Assignment 3: Bitcoin Scripting

CS 216: Introduction to Blockchain

13.02.2025

Assignment Deadline: 12th March 2025 (Thursday) by 11:30 PM.

Submission Instruction

Stick to the team that you formed for the 1st programming assignment. Any change of team members for 2nd programming assignment will lead to marks deduction. Each team should follow this submission rule. The team member who created the a github account for 1st assignment, will create a repository for the 2nd programming assignment and share the link of the same. If you fail to follow this rule then points will be deducted.

1. Create a GitHub repository with a meaningful name

2. Add all relevant source files, including:

- The main program source code/s
- Any additional configuration or dependency files.
- A README.md file with instructions on how to run the program. Mention in this file
- the name and roll numbers of all the team members.
- **Upload a single report for the assignment in PDF format.**

3. Ensure your repository is public.

4. Submit the repository URL in the designated submission portal mentioned in Moodle.

Example Submission Format:

GitHub Repository: https://github.com/yourusername/your_repository_name

Objective:

The objective of this assignment is to understand the process of creating and validating Bitcoin transactions using Legacy (P2PKH) and SegWit (P2SH-P2WPKH) address formats. You will write code in C or Python to interact with `bitcoind`, create transactions, and analyze the scripts involved. You will also compare the transaction sizes and explain the differences.

For your help all the details are mentioned in this tutorial:
https://github.com/BlockchainCommons/Learning-Bitcoin-from-the-Command-Line/blob/master/18_0_Talking_to_Bitcoind_Other.md

N.B. : Regtest uses IP: 127.0.0.1 and PORT: 18443, for connecting to Bitcoin Daemon, you need the rpc user and rpc password that you had written in your bitcoin.conf file.

Tools Required:

1. **Bitcoin Core (`bitcoind`)**: A full node implementation of Bitcoin.
2. **Python** or **C**: For scripting and interacting with `bitcoind`.
3. **Bitcoin CLI (`bitcoin-cli`)**: Command-line tool to interact with `bitcoind`.
4. **Bitcoin Debugger**: For validating and decoding scripts.
5. **Library Dependencies**:
    - For Python: `python-bitcoinlib` or `bitcoinrpc`.
    - For C: `libbitcoin` or custom RPC calls using `curl` or `libcurl`.
    - For others check https://github.com/BlockchainCommons/Learning-Bitcoin-from-the-Command-Line/blob/master/18_0_Talking_to_Bitcoind_Other.md

Assignment Steps:

Part 1: Legacy Address Transactions

1. **Setup Environment**:
    - Install and configure `bitcoind` on your system.
    - Before executing bitcoind in regtest mode, make the following changes in **bitcoin.conf** file:

In Bitcoin, the fee policy and transaction confirmation target are important settings that determine how your node prioritizes transactions and calculates fees. These settings can be configured in your `bitcoin.conf` file or passed as command-line arguments when running `bitcoind` or `bitcoin-cli`. Here's an explanation of what they are and why they are needed:

Key Fee-Related Settings in `bitcoin.conf`:

- **`paytxfee`**:

  - This sets a default fee rate (in BTC/kB) for transactions created by your node.

  - Example: `paytxfee=0.0001` sets a default fee rate of 0.0001 BTC per kilobyte.

- **`fallbackfee`**:

  - This sets a fallback fee rate (in BTC/kB) to use when the fee estimation mechanism fails.

  - Example: `fallbackfee=0.0002` sets a fallback fee rate of 0.0002 BTC per kilobyte.

- **`mintxfee`**:

  - This sets the minimum fee rate (in BTC/kB) for transactions created by your node. Transactions with fees below this rate will not be accepted.

  - Example: `mintxfee=0.00001` sets a minimum fee rate of 0.00001 BTC per kilobyte.

2. Transaction Confirmation Target

The **transaction confirmation target** specifies how quickly you want your transactions to be confirmed. This target influences the fee estimation algorithm to calculate an appropriate fee rate.

Key Confirmation Target Settings in `bitcoin.conf`:

- **`txconfirmtarget`**:

  - This sets the default number of blocks in which you want your transactions to be confirmed.

  - Example: `txconfirmtarget=6` means you want your transactions to be confirmed within 6 blocks.

Example `bitcoin.conf` Configuration:

Here's an example of how you might configure these settings in your `bitcoin.conf` file:

paytxfee=0.0001 # Default fee rate of 0.0001 BTC/kB

fallbackfee=0.0002 # Fallback fee rate of 0.0002 BTC/kB

mintxfee=0.00001 # Minimum fee rate of 0.00001 BTC/kB

txconfirmtarget=6 # Aim for confirmation within 6 blocks

(Note that you can reduce the number from 6 to 1 for faster confirmation)

3. Once you have set all the parameters in configuration file, **start bitcoind in regtest mode**

Write a Python or C or (any other language) program to:

- Connect to `bitcoind` using RPC.
- Create a new wallet or load an existing wallet
- Generate three legacy addresses: `A`, `B`, and `C`.
- Fund address `A` by sendtoaddress A. The `sendtoaddress` command is used to send Bitcoin from your wallet to a specified Bitcoin address. It is one of the most commonly used commands for creating and broadcasting transactions. Check the command for other options as well.
- **Create a Transaction from Address A to Address B**:
  - Create a raw transaction sending coins from `A` to `B`.

Decode and Analyze the Transaction:

- Decode the raw transactions to extract the locking script (ScriptPubKey) for Address B.

Broadcasting Transaction

- Sign the transaction using the private key of `A (or signwithwallet)`.
- Broadcast the transaction to the Bitcoin network.

Write another Python or C or (any other language) program to:

- Get the listunspent enlisting the **txid** that shows address B as UTXO due to previous transaction from A to B
- Create a Transaction from Address B to Address C funded by this **txid**
- Repeat the rest of the process to send coins from `B` to `C` as we did for the last transaction.
- Decode the rawtransaction and analyze the response part of the transaction (ScriptSig) and check if this response matches with the challenge script provided by the transaction from **address A to address B**. Analyze the scripts to understand the locking and unlocking mechanisms. Validate the scripts using the Bitcoin Debugger to ensure correctness.

Report Content for legacy (P2PKH):

- Mention the workflow that states what was the txid for transaction from A to B, how this transaction was used as input for the next transaction from B to C
- Provide the decoded scripts for both transactions.
- Explain the structure of the challenge and response scripts and how they validate the transaction
- Share screenshots of decoded scripts, and steps of Bitcoin debugger executing challenge and response script.

Part 2: P2SH-SegWit Address Transactions

Write a Python or C or (any other language) program to:

1. Connect to `bitcoind` using RPC.
2. Create a new wallet or load an existing wallet
3. **Generate P2SH-SegWit Addresses**:
    - Generate three P2SH-SegWit addresses: `A'`, `B'`, and `C'`.
    - Fund address `A` by sendtoaddress A'. The `sendtoaddress` command is used to send Bitcoin from your wallet to a specified Bitcoin address. It is one of the most commonly used commands for creating and broadcasting transactions. Check the command for other options as well.
4. **Create a Transaction from Address A' to Address B'**:
    - Create a raw transaction sending coins from `A'` to `B'`. Decode the raw transaction and extract the challenge script for B'.
    - Sign the transaction using the private key of `A' (or signwithwallet)`.
    - Broadcast the transaction.
5. **Create a Transaction from Address B' to Address C'**:
    - Repeat the process to send coins from `B'` to `C'`.
    - Decode and analyze the transaction scripts. Analyze the scripts to understand the locking and unlocking mechanisms. Validate the scripts using the Bitcoin Debugger to ensure correctness.

Report Content for Segwit:

- Mention the workflow that states what was the txid for transaction from A to B, how this transaction was used as input for the next transaction from B to C
- Provide the decoded scripts for both transactions.
- Explain the structure of the challenge and response scripts and how they validate the transaction
- Share screenshots of decoded scripts, and steps of Bitcoin debugger executing challenge and response script.

Part 3: Analysis and Explanation

1. **In the report do a Comparison of the Part 1 and Part 2**:
    - Check the size of the P2PKH transactions (Part 1) and the P2SH-P2WPKH transactions (Part 2).
    - Compare the script structures of P2PKH (legacy) and P2SH-P2WPKH (Segwit) transactions in terms of challenge-response script and size of the script (weight, vbyte)
    - Explain why the SegWit transactions are smaller, and what is the benefit of such transactions.

2. **Submit Your Code and Report**:
    - Upload your code and report in the github
    - In the report, provide screenshots of decoded scripts, and execution of Bitcoin debugger and analysis.
