EE 374: Fundamentals of Blockchain Infrastructure **Stanford, Spring 2025**

Lecture 4: Bitcoin Transactions and Bitcoin Script

April 9, 2025

Lecturer: Robin Linus Scribe: Matthew Sato

1 Introduction

In this lecture, two main topics are discussed: Bitcoin transactions and Bitcoin scripts. The differences between a typical account model and UTXO (coin) model are discussed. Then, an introduction to the Bitcoin scripting language is provided with several examples.

2 Bitcoin Transactions

Bitcoin transactions are the mechanism by which bitcoins can be transferred from one owner to another. Before diving into the Bitcoin model, let's consider several requirements that guide the design of a transaction:

- **sender**: the current owner of the coin
- **recipient**: the receiver of the coin
- **amount**: the amount of Bitcoin to receive
- **signature**: a cryptographic signature to authenticate the transaction

Two distinct transaction models are commonly used to keep records for payments and tracking a user's balance. The *Account Model* represents a typical balance sheet, but Bitcoin **does not** use the Account Model. Instead, Bitcoin uses the *unspent transaction output (UTXO) model.*

2.1 The Account Model

The account model uses a ledger which assigns a single entry per user. Each user has a unique account which stores their current balance, demonstrated in Table 1. Transactions modify the balance directly, similar to how a bank tracks user balances. The account model is simple and straightforward, but is vulnerable to replay attacks. Blockchains such as Ethereum use the account model, but Bitcoin does not.

2.2 The UTXO (Coin) Model

The UTXO model uses a single entry per coin (rather than per user in the account model) as shown in Table 2. Users can own multiple coins and must refer to specific coins when spending. In a UTXO transaction, coins used as inputs are consumed and cannot be used in future transactions. The result of a transaction are new coins.

Name	Balance
Alice	42 ₿
Bob	23 ₿
Carol	7 ₿
Dave	59 ₿

Table 1: The account model for transactions.

Coin	Owner	Amount
1	Alice	11 ₿
2	Bob	29 ₿
3	Carol	17 ₿
4	Alice	5 ₿

Table 2: The UTXO model for transactions.

Since Bitcoin uses the UTXO model, let's examine the UTXO model in more detail with several examples. Figure 1a shows the framework for a typical Bitcoin transcation. In the most simple form, the input of the transaction is the `coin_id` and the `signature` of the owner of the coin. Meanwhile, the output is the `next_owner` of the coin and the `amount` of the coin they are given. In Figure 1b, a discrete example is provided whereas Alice starts with 2 ₿ and transfers 1.5 ₿ to Bob and 0.5 ₿ to herself. Any number of inputs or outputs can be listed in the Bitcoin transaction so long as they fit within the maximum block size of 4 MB.

(a) Most Simple TX

in	out
`coin_id`	`next_owner`
`signature`	`amount`

(b) Typical Bitcoin TX

in	out
	1.5 ₿
	Bob
2 ₿	
Alice	
	0.5 ₿
	Alice

Figure 1: A simplified version of a Bitcoin transaction.

In the UTXO model, the `coin_id` is represented in the form <`hash`>:<`index`>, where `hash` is the hash of the Bitcoin transaction and `index` is the index of the transaction. For example, in Figure 1b, the index for Bob's 1.5 ₿ is 0 and the index for Alice's 0.5 ₿ is 1. An example using this format for `coin_id` is shown in Table 3. Now that the UTXO model has been defined, let's look at a concrete example.

Example *A Bitcoin Transaction using the UTXO Model.*
Consider that Alice mines a block and receives 2 ₿ as reward for mining the block. Then, Alice transfers 1.5 ₿ to Bob. Each of these events are represented as two separate transactions, TX_1 and TX_2. These transactions and the corresponding UTXO model are shown in Figure 2. TX_1 does not have any input since Alice received the bitcoin as reward for mining the block. This transaction is

Coin	Owner	Amount
$7f5d8 \cdots 5a : 1$	Alice	26 ₿
$4bc96 \cdots dc : 0$	Bob	34 ₿
$89c5d \cdots 4a : 2$	Carol	46 ₿
$e6f5c \cdots a3 : 1$	Alice	18 ₿

Table 3: The UTXO model using the proper identification for coins.

(a) TX_1

in	out
	Alice 2 ₿

(b) TX_2

in	out
Alice 2 ₿	Bob 1.5 ₿
	Alice 0.5 ₿

(c) UTXO Model

ID	Owner	Amount
$TXID_1$:0	Alice	2 ₿
$TXID_2$:0	Bob	1.5 ₿
$TXID_2$:1	Alice	0.5 ₿

Figure 2: An example of transactions and the corresponding UTXO model.

represented with $TXID_1$:0 as shown in Figure 2c. The TX_2 transactions are represented with ID's $TXID_2$:0 and $TXID_2$:1.

2.3 Requirements of Transactions

Last, we note three requirements of a UTXO transaction:

1. The sum of all outputs must be less than or equal to the sum of all inputs.
2. The signature must be valid.
3. The input must be part of the UTXO set.

The first requirement ensures that Bitcoins are not created out of thin air. The outputs are allowed to be less than the sum of inputs to provide a **transaction fee** to the miner. The fee incentivizes a miner to include the transaction in their block. Typically, the miner will prioritize transactions based on the bitcoin/byte of block space they receive as a fee. An example of a transaction providing 0.1 ₿ to the miner is shown in Table 4.

in	out
	1.5 ₿ Bob
2 ₿ Alice	
	0.4 ₿ Alice

Table 4: A transaction in which the Bitcoin miner receives a transaction fee of 0.1 ₿.

The only exception to these rules are coin-based transactions, which is when a coin is created as reward for mining a block. In this case, the transaction has no input and the miner receives Bitcoin according to a predetermined schedule.

3 Bitcoin Script

The Bitcoin script is a language to express contracts. The typical objectives of the Bitcoin script are:

- self-custody
- scalability
- trading

A small list of examples of common script primitives include:

- signature verification
- multi-signature (t-of-n) verification: need t of n signatures to spend the coin
- time locks: coin is unspendable for a certain amount of time
- hash locks: coin is locked until a specific piece of data is revealed

3.1 Bitcoin Script Design

- The Bitcoin script is a stack-based language inspired by Forth
- There are no loops, which was done intentionally to ensure programs always terminate
- Bitcoin scripts are stateless
- Bitcoin scripts consist of a locking script (who a coin is sent to) and an unlocking script (to authorize a TX)
- Bitcoin script consists of a simple set of Opcodes.

The Opcodes can be separated into the following categories:

constants	flow control	stack
splice	bitwise logic	arithmetic
crypto	locktime	

A full list of the Opcodes can be found at https://en.bitcoin.it/wiki/Script#Opcodes. Notice, some of the Opcodes are disabled which reduces the power of Bitcoin scripting. The disabling of these Opcodes were done by Nakamoto, who disabled them for security related reasons.

3.2 Time Locks

A time lock can be added to restrict spending on a coin. Time locks can be done based on time stamps or block height. Additionally, time locks can be specified on absolute terms or relative terms (i.e., 10 minutes after TX). Last, time locks may be a transaction time lock or a script time lock.

3.3 Bitcoin Script Examples

Example *An unlocking and locking script with a single signature.*

An example of a Bitcoin program is shown in Figure 3. In this script, a signature is first pushed onto the stack followed by a public key. Last, an operation is performed to check if the signature is valid, returning a 0 or a 1.

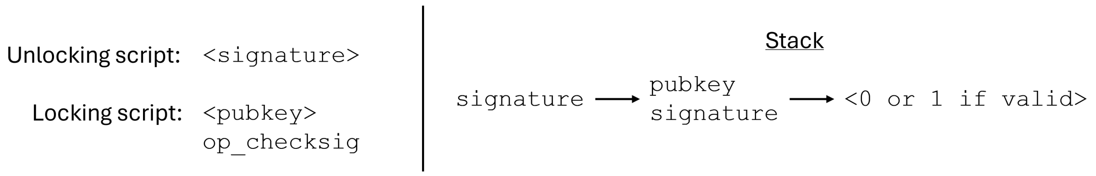

Figure 3: An example of an unlocking and locking script.

Example *An unlocking and locking script with a multi signature.*

The example in Figure 4 shows a script for verifying a multi signature. This script uses a t-of-n clause, where 2-of-3 signatures must be valid for the transaction. In this script, two signatures are first pushed onto the stack followed by three public keys. Last, an operation is performed to check if the signatures meet the 2-of-3 standard.

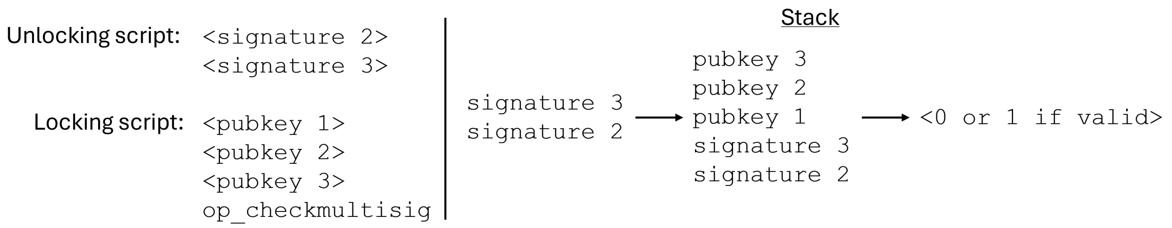

Figure 4: An example of an unlocking and locking script.

To experiment with your own Bitcoin scripts, visit https://ide.scriptwiz.app/.

3.4 A Raw Transaction

Last, we examine what a parsed raw Bitcon transaction looks like. The raw Bitcoin transaction is of the form:

0100000001c997a5e56e104102fa209c6a852dd90660a20b2d9c352423edce25857fcd37040···

Parsing this raw transaction shows that there is a single input and two outputs:

```
version: 01000000
inputsCount: 01
input #1:
        txid: c997a5e56e104102...
        vout: 00000000
        scriptSigSize: 48
        scriptSig:
                OP_PUSHBYTES_71: 47
                data: 304402204e45e16932b...
        sequence: ffffffff
```

```
outputsCount: 02
output #1:
        value: 00ca9a3b00000000
        scriptPubKeySize: 43
        scriptPubKey:
                OP_PUSHBYTES_65:  41
                data: 04ae1a62fe09c5f51b13905f0...
                OP_CHECKSIG: ac
output #2:
        value: 00286bee00000000
        scriptPubKeySize: 43
        scriptPubKey:
                OP_PUSHBYTES_65: 41
                data: 0411db93e1dcdb8a016b49840f...
                OP_CHECKSIG: ac
locktime: 00000000
```

The example above is a Pay to Public Key (P2PK) transaction. There are many other types of transactions, including:

- P2PKH: Pay to Public Key Hash
- P2SH: Pay to Script Hash
- P2WPKH: Pay to Witness Public Key Hash
- P2WSH: Pay to Witness Script Hash
- P2TR: Pay to Tap Root

For more examples, visit this gist by Robin Linus.