною повноцінною нодою в мережі, і тому цикли могли б використовуватись як засіб завантаження мережі (здійснення DoS-атак).
- Програми з циклами значно гірше піддаються статичному аналізу (тобто аналізу, який "розглядає" програму, але не виконує її).
- Мережа Етереум використовує Тюрінг-повну мову **Solidity**, що є безпсереднім чинником найгірших інцидентів безпеки мережі за всю історію існування Етереум-мережі.

# Операції у Біткоїн Скрипті 1/3

- Виконаня Скрипт-програм - це основна складова процесу перевірки транзакцій.
- *Інтерпретатор Скрипта* складається зі стеку команд та стеку даних.
- Для кожного *входу* транзакції, спочатку виконується його програма-відмикання, а отриманий стек даних використовується для подальшого виконання програми-замикання відповідного *виходу*:
  - створюємо порожній стек $S_0 = S_{empty}$
  - виконуємо програма-відмикання входу на стеку $S_0$:

$$S_1 = Execute(UnlockScript, S_0)$$

  - виконуємо програма-замикання відповідного виходу на стеку $S_1$:

$$S_2 = Execute(LockScript, S_1)$$

  - перевіряємо, чи верхній елемент на стеку $S_2$ - не *False* (“неправда”).

# Операції у Біткоїн Скрипті 2/3

- Значення на стеку даних - це послідовності байтів, але вони можуть інтерпретуватись, як числа, коли це необхідно.
- Значення *False* (“неправда”) представляється числом 0, яке в свою чергу представляється порожньою послідовністю байтів, бо послідовністю з одного байта $[0x80]$.
- Будь-яке значення, що не є *False*, вважається значенням *True* (“правда”), тобто будь-яка послідовність байтів, яка відрізняється від [] та $[0x80]$ на верхівці стека даних після виконання скриптів означає, що право власності доведена для даного “шматка” біткоїна.
- Виконання скрипта також може завершитись помилкою, що прирівнюється до значенна *False*, тобто власність недоведена, а транзакція - неправильна.

# Операції у Біткоїн Скрипті 3/3

- Операції Скрипта поділяються на наступні категорії:
  - **константи** - розміщення даних у стеку
  - **контроль виконання** - умовні операції, а також
    - `OP_VERIFY` - помилка, якщо на верхівці стека - не *True*
    - `OP_RETURN` - завжди помилка (використовується для розміщення довільних даних всередині транзакції: зображень, тощо)
  - **маніпуляція стеком** - викидання, копіювання, перестановка елементів стека
  - **бітова логіка та арифметика**
  - **криптографія** - криптографічні операції (хеш-функції)
    - `OP_CHECKSIG` - перевірка підпису для заданого публічного ключа
    - `OP_CHECKMULTISIG` - перевірка декількох підписів для декількох публічних ключів (контракти типу "*N* з *M* власників")
  - **блокування** - перевірка умов блокування транзакцій

# Стандартні Скрипти-програми 1/4

- **P2PKH** - pay-to-pubkey-hash (платіж за хешем публічного ключа)

  Замикання `OP_DUP OP_HASH160 <pubKeyHash> OP_EQUALVERIFY OP_CHECKSIG;`

  Відмикання `<sig> <pubKey>;`

- Виконання P2PKH-скриптів

1. Програма `<sig> <pubKey>;`
   Дані `;`
2. Програма `<pubKey>;`
   Дані `<sig>;`
3. Програма `;`
   Дані `<pubKey> <sig>;`

# Стандартні Скрипт-програми 2/4

- Виконання P2PKH-програми

| | | |
|---|---|---|
| 1. | Програма | `OP_DUP OP_HASH160 <pubKeyHash> OP_EQUALVERIFY OP_CHECKSIG;` |
| | Дані | `<pubKey> <sig>;` |
| 2. | Програма | `OP_HASH160 <pubKeyHash> OP_EQUALVERIFY OP_CHECKSIG;` |
| | Дані | `<pubKey> <pubKey> <sig>;` |
| 3. | Програма | `<pubKeyHash> OP_EQUALVERIFY OP_CHECKSIG;` |
| | Дані | `<pubKeyHash> <pubKey> <sig>;` |
| 4. | Програма | `OP_EQUALVERIFY OP_CHECKSIG;` |
| | Дані | `<pubKeyHash> <pubKeyHash> <pubKey> <sig>;` |
| 5. | Програма | `OP_CHECKSIG;` |
| | Дані | `<pubKey> <sig>;` |
| 6. | Програма | `;` |
| | Дані | `True;` |

# Стандартні Скрипт-програми 3/4

- **P2PK** - pay-to-pubkey (платіж за публічним ключем, більше не використовується, бо публічний ключ потрапляє в ланцюг блоків занадто рано)

  Замикання `<pubKey> OP_CHECKSIG;`

  Відмикання `<sig>;`

- **P2MS** - багатопідписні транзакції ("$M$ з $N$ власників")

  Замикання `<M> <pk1> ... <pkN> <N> OP_CHECKMULTISIG;`

  Відмикання `OP_0 <sig1> ... <sigM>;`

- **P2SH** - pay-to-script-hash (платіж за хешем скрипта) - зміна протоколу, впроваджена у 2012 році, яка дозволила створювати довільні скрипти-замикання, які при цьому мають фіксований розмір та визначену адресу

  Замикання `OP_HASH160 <scriptHash> OP_EQUAL;`

  Відмикання `<customLockScript...> <serializedRedeemScript>;`

# Стандартні Скрипт-програми 4/4

- **P2SH** потребував зміни правил виконання *Скрита* за допомогою *м'якого розгалуження*:
  - виконуємо скрипт-відмикання, отримуємо `<serializedRedeemScript>` на верхівці стека
  - виконуємо скрипт-замикання, перевіряючи, що закодований скрипт `<serializedRedeemScript>` відповідає хешу `<scriptHash>`
  - учасники мережі з **старою** версією протоколу вважають, що транзакція правильна
  - учасники мережі з **новою** версією протоколу продовжують перевірку: декодують скрипт `<serializedRedeemScript>` і виконують його, так ніби це справжній скрипт-замикання

# Види розгалужень у мережі

- **М'яке розгалуження** (англ. *softfork*) робить правила **більш строгими**, тобто старі учасники зі старими правилами вважають нові дані завжди правильними, а нові учасники здійснюють додаткові перевірки
- **Жорстке розгалуження** (англ. *hardfork*) робить правила **менш строгими**, тобто старі учасники відкинуть дані, як неправильні, але нові учасники вважають ці дані правильними, що розділяє мережу на дві частини, і тому всі учасники повинні оновити набір правил, для того, щоб мережа продовжила працювати
- **Жорстке розгалуження** "викидає" старих учасників мережі з протоколу

# Нестандартні Скрипт-програми

- **Задача SHA256** - “шматок” біткоїна може використати будь-хто, хто надасть таку послідовність байтів $s$, що $h = SHA256(s)$

  Замикання `OP_HASH256 <h> OP_EQUAL;`

  Відмикання `<s>;`

- **Задача колізії SHA1** - створена Пітером Тодом в 2013 році для заохочення пошуку колізій для хеш-функції SHA1, яка вже на той час вважалась небезпечною; 2.48 біткоїнів винагороди хтось забрав у 2017 році:

  Замикання `OP_2DUP OP_EQUAL OP_NOT OP_VERIFY OP_SHA1 OP_SWAP OP_SHA1 OP_EQUAL;`

  Відмикання `<preimage1> <preimage2>;`

# Біткоїн-адреси 1/2

- Для *стандартних* виходів транзакцій (тобто виходів, що мають *стандартну* програму-замикання), існують визначені формати “адрес”.
- Біткоїн-адреси - це порівняно невеликі ідентифікатори, які однозначно задають ключову інформацію у програмі-замиканні, і можуть бути використані, щоб ідентифікувати і/або відтворити відповідну програму-замикання
  - для *P2PKH*, це `<pubKeyHash>`:

$$A_{P2PKH} = Encode_{Base58Check}(HASH160(pubkey))$$

  - для *P2SH*, це `<scriptHash>`:

$$A_{P2SH} = Encode_{Base58Check}(HASH160(redeemscript))$$

# Біткоїн-адреси 2/2

- Для того, щоб уникнути будь-яких неоднозначностей і зменшити ймовірність помилок, Біткоїн-адреси використовують спеціальне кодування **Base58Check**:

$$Base58Check(t, s) = Base58(t + s + HASH256(t + s)[0 : 4])$$

- **Base58**-кодування схоже на 64-кове кодування (**base64**), але навмисне не використовує символи, які можна легко сплутати з іншими: 0, O, I, and l.
- Значення $t$ використовується, щоб ідентифікувати тип інформації, яка кодується:

| | | |
|---|---|---|
| 0 | 1 | P2PKH-адреси |
| 5 | 3 | P2SH-адреси |
| 111 | m або n | P2PKH-адреси у тестовій мережі |
| 196 | 2 | P2SH-адреси у тестовій мережі |

# Біткоїн-гаманець 1/2

- **То що ж таке Біткоїн-гаманець?**
- В загальному випадку, Біткоїн-гаманець - це будь-яка інформація, використовуючи яку, можна створити програму-відмикання для деякого *невикористаного транзакційного виходу*.
- Оскільки більшість транзакційних виходів є *стандартними* і потребують криптографічних підписів власників, типовий Біткоїн-гаманець - це пристрій чи програмне забезпечення для зберігання криптографічних ключів.
- Коли користувач хоче отримати біткоїн, він за допомогою Біткоїн-гаманця генерує нову пару криптографічних ключів $(p_i, P_i)$ і обчислює нову P2PKH-адресу $A_i$ наступним чином

$$A_i = Encode_{Base58Check}(HASH160(P_i))$$

# Біткоїн-гаманець 2/2

- Цю адресу отримувач передає надсилачеві.
- Гаманець надсилача обчислює $h = Decode_{Base58Check}(A_i)$ і створює транзакцію, що містить *вихід*, в якому вказана відповідна кількість, а програма-замикання якого містить вказаний хеш $h$, який у вигляді адреси було передано отримувачем

```
OP_DUP OP_HASH160 <h> OP_EQUALVERIFY OP_CHECKSIG;
```

- Гаманець надсилача знаходить у своєму сховищі криптографічні ключі, що відповідають *виходам*, які він "витрачає", і обчислює відповідні підписи для того, щоб створити відповідні програми-відмикання.
- Заповнена транзакція публікується, розходиться по мережі, потрапляє у блок і підтверджується.
- Тепер отримувач має право власності над *виходом*, створеним надсилачем, і може аналогічним чином надіслати цей *вихід* далі.

# Кінець

Дякую за увагу!