README

Review the assignment due date

Bitcoin Protocol Development - Week 2: Building a P2SH-P2WSH Multisig Transaction

Overview

In this exercise you will learn about building a P2SH wrapped P2WSH transaction that spends a 2-of-2 multisig redeemscript. This is one of the most rudimentary yet involved transaction building process where you will learn how to calculate a sighash, create ecdsa signatures, create the right `script_sig` and the `witness` to pass validation of the transaction. Even though P2SH-P2WSH transactions are less optimal than direct P2WSH transactions, this exercise will show you how to correctly compute all the different parts of a transaction.

You can use any popular Bitcoin library in your language of choice to craft the transaction.

Objective

Successfully create a signed P2SH-P2WSH multisig transaction.

Your tasks are to:

- Craft a transaction with the requirements.
- Calculate the signhash of the unsigned transaction and compute ecdsa signatures with both private keys.
- Correctly create the `script_sig` of the transaction from the given `redeem_script`.
- Correctly create the `witness` stack of the transaction for a valid multisig witness.
- Write the transaction hex to `out.txt`.

Place your solution in the appropriate directory based on your chosen language:

- bash

- javascript
- python
- rust

Requirements

Transaction Spec

- Private Key 1: `39dc0a9f0b185a2ee56349691f34716e6e0cda06a7f9707742ac113c4e2317bf`
- Private Key 2: `5077ccd9c558b7d04a81920d38aa11b4a9f9de3b23fab45c3ef28039920fdd6d`
- Redeem Script (ASM): `OP_2 032ff8c5df0bc00fe1ac2319c3b8070d6d1e04cfbf4fedda499ae7b775185ad53b 039bbc8d24f89e5bc44c5b0d1980d6658316a6b2440023117c3c03a4975b04dd56 OP_2 OP_CHECKMULTISIG`
- Redeem Script (HEX): `5221032ff8c5df0bc00fe1ac2319c3b8070d6d1e04cfbf4fedda499ae7b775185ad53b21039bbc8d24f89e5bc44c5b0d1980d6658316a6b2440023117c3c03a4975b04dd5652ae`

- Transaction should contain exactly 1 input with:
  - Outpoint:
    - Hash: `0000000000000000000000000000000000000000000000000000000000000000`
    - Index: `0`
  - Sequence: `0xffffffff`
- Transaction should contain exactly 1 output with:
  - Value: `0.001`
  - Address: `325UUecEQuyrTd28Xs2hvAxdAjHM7XzqVF`
- Locktime: `0`

Output

After creating and the transaction, save the serialized transaction hex to out.txt.

Execution

To test your solution locally:

- Uncomment the line corresponding to your language in run.sh.
- Execute `test.sh` .

If your code works, you will see the test completed successfully.

Evaluation Criteria

Your submission will be evaluated based on:

- **Autograder**: Your code must pass the autograder test script.
- **Explainer Comments**: Include comments explaining each step of your code.
- **Code Quality**: Your code should be well-organized, commented, and adhere to best practices.

Plagiarism Policy

Our plagiarism detection checker thoroughly identifies any instances of copying or cheating. Participants are required to publish their solutions in the designated repository, which is private and accessible only to the individual and the administrator. Solutions should not be shared publicly or with peers. In case of plagiarism, both parties involved will be directly disqualified to maintain fairness and integrity.

AI Usage Disclaimer

You may use AI tools like ChatGPT to gather information and explore alternative approaches, but avoid relying solely on AI for complete solutions. Verify and validate any insights obtained and maintain a balance between AI assistance and independent problem-solving.

Why These Restrictions?

These rules are designed to enhance your understanding of the technical aspects of Bitcoin. By completing this assignment, you gain practical experience with the technology that secures and maintains the trustlessness of Bitcoin. This challenge not only tests your ability to develop functional Bitcoin applications but also encourages deep engagement with the core elements of Bitcoin technology.