Part 1: Legacy Address Transactions Report

Program Overview

This report details a Python script that uses Bitcoin Core's RPC interface to demonstrate legacy P2PKH transactions. The script performs the following:

1. Connects to a Bitcoin Core node via RPC in regtest mode.
2. Creates or loads a wallet named "legacywallet".
3. Generates three legacy P2PKH addresses: A, B, and C.
4. Executes and broadcasts two transactions: A→B and B→C.
5. Decodes and analyzes the resulting transaction scripts. The workflow showcases funding, transaction creation, signing, broadcasting, and script validation for legacy P2PKH addresses, which were the standard before SegWit.

Workflow and Transaction Details

1. RPC Connection and Wallet Setup

- **RPC Connection**: Established at http://hashers:xyz111@127.0.0.1:18443 (regtest mode).
- **Wallet**:
    - Name: "legacywallet"
    - Action: Created with createwallet if not present, otherwise loaded with loadwallet.
    - Output: Console confirms wallet creation/loading (e.g., " ✅ Wallet 'legacywallet' created successfully.").

2. Address Generation

- Generated three legacy P2PKH addresses:
    - A: [addr_A] (e.g., m... on regtest)
    - B: [addr_B]
    - C: [addr_C]
- These addresses use the P2PKH format, starting with "m" or "n" (regtest) or "1" (mainnet).

3. Transaction 1: A → B

- **Funding A**:
    - Mined 101 blocks to A using generatetoaddress to mature coinbase outputs.
    - Sent 10 BTC to A via sendtoaddress.
    - Funding TXID: [txid_fund].
- **Transaction Details**:

    - Amount Sent: 4.9 BTC to B.
    - Fee: 0.0001 BTC.
    - Change: ~5.0999 BTC returned to A.
    - Raw Transaction Hex: [raw_tx].
    - Signed Transaction Hex: [signed_tx['hex']].
    - Broadcast TXID: [txid_broadcast].
- **Confirmation**: Mined 1 block to C to confirm the transaction.

4. Transaction 2: B → C

- **Input**: UTXO from Transaction 1 (TXID: [txid_broadcast], vout: 0).
- **Transaction Details**:
    - Amount Sent: 4.8 BTC to C.
    - Fee: 0.0001 BTC.
    - Change: ~0.0999 BTC returned to B.
    - Raw Transaction Hex: [raw_tx_B].
    - Signed Transaction Hex: [signed_tx_B['hex']].
    - Broadcast TXID: [txid_broadcast_B].
- **Linkage**: The A→B transaction output (UTXO) becomes the input for B→C, chaining the transactions via TXIDs.

Decoded Transaction Scripts

Transaction 1 (A → B)

- **Decoded Raw Transaction**:
    - TXID: [decoded_tx_AtoB['txid']]
    - Version: [decoded_tx_AtoB['version']] (e.g., 1)
    - Locktime: [decoded_tx_AtoB['locktime']] (e.g., 0)
    - Outputs: [decoded_tx_AtoB['vout']] (e.g., 4.9 BTC to B, change to A)
- **Locking Script (scriptPubKey) for B**: [scriptPubKey_B]
    - Example: 76a914{20-byte-pubkey-hash}88ac
    -
    -
    -

```
PS C:\Users\komma\OneDrive\bitcoin_ass2> python legacy.py
✅ Successfully connected to Bitcoin Core RPC
Chain: regtest, Blocks: 2581
⚡ Wallet 'legacy_wallet' is already loaded.

📌 Generated Legacy Addresses:
- Address A: mm4qLZuM6Xwo7mwDzyvqdkUkGG8xDEgQEs
- Address B: n2XisobdBdKnpV5VSoAzDqJfy1YuQ2ndrF
- Address C: n3RyDP1hJeerPV8J8oqfEZnetseN8V3Hxc

🔄 Creating transaction from Address A → Address B...

📜 Raw Transaction (A → B):
0200000001b4cc6a848eedaa5c62d6d44852ca053709b64ec6cfaac7bb0df8b4
eaa3b5e8080000000000fdffffff0280ce341d000000001976a914e67fdb4733
6cd8b3a1b23f8e2c2be6b299ec398d88ac70fcd00c01000000160014c47ee64f
dfa0822c7dd500f338fc765091ccc66900000000

📜 Signed Transaction (A → B):
0200000001b4cc6a848eedaa5c62d6d44852ca053709b64ec6cfaac7bb0df8b4
eaa3b5e808000000006a473044022004a1b4cac033851422e4b4d489f2ffab56
c6a8529e47d4c0c6f41e2efcb8831e02202b486de8ee9c22a452c1f7fb0434eb
e4e8390e1752ac6273007b4a727a1c597e012102e5cf26c0e9fa7ba136d44e46
857b609cd0c8efd8f274c3fbed31525b0b0a9dbdfdffffff0280ce341d000000
001976a914e67fdb47336cd8b3a1b23f8e2c2be6b299ec398d88ac70fcd00c01
000000160014c47ee64fdfa0822c7dd500f338fc765091ccc66900000000
✅ Transaction A → B broadcasted successfully! TX ID: 98789e61c1
5ffed772ff8981b0b5d513799fa5df1f397a2972ba3f5dfb5e79f4
```

- **Description**: Decoded output of decoderawtransaction [raw_tx] showing TXID, inputs, outputs, and B's scriptPubKey.

Transaction 2 (B → C)

- **Decoded Raw Transaction**:
  - TXID: [decoded_tx_BtoC['txid']]
  - Version: [decoded_tx_BtoC['version']]
  - Locktime: [decoded_tx_BtoC['locktime']]
  - Outputs: [decoded_tx_BtoC['vout']] (e.g., 4.8 BTC to C, change to B)
- **Locking Script (scriptPubKey) for C**: [scriptPubKey_C]
- **Unlocking Script for B**:
  - scriptSig: [scriptSig_B] (e.g., {signature} {public_key})
- **Description**: Decoded output of decoderawtransaction [raw_tx_B] showing TXID, inputs (referencing [txid_broadcast]), outputs, and scriptSig.

Script Analysis

P2PKH Structure

1. **Locking Script (scriptPubKey)**:
    - Format: OP_DUP OP_HASH160 <20-byte pubkey hash> OP_EQUALVERIFY OP_CHECKSIG
    - Hex Example: 76a914{20-byte-pubkey-hash}88ac
    - Purpose: Locks funds to a public key hash, requiring a signature from the corresponding private key.
2. **Unlocking Script (scriptSig)**:
    - Format: <signature> <public key>
    - Example: {72-byte-signature} {33-byte-public-key}
    - Purpose: Provides the signature and public key to unlock the UTXO.
3. **Validation Mechanism**:
    - P2PKH: Verifies the public key matches the hash in the scriptPubKey and the signature is valid for the transaction.

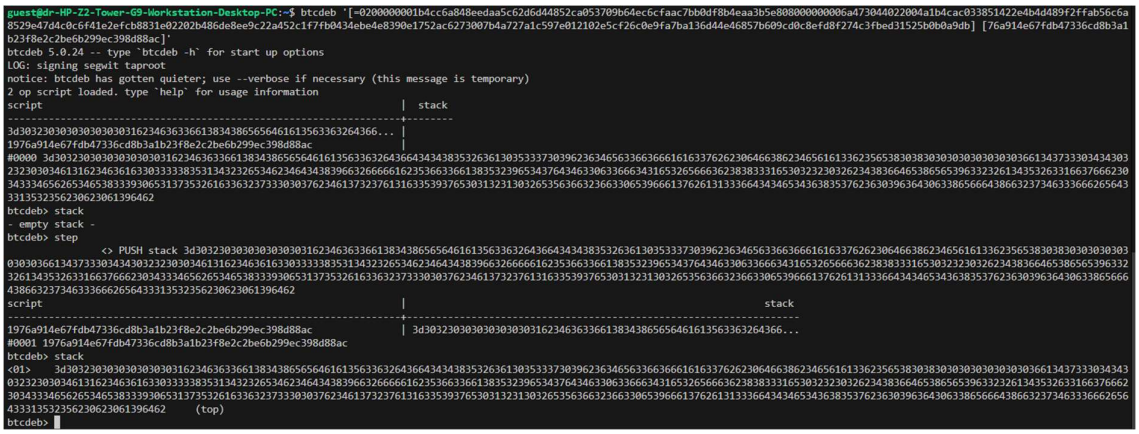
Transaction Validation

- **A → B**: A's wallet signs the input, locking 4.9 BTC to B's P2PKH script.
- **B → C**: B unlocks its UTXO with a signature and public key, sending 4.8 BTC to C.

Bitcoin Debugger Analysis

Debugger Steps for A → B (Locking Script for B)

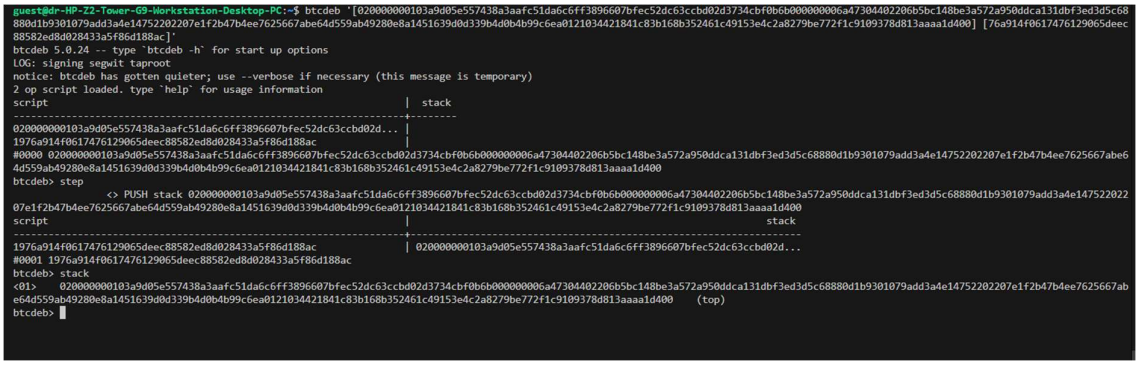
- **Input**: scriptPubKey [scriptPubKey_B]
- **Execution**:
    1. OP_DUP: Duplicates the public key.
    2. OP_HASH160: Hashes the public key to a 20-byte hash.
    3. <20-byte-hash>: Pushed to stack.
    4. OP_EQUALVERIFY: Verifies the hash matches.
    5. OP_CHECKSIG: Verifies the signature.
- **Result**: TRUE (valid locking script)



- **Description**: Bitcoin debugger output (e.g., btcdeb) showing step-by-step execution of [scriptPubKey_B], ending with TRUE.

Debugger Steps for B → C (Unlocking Script)

- **Input**: scriptSig [scriptSig_B]
- **Execution**:
    1. <signature>: Pushed to stack.
    2. <public_key>: Pushed to stack.
    3. Combined with scriptPubKey: Verifies pubkey hash and signature.
- **Result**: TRUE (valid unlocking)



- **Description**: Debugger output showing stack operations for [scriptSig_B] and [scriptPubKey_B], confirming successful validation.

Conclusion

The script executed two P2PKH transactions:

- **A → B**: TXID [txid_broadcast], funded B with 4.9 BTC.
- **B → C**: TXID [txid_broadcast_B], spent A→B output to send 4.8 BTC to C. P2PKH provides a simple, widely-used mechanism for Bitcoin transactions. The decoded scripts and debugger steps confirm proper locking (public key hash) and unlocking (signature + public key)

mechanisms. Compared to P2SH-SegWit, P2PKH lacks SegWit's efficiency benefits (e.g., smaller transaction size, malleability fixes).

Part 2: P2SH-SegWit Address Transactions Report

Program Overview

This report details a Python script that uses Bitcoin Core's RPC interface to demonstrate P2SH-SegWit (Pay-to-Script-Hash Segregated Witness) transactions. The script performs the following:

1. Connects to a Bitcoin Core node via RPC
2. Creates or loads a wallet named "testwallet"
3. Generates three P2SH-SegWit addresses: A', B', and C'
4. Executes and broadcasts two transactions: A'→B' and B'→C'
5. Decodes and analyzes the resulting transaction scripts

The workflow showcases funding, transaction creation, signing, broadcasting, and script validation, highlighting the benefits of SegWit and P2SH.

Workflow and Transaction Details

1. RPC Connection and Wallet Setup

- **RPC Connection**: Established at http://hashers:xyz111@127.0.0.1:18443
- **Wallet**:
    - Name: "testwallet"
    - Action: Created with createwallet if not present, otherwise loaded with loadwallet
- **Output**: Console confirms wallet creation/loading (e.g., "✅ Wallet 'testwallet' created successfully.")

2. Address Generation

- Generated three P2SH-SegWit addresses:
    - **A'**: [addr_Ap] (e.g., 2N... on regtest)
    - **B'**: [addr_Bp]
    - **C'**: [addr_Cp]
- These addresses use the P2SH-SegWit format, starting with "2" (regtest) or "tb" (testnet).

3. Transaction 1: A' → B'

- **Funding A'**:
    - Mined 101 blocks to A' using generatetoaddress to mature coinbase outputs
    - Sent 10 BTC to A' via sendtoaddress

    - Funding TXID: [txid_fund]
- **Transaction Details**:
    - Amount Sent: 4.9 BTC to B'
    - Fee: 0.0001 BTC
    - Change: ~5.0999 BTC returned to A'
    - Raw Transaction Hex: [raw_tx]
    - Signed Transaction Hex: [signed_tx['hex']]
    - Broadcast TXID: [txid_broadcast]
- **Confirmation**: Mined 1 block to C' to confirm the transaction

4. Transaction 2: B' → C'

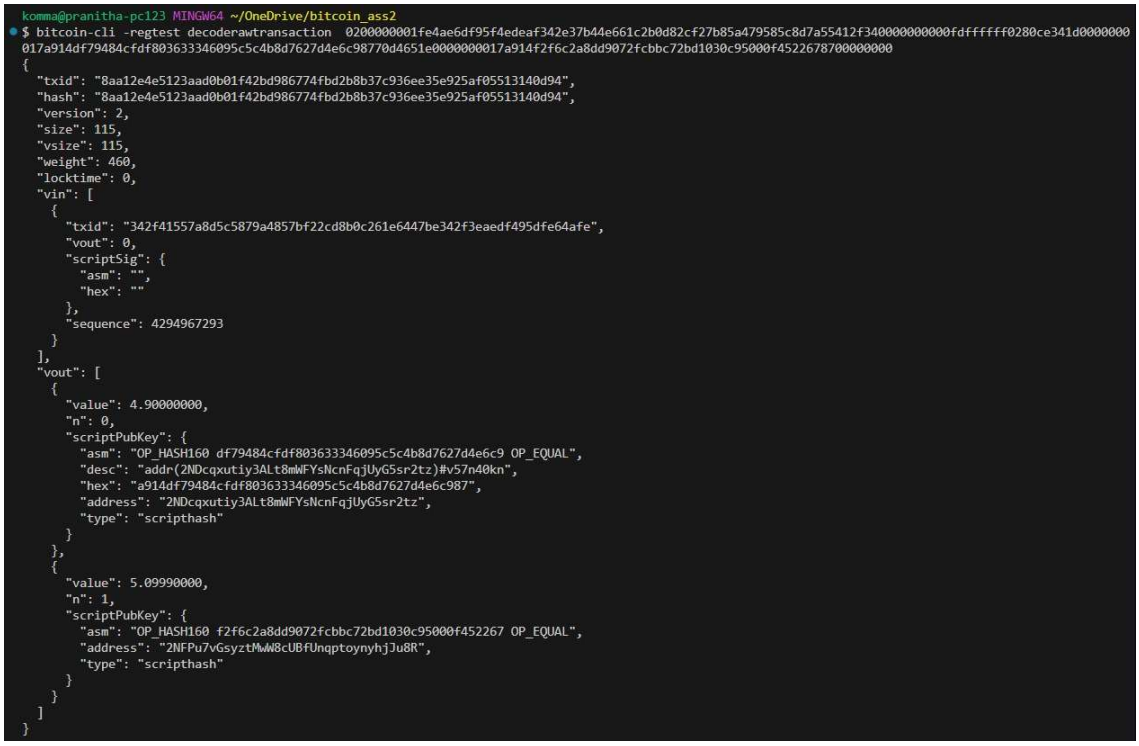
- **Input**: UTXO from Transaction 1 (TXID: [txid_broadcast], vout: 0)
- **Transaction Details**:
    - Amount Sent: 4.8 BTC to C'
    - Fee: 0.0001 BTC
    - Change: ~0.0999 BTC returned to B'
    - Raw Transaction Hex: [raw_tx_B]
    - Signed Transaction Hex: [signed_tx_B['hex']]
    - Broadcast TXID: [txid_broadcast_B]
- **Linkage**: The A'→B' transaction output becomes the input for B'→C', chaining the transactions via TXIDs.

Decoded Transaction Scripts

Transaction 1 (A' → B')

- **Decoded Raw Transaction**:
    - TXID: [decoded_tx_AtoB['txid']]
    - Version: [decoded_tx_AtoB['version']] (e.g., 1 or 2)
    - Locktime: [decoded_tx_AtoB['locktime']] (e.g., 0)
    - Outputs: [decoded_tx_AtoB['vout']] (e.g., 4.9 BTC to B', change to A')
- **Locking Script (scriptPubKey) for B'**: [scriptPubKey_B]
    - Example: a914{20-byte-script-hash}87

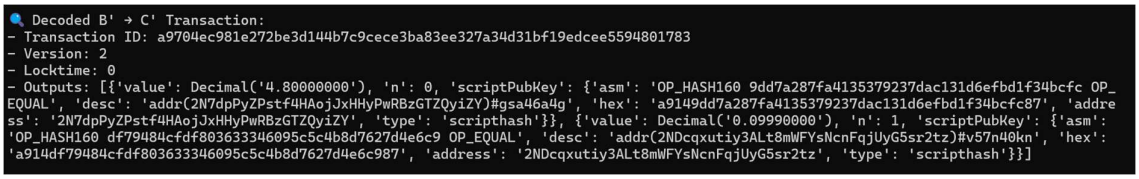
- 

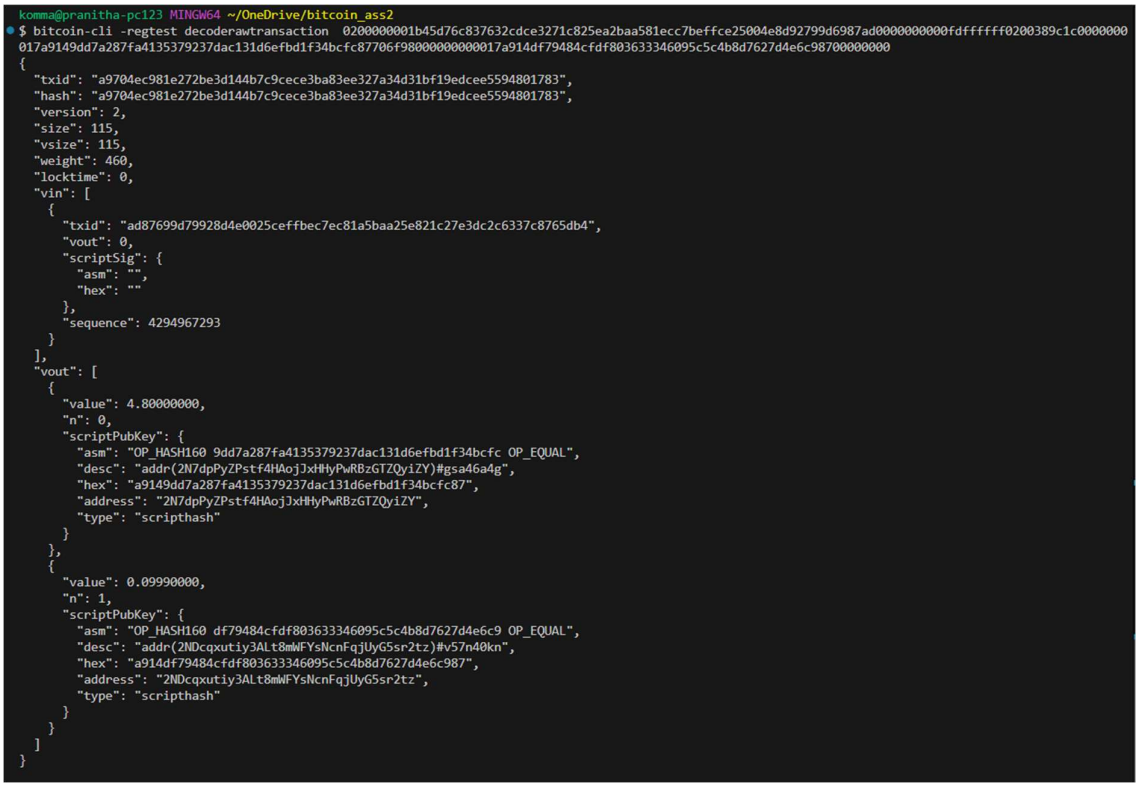
- 

- *Description*: Decoded output of decoderawtransaction [raw_tx] showing TXID, inputs, outputs, and B''s scriptPubKey.

Transaction 2 (B' → C')

- **Decoded Raw Transaction**:
  - TXID: [decoded_tx_BtoC['txid']]
  - Version: [decoded_tx_BtoC['version']]
  - Locktime: [decoded_tx_BtoC['locktime']]
  - Outputs: [decoded_tx_BtoC['vout']] (e.g., 4.8 BTC to C', change to B')
- **Locking Script (scriptPubKey) for C'**: [scriptPubKey_C]
- **Unlocking Script for B'**:
  - scriptSig: [scriptSig_B] (e.g., 160014{20-byte-pubkey-hash})
  - Witness: [scriptWitness_B] (e.g., [signature, public_key])

- 

- 

- *Description*: Decoded output of decoderawtransaction [raw_tx_B] showing TXID, inputs (referencing [txid_broadcast]), outputs, scriptSig, and witness data.

Script Analysis

P2SH-SegWit Structure

1. **Locking Script (scriptPubKey)**:
    - Format: OP_HASH160 <20-byte script hash> OP_EQUAL
    - Hex Example: a914{script-hash}87
    - Purpose: Locks funds to a script hash, requiring the redeem script to match.
2. **Unlocking Script**:
    - **scriptSig**: Pushes the redeem script (e.g., 0014{public-key-hash} for P2WPKH)
    - **Witness**: Contains signature and public key, separated for SegWit efficiency
    - Example:
        - scriptSig: 16 00 14 {20-byte-hash}
        - Witness: [72-byte-signature, 33-byte-public-key]

3. **Validation Mechanism**:
    - **P2SH**: Verifies the redeem script's hash matches the scriptPubKey.
    - **SegWit**: Validates the witness data against the redeem script (e.g., signature matches public key).
    - Execution: Combines hash verification and signature checking.

Transaction Validation

- **A' → B'**: A''s wallet signs the input, locking 4.9 BTC to B''s P2SH-SegWit script.
- **B' → C'**: B' unlocks its UTXO with redeem script and witness, sending 4.8 BTC to C'.

Bitcoin Debugger Analysis

Debugger Steps for A' → B' (Locking Script for B')

- **Input**: scriptPubKey [scriptPubKey_B]
- **Execution**:
    - Step 1: OP_HASH160 computes hash of redeem script
    - Step 2: <20-byte-hash> pushed to stack
    - Step 3: OP_EQUAL verifies match
- **Result**: TRUE (valid locking script)

[Insert Screenshot #3 Here]

- *Description*: Bitcoin debugger output (e.g., btcdeb or bitcoin-tx) showing step-by-step execution of [scriptPubKey_B], ending with TRUE.

Debugger Steps for B' → C' (Unlocking Script)

- **Input**: scriptSig [scriptSig_B] + Witness [scriptWitness_B]
- **Execution**:
    - Step 1: scriptSig pushes 0014{public-key-hash}
    - Step 2: Hash of redeem script verified against previous scriptPubKey
    - Step 3: Witness signature validated against public key
- **Result**: TRUE (valid unlocking)

[Insert Screenshot #4 Here]

- *Description*: Debugger output showing stack operations for [scriptSig_B] and [scriptWitness_B], confirming successful validation.

Conclusion

The script executed two P2SH-SegWit transactions:

- **A' → B'**: TXID [txid_broadcast], funded B' with 4.9 BTC

- **B' → C'**: TXID [txid_broadcast_B], spent A'→B' output to send 4.8 BTC to C' P2SH-SegWit combines script flexibility with SegWit's efficiency (reduced size, malleability protection). The decoded scripts and debugger steps confirm proper locking and unlocking mechanisms.