Understanding Bitcoin's Scripting Language

1 Objective

The objective of this lab assignment is to gain a deeper understanding of Bitcoin Script and how it operates within the Bitcoin network.

2 Prerequisites

2.1 Introduction to Bitcoin and Bitcoin Script

Bitcoin is a decentralized digital currency that allows peer-to-peer transactions without the need for intermediaries like banks. It operates on a distributed ledger called the blockchain, which records all transactions in a secure and transparent manner. One of the key features of Bitcoin is its scripting language, known as Bitcoin Script, which plays a crucial role in defining the rules for spending bitcoins.

Bitcoin Addresses: Bitcoin addresses are used to receive and send bitcoins. They are derived from public keys, which are in turn derived from private keys. A typical Bitcoin address is a long string of alphanumeric characters, and it serves as a destination for receiving funds.

Bitcoin Transactions: Bitcoin transactions are the fundamental building blocks of the Bitcoin network. Each transaction contains inputs and outputs. Inputs reference unspent transaction outputs (UTXOs) from previous transactions and provide the necessary funds for the new transaction. Outputs specify the destinations and amounts of the transferred funds.

Bitcoin Script: Bitcoin Script is a stack-based, non-Turing-complete programming language used to define the spending conditions for Bitcoin transactions. It consists of a set of opcodes that perform various operations on the data stack. Bitcoin Script is embedded in transaction outputs and is evaluated by nodes on the network to determine if a transaction is valid.

2.2 Tutorial: Understanding Bitcoin Script Basics

In this lab assignment, we will explore Bitcoin Script and its role in creating secure and customizable spending conditions for Bitcoin transactions. Before we delve into the tasks, let's cover some basics of Bitcoin Script.

Bitcoin Script Opcodes Bitcoin Script uses various opcodes to perform operations on the data stack. Here are some essential Bitcoin Script opcodes:

- `OP_DUP`: Duplicates the top item on the stack.
- `OP_HASH160`: Applies a SHA-256 hash followed by RIPEMD-160 hash to the top item.
- `OP_EQUALVERIFY`: Compares the top two items for equality and fails if they are not equal.
- `OP_CHECKSIG`: Checks the signature against the public key and fails if invalid.

- `OP_CHECKMULTISIG`: Checks multiple signatures against multiple public keys.
- `OP_CHECKLOCKTIMEVERIFY`: Checks if the transaction's locktime has been reached.

Creating Bitcoin Script Bitcoin Script is typically used in two main types of transactions: Pay-to-Public-Key-Hash (P2PKH) and Pay-to-Script-Hash (P2SH). P2PKH scripts lock funds to a single public key, while P2SH scripts allow more complex spending conditions using a script hash.

In this tutorial, we'll focus on creating a simple P2PKH script to lock and unlock funds.

P2PKH Script Example Below is an example of a P2PKH script in Python, this script generates a random private key, derives the public key, and calculates the corresponding Bitcoin address:

```
# P2PKH Script
from bitcoin import *
from bitcoin.wallet import CBitcoinSecret, P2PKHBitcoinAddress

# Generate a random private key
private_key = CBitcoinSecret.from_secret_bytes(os.urandom(32))

# Derive the public key and Bitcoin address
public_key = private_key.pub
address = P2PKHBitcoinAddress.from_pubkey(public_key)

print("Private Key:", private_key)
print("Public Key:", public_key.hex())
print("Bitcoin Address:", address)
```

Now that you have a basic understanding of Bitcoin and Bitcoin Script, let's proceed with the lab assignment tasks.

3 Tools and Resources

You will need the following tools and resources for this lab assignment:

- A Bitcoin testnet wallet (for testing purposes).
- Bitcoin Script documentation and examples (provided as reference).
- A code editor and development environment (e.g., Python, JavaScript, or any language of your choice).
- Access to a Bitcoin testnet faucet for receiving testnet BTC.

4 Task 1: Basic Script Execution

4.1 Introduction

In this task, we will start by explaining the fundamental structure of Bitcoin transactions and how scripts are used to lock and unlock funds.

4.2 Task

Your first task is to create a simple Bitcoin script that locks funds to a specific address using the Pay-to-Public-Key-Hash (P2PKH) script. You will be provided with a testnet address.

4.3 Requirements

- **Create a P2PKH Script:** In Python, you can create a P2PKH script as follows:

```
# P2PKH Script
from bitcoin import *
from bitcoin.wallet import CBitcoinSecret, P2PKHBitcoinAddress

# Generate a random private key
private_key = CBitcoinSecret.from_secret_bytes(os.urandom(32))

# Derive the public key and Bitcoin address
public_key = private_key.pub
address = P2PKHBitcoinAddress.from_pubkey(public_key)

print("Private Key:", private_key)
print("Public Key:", public_key.hex())
print("Bitcoin Address:", address)
```

- **Lock Funds:** Use a Bitcoin testnet faucet to send testnet BTC to the generated Bitcoin address.
- **Spend Locked Funds:** Write a Python script to spend the locked funds:

```
# Spend Locked Funds
from bitcoin import *
from bitcoin.wallet import CBitcoinSecret, P2PKHBitcoinAddress

# Private key and Bitcoin address from the previous step
private_key = CBitcoinSecret.from_secret_bytes(...)
address = P2PKHBitcoinAddress.from_pubkey(...)

# Create a transaction input (UTXO)
txid = '...'  # Transaction ID of the UTXO you want to spend
output_index = ...  # Index of the output in the transaction
txin = create_txin(txid, output_index)

# Create a transaction output to the desired destination
destination_address = '...'  # Recipient's address
amount_to_send = ...  # Amount to send in satoshis
txout = create_txout(amount_to_send, destination_address)
```

```
# Create the transaction
tx = create_signed_transaction([txin], [txout], [private_key])

# Broadcast the transaction
broadcast_tx(tx)
```

4.4 Evaluation

Your code will be evaluated to check if it successfully locks and unlocks funds. We will discuss the results and any issues you encountered.

5 Task 2: Multisignature Transactions

5.1 Introduction

In this task, we will introduce the concept of multisignature (multisig) scripts and their use cases in Bitcoin.

5.2 Task

Your second task is to create a 2-of-2 multisig script using provided public keys.

5.3 Requirements

- **Create a 2-of-2 Multisig Script:** Write scripts to create a 2-of-2 multisig address, lock funds to it, and then spend those funds.

```
# 2-of-2 Multisig Script
from bitcoin import *
from bitcoin.wallet import CBitcoinSecret, P2SHBitcoinAddress

# Generate two random private keys
private_key1 = CBitcoinSecret.from_secret_bytes(os.urandom(32))
private_key2 = CBitcoinSecret.from_secret_bytes(os.urandom(32))

# Derive the public keys
public_key1 = private_key1.pub
public_key2 = private_key2.pub

# Create a 2-of-2 multisig redeem script
redeem_script = CScript([OP_2, public_key1, public_key2, OP_2, OP_CHECKMULTISIG])

# Create a P2SH address from the redeem script
address = P2SHBitcoinAddress.from_redeemScript(redeem_script)

print("Private Key 1:", private_key1)
```

```
print("Private Key 2:", private_key2)
print("Redeem Script:", redeem_script.hex())
print("Multisig Address:", address)
```

- **Lock Funds:** Use a Bitcoin testnet faucet to send testnet BTC to the generated multisig address.
- **Spend Locked Funds:** Write a Python script to spend the locked funds from the multisig address:

```
# Spend Funds from Multisig Address
from bitcoin import *
from bitcoin.wallet import CBitcoinSecret, P2SHBitcoinAddress

# Private keys and multisig address from the previous step
private_key1 = CBitcoinSecret.from_secret_bytes(...)
private_key2 = CBitcoinSecret.from_secret_bytes(...)
address = P2SHBitcoinAddress.from_redeemScript(...)

# Create a transaction input (UTXO)
txid = '...'  # Transaction ID of the UTXO you want to spend
output_index = ...  # Index of the output in the transaction
txin = create_txin(txid, output_index)

# Create a transaction output to the desired destination
destination_address = '...'  # Recipient's address
amount_to_send = ...  # Amount to send in satoshis
txout = create_txout(amount_to_send, destination_address)

# Create the transaction
tx = create_signed_transaction([txin], [txout], [private_key1, private_key2], redeem_s

# Broadcast the transaction
broadcast_tx(tx)
```

5.4 Evaluation

Your ability to create and spend funds from a multisig address will be evaluated. We will also discuss the security benefits of multisig addresses.

6 Task 3: Analysis and Reflection

6.1 Introduction

In the final task, analyze the security implications and potential use cases of the scripts you have created.

6.2 Task

Write a report summarizing your experiences and insights gained from this lab assignment. Discuss the advantages and limitations of Bitcoin Script.

6.3 Evaluation

Your reflections and understanding of the practical applications of Bitcoin Script will be evaluated.

7 Submission regulation

- Students create a folder <Student's ID> containing the contents following:
  - <Code> folder: contains the whole project.
  - Executable file (optinal).
  - <Report.pdf> file(at most 5 pages, single page, 12pt font) that includes:
    - Project structure.
    - Data structure.
    - Any other remarks about your design and implementation.
    - Table of complete features with self-grading.
    - All links and books related to your submission must be mentioned.
    - **DO NOT** insert you source code in the report.
- Compress the above folder into Student's ID.zip for submission.
- Submission with wrong regulation will result in a "0" (zero).
- Plagiarism and Cheating will result in a "0" (zero) for the entire course and will be subject to appropriate referral to the Management Board for further action.

8 Grading (may be changed later)

1. Coding: 7 pts.
2. Report: 5 pts.
3. Total: 12 pts.

Good luck with your lab assignment!