CS 216: Bitcoin Scripting Assignment Report

Part 1: Legacy Address Transactions

1. Transaction Workflow:

The script executes a sequence of Bitcoin transactions on a local Bitcoin Core node using the `AuthServiceProxy` RPC interface. The primary workflow follows:

1. **Wallet and Address Setup:**
    - Load a wallet (`mywallet`)
    - Generate three legacy addresses: `Legacy A`, `Legacy B` and `Legacy C`.
2. **Mining and Funding:**
    - Mine 201 blocks to `Legacy A`, ensuring sufficient balance.
    - Send 10 BTC from `Legacy A` to itself, generating an unspent transaction output (UTXO) for further use.
3. **Transaction 1: Sending BTC from A to B:**
    - Fetch the UTXO created from the funding transaction.
    - Construct a raw transaction where:
        - 5 BTC is sent to `Legacy B`.
        - Change (original UTXO minus sent amount and fee) is returned to `Legacy A`.
    - Sign and broadcast the transaction.
4. **Transaction 2: Sending BTC from B to C**
    - Fetch the newly created UTXO from the A to B transaction.
    - Construct a raw transaction where:
        - 2.5 BTC is sent to `Legacy C`.
        - Remaining balance (minus fee) is returned to `Legacy B`.
    - Sign and broadcast the transaction.

The transaction sending BTC

2. Decoded Scripts for Both Transactions

1. **Transaction 1 (A → B):**


```
Decoded signed transaction {'txid': '07055bf40b43a6f859c4de9c4af7d82bb9b0c6a69173ec44b01556dfcf5411f8', 'hash': '07055bf40b43a6f859c4de9c4af7d82bb9b0c6a69173ec44b01556dfcf5411f8', 'version': 2, 'size': 225, 'vsize': 225, 'weight': 900, 'locktime': 0, 'vin': [{'txid': '5e53be790ad26e97f518a74b02e98520617c0b8b5b0fd00e483fa32edad10489', 'vout': 0, 'scriptSig': {'asm': '3044022052d5fcef3ea8221eec24d69e63ff941618fe7cc06a1bfa1d7765ca42245d47f00220119ca14a164b101f58fdd480a4e844199e55c12a35c2f3c2cc02e19a1943e274[ALL] 0388623bbefb96515960891573c2ca44b3a1410c352651225f9242525dd5f85e31', 'hex': '473044022052d5fcef3ea8221eec24d69e63ff941618fe7cc06a1bfa1d7765ca42245d47f00220119ca14a164b101f58fdd480a4e844199e55c12a35c2f3c2cc02e19a1943e27401210388623bbefb96515960891573c2ca44b3a1410c352651225f9242525dd5f85e31'}, 'sequence': 4294967293}], 'vout': [{'value': Decimal('5.00000000'), 'n': 0, 'scriptPubKey': {'asm': 'OP_DUP OP_HASH160 a1700221e2435191a355d390d6064377326f3753 OP_EQUALVERIFY OP_CHECKSIG', 'desc': 'addr(mvEZGQxaaDEr8Eb5W5Ryu9F44PBkcAdvRD)#z82zwgdf', 'hex': '76a914a1700221e2435191a355d390d6064377326f375388ac', 'address': 'mvEZGQxaaDEr8Eb5W5Ryu9F44PBkcAdvRD', 'type': 'pubkeyhash'}}, {'value': Decimal('4.99990000'), 'n': 1, 'scriptPubKey': {'asm': 'OP_DUP OP_HASH160 44f71eee441438c6f4efc4ef50fe1e55d2f77b4c OP_EQUALVERIFY OP_CHECKSIG', 'desc': 'addr(mmocHMW7DdxpgUm5UKgEroQLKEkZJDxuah)#9l8f84h9', 'hex': '76a91444f71eee441438c6f4efc4ef50fe1e55d2f77b4c88ac', 'address': 'mmocHMW7DdxpgUm5UKgEroQLKEkZJDxuah', 'type': 'pubkeyhash'}}]}
```


2. **Transaction 2 (B → C):**

3. Challenge and Response Script Analysis:

Each transaction uses P2PKH scripts, which involve a challenge (locking script) and a response (unlocking script).

1. **Locking Script (Challenge):** Ensures only the recipient can spend the output.
2. **Unlocking Script (Response):** Provides a valid signature and public key.
3. **Validation Process:**
    - Bitcoin nodes execute the unlocking script first, pushing the provided signature and public key onto the stack.
    - Then, the locking script is executed to verify ownership by checking that the provided public key hash matches and that the signature is valid.

4. Validation Using Bitcoin Debugger:

Part 2: P2SH-SegWit Address Transactions

1. Transaction Workflow and Execution

1. Initial Funding of Address A:

- A Bitcoin address (Address A) is generated.
- 201 blocks are mined to this address, ensuring the funds mature and become spendable.
- The wallet balance is retrieved.

- A transaction of 10 BTC is sent to Address A, generating a unique transaction ID.
- The unspent transaction output (UTXO) associated with Address A is then identified.

2. **Transaction from Address A to Address B:**
    - A UTXO from `txid_fund` is selected as input for the new transaction.
    - A transfer of 5 BTC is initiated from Address A to Address B, with a small transaction fee deducted.
    - A raw transaction is then formulated using `createrawtransaction()`.
    - The raw transaction is then signed using `signrawtransactionwithwallet()`.
    - The signed transaction is broadcasted, generating a new transaction ID.
    - The decoded transaction confirms that Address B successfully received 5 BTC.
3. **Transaction from Address B to Address C:**
    - The UTXO from `txid` is used as an input for another transaction.
    - 2.5 BTC is sent from Address B to Address C.
    - A new raw transaction is generated, signed, and broadcasted following the same process.
    - The decoded transaction confirms that Address C received 2.5 BTC.

2. Decoded Transaction Scripts:

1. **Transaction 1 (A → B):**

2. **Transaction 2 (B → C):**

3. Analysis of Challenge and Response Scripts:

Bitcoin Script Execution

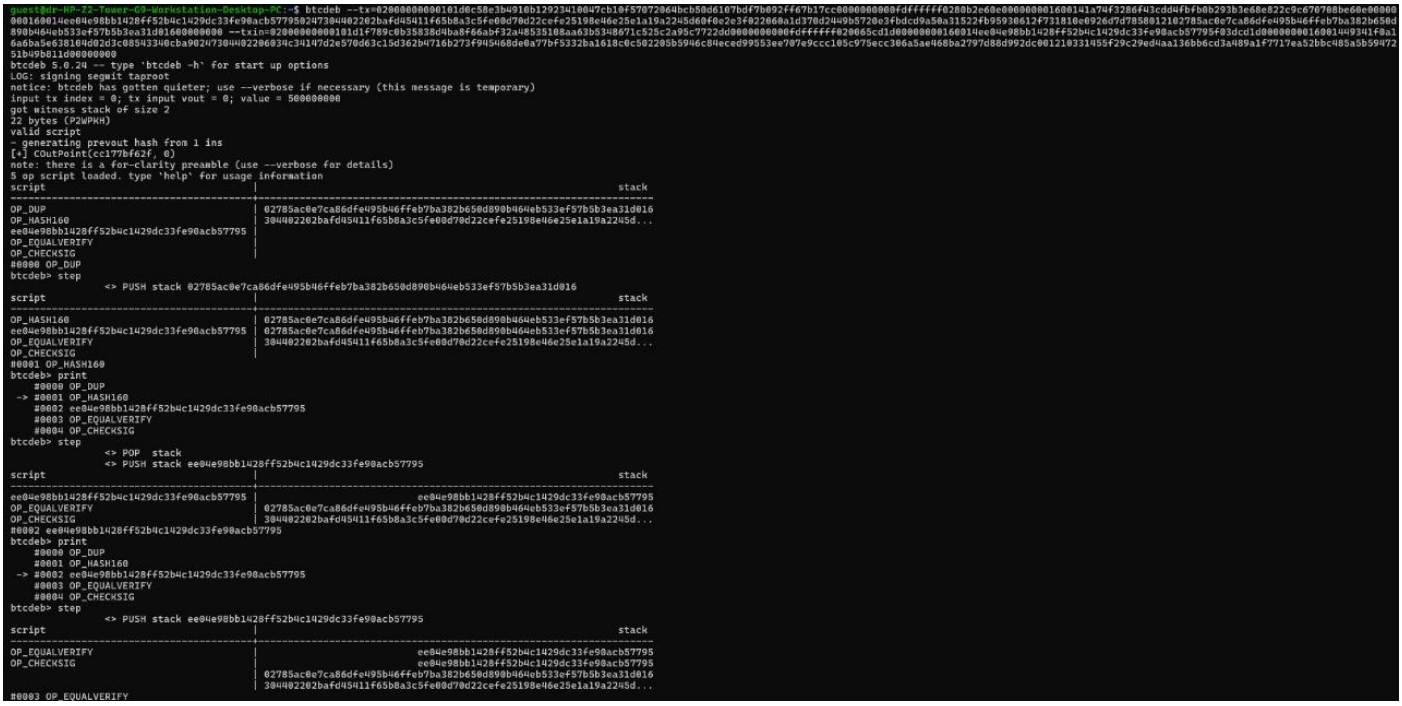
Bitcoin transactions use a challenge-response mechanism through scripting.

1. **Locking Script (scriptPubKey):** This script defines the conditions required to unlock the UTXO. A typical Pay-to-Public-Key-Hash (P2PKH) script follows the structure:
    - OP_DUP: Duplicates the public key on the stack.
    - OP_HASH160: Computes the hash of the public key.
    - OP_EQUALVERIFY: Verifies that the provided public key hash matches the expected hash.
    - OP_CHECKSIG: Validates the signature against the public key.
2. **Unlocking Script (scriptSig):** This script provides the necessary proof to meet the locking script conditions:
    - The digital signature is generated using the sender's private key and proves ownership of the UTXO.

Transaction Validation Process:

3. The unlocking script (scriptSig) executes first, placing the signature and public key onto the stack.
4. The locking script (scriptPubKey) runs, verifying the public key's hash and the authenticity of the digital signature.
5. If all conditions hold, the transaction is deemed valid and accepted into the blockchain.

4. Validation using Bitcoin Debugger:



Step 3: Comparison of Part 1 and Part 2:

1. Address Format:

1. **Legacy:** Uses P2PKH (Pay-to-PubKey-Hash) addresses, starting with `m` (for regtest). Eg: mmocHMW7DdxpgUm5UKgEroQLKEkZJDxuah
2. **SegWit:** Uses Bech32 (P2WPKH) addresses, starting with `bcrt1`. Eg: bcrt1qfy6p7zsk56a9ucupqngz60qg2se5pjafmkf5zg

2. Transaction Size & Weight:

1. **Legacy:**
    i. **Size:** Larger in size (225 bytes - vsize). This is because it includes full signatures inside `scriptSig`.
    ii. **Weight:** Higher weight (900). This is because of inefficient signature handling.
2. **SegWit:**
    i. **Size:** Smaller in size (141 bytes – vsize). This is because signatures are moved to `witness` data.
    ii. **Weight:** Reduced weight (561), allowing more transactions per block.

3. Transaction Structure:

1. **Legacy:**
    i. Uses `scriptSig` in the input for unlocking funds, including the full signature and public key.
2. **SegWit:**
    i. Uses `witness` data instead of `scriptSig`, making transactions more compact.

SegWit transactions are smaller primarily because they move the signature data (witness data) outside the main transaction structure. Here's how it works:

1. **Signature Data is Moved to the Witness Field:**
    - **Legacy transactions:** Signatures are included in the scriptSig field of each input. This makes the transaction larger because every input must include a full signature.
    - **SegWit transactions**: Signatures are moved to the witness section, which is not included in the transaction ID calculation. This reduces the size of the main transaction.
2. **Witness Data is Discounted:**
    - Bitcoin blocks are limited by weight units instead of just size in bytes.
    - Witness data is discounted by a factor of 4, meaning 1 byte of witness data counts as only 1/4th of a regular byte.
    - As a result, SegWit transactions are smaller in terms of vsize (virtual size) when compared to Legacy transactions.

Benefits of SegWit Transactions:

1. **Lower Transaction Fees:** Since SegWit transactions have a smaller virtual size, they pay lower fees compared to legacy transactions.
2. **More Transactions per Block:**
    a. Bitcoin blocks have a maximum weight limit of 4 million weight units instead of a strict byte limit.
    b. Since witness data is discounted, SegWit transactions take up less weight, meaning more transactions can fit in a single block.
    c. This increases Bitcoin's transaction throughput without increasing the block size.
3. **Fixes Transaction Malleability:**
    a. In Legacy transactions, changing the scriptSig (even in a trivial way) changes the transaction ID (TXID).
    b. In SegWit, the signature data (witness) is excluded from the TXID calculation, meaning TXIDs remain stable even if the witness data is modified.
    c. This fix enables second-layer solutions like the Lightning Network, which relies on predictable TXIDs.