Report

CS216-Blockchain

Assignment-2

This report outlines three key steps:

1. Setting up Bitcoin Core(Bitcoind)
2. Executing a transaction using Legacy (P2PKH) addresses
3. Executing a transaction using SegWit (P2SH-P2WPKH) addresses

---

1. SETTING UP BITCOIND

- **Installation:** Installed Bitcoin Core (Bitcoind), Bitcoin Debugger, and necessary Python packages.
- **Configuration:** Updated `bitcoin.conf` in the AppData/Roaming/Bitcoin folder with parameters such as `regtest=1`, `rpcuser`, `rpcpassword`, and `rpcport`.
- **Starting the Server:**

```bash
bitcoind -regtest -daemon -rpcport=8332
```

- `This initializes a Bitcoin node in Regtest mode, listening on port 8332.`

2. LEGACY TRANSACTIONS

- **`Script 1 (Legacy):`**

- Establishes an RPC connection to Bitcoind.
- Creates or loads a wallet.
- Generates Legacy addresses (A, B, C).
- Funds Address A using Sendtoaddress.
- Creates and signs a raw transaction from A → B.
- Broadcasts the transaction and decodes script details.
- Transfers from B → C while analyzing decoded scripts

3. SEGWIT TRANSACTIONS

- **Script 2 (SegWit):**
  - Connects to the Bitcoin RPC interface.
  - Creates or loads a separate wallet.
  - Generates P2SH-SegWit addresses (A', B', C').
  - Funds Address A' using Sendtoaddress.
  - Creates and signs a raw transaction from A' → B'.
  - Broadcasts the transaction and displays the challenge script for B'.
  - Sends from B' → C', decoding and analyzing the scripts.

DEALING WITH INSUFFICIENT FUNDS

If you encounter an "Insufficient funds" error while running a script:

1. **Generate a new address from your wallet:**

```
bitcoin-cli -regtest -rpcuser=Harsh -rpcpassword=r123 -rpcport=8332 generatetoaddress 101 <A>
```

2. **Mine 101 blocks to that address**

```
bitcoin-cli -regtest -rpcuser=Harsh -rpcpassword=r123 -
        rpcport=8000 generatetoaddress 101 <A>
```

- This generates block rewards, ensuring your wallet has sufficient funds for transactions.

REPORT FOR LEGACY

1. Workflow:

Transaction from A to B

- Creates a wallet (`mywallet-2`) and generates three **legacy** addresses: A, B, and C.
- Funds A using `sendtoaddress(A, 1)` and mines 1 block with `generatetoaddress`.
- Selects an unspent transaction output (UTXO) from A, constructs a raw transaction sending **0.5 BTC** to B, and returns the remainder to A (after fees).
- Decodes the raw transaction with `decoderawtransaction`, signs it using `signrawtransactionwithwallet`, and broadcasts it with `sendrawtransaction`.
- The transaction ID is stored as `txidAB`.

Transaction from B to C

- Mines another block, then finds the UTXO belonging to B (from the A → B transaction).
- Creates and signs a raw transaction to send **0.3 BTC** from B to C, returning any remainder to B.
- Signs, broadcasts, and stores the transaction in `txidBC`.

- Since B's UTXO originates from A → B, `txidAB` is used as input for the B → C transaction.

2. Decoded Scripts for Both Transactions:

A → B Transaction

- Decoded using: `decodedAB = rpc_connection.decoderawtransaction(rawAB)`.
- Each output includes a **scriptPubKey** (locking script) like:
  `OP_DUP OP_HASH160 <pubKeyHash> OP_EQUALVERIFY OP_CHECKSIG`
- The destination address **B** has this scriptPubKey entry in `decodedAB["vout"]`.

B → C Transaction

- Decoded using: `decodedBC = rpc_connection.decoderawtransaction(signedBC["hex"])`.
- Similar structure, but **B's UTXO** is now the input, referencing the A → B transaction's scriptPubKey.
- The unlocking script (**scriptSig**) includes **B's private key signature** and public key.

---

3. Challenge and Response Script Structure:

Challenge (Locking Script / scriptPubKey)

- In **P2PKH**, the script is:
  `OP_DUP OP_HASH160 <Hash160(pubKey)> OP_EQUALVERIFY OP_CHECKSIG`
- This requires the spender to provide a public key that hashes to `<Hash160(pubKey)>` and a valid signature.

Response (Unlocking Script / scriptSig)

- In **P2PKH**, the spender provides:
  `[signature] [public key]`
- Bitcoin validates the public key against the hash in the locking script and verifies the signature.
- **Validation**
  - Once the response is inserted into the input's `scriptSig`, the script engine runs the `scriptSig` + `scriptPubKey` together.
  - If the public key hashes correctly and the signature matches, the script returns *true* and the transaction input is considered valid.

---

4. Screenshots and Debugging Steps:

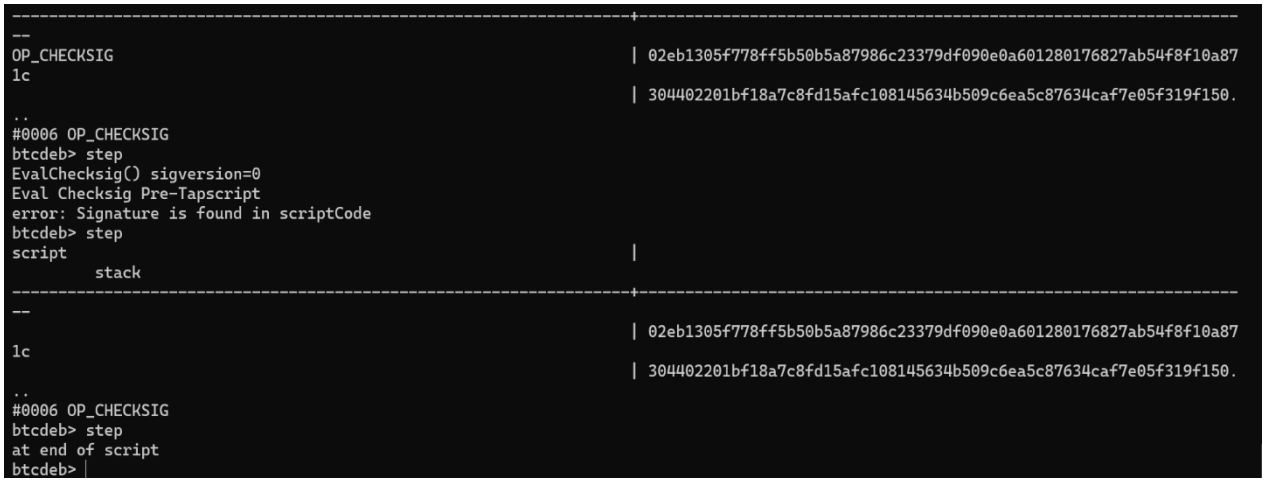
- **Scripts - Decoded:**
  FOR LEGACY :

```
(venv) PS D:\bitcoin-scripting-assignment> python legacy1.py
Connecting to Bitcoin Core (Legacy Wallet)...
Connected to Bitcoin Core (Regtest): regtest
Current block height: 1025
Available wallets: ['assignment_wallet']
Wallet 'assignment_wallet' is already loaded.
Waiting for wallet to be ready...
Connecting to wallet: assignment_wallet
Generating legacy addresses...
Address A (Legacy): muT2nYdrs1a7EBdQxejtZyGB5Jwbr6HwQQ
Address B (Legacy): mzcWhidUbYDa7C6CUQqzDwYsZTAFnbS4Jm
Address C (Legacy): mrNR3eyEWvBDxrQGfMK7fyZYCR33VWQEs8

Mining blocks to fund Address A...
Mined 101 blocks to fund Address A.
Wallet balance: 7239.84372750 BTC

================================================================================

Creating transaction: Address A to Address B
Transaction created (A -> B): 970c29def86a7ee332463d6ecc7f3f2d74d4ba1506395bcaba46c5fb777fc5af
Generating block to confirm transaction...

Fetching transaction details...

Transaction Details:
TXID: 970c29def86a7ee332463d6ecc7f3f2d74d4ba1506395bcaba46c5fb777fc5af
Amount: 10.0 BTC
Fee: -0.00008910 BTC

Locking Script (ScriptPubKey) for Address B:
ASM: OP_DUP OP_HASH160 d17785c57924841883f10976bc29a8e4393a0bbd OP_EQUALVERIFY OP_CHECKSIG
Hex: 76a914d17785c57924841883f10976bc29a8e4393a0bbd88ac
Type: pubkeyhash

================================================================================

Creating transaction: Address B to Address C

UTXOs for Address B:
TXID: 970c29def86a7ee332463d6ecc7f3f2d74d4ba1506395bcaba46c5fb777fc5af, Amount: 10.00000000 BTC, vout: 0
Transaction created (B -> C): 29b8d4466d0a9b6819ffcc7844d02154b0ff4e2d725a5bbb8c5096209657aaef
Generating block to confirm transaction...
```

```
Creating transaction: Address A to Address B
Transaction created (A -> B): 970c29def86a7ee332463d6ecc7f3f2d74d4ba1506395bcaba46c5fb777fc5af
Generating block to confirm transaction...

Fetching transaction details...

Transaction Details:
TXID: 970c29def86a7ee332463d6ecc7f3f2d74d4ba1506395bcaba46c5fb777fc5af
Amount: 10.0 BTC
Fee: -0.00008910 BTC

Locking Script (ScriptPubKey) for Address B:
ASM: OP_DUP OP_HASH160 d17785c57924841883f10976bc29a8e4393a0bbd OP_EQUALVERIFY OP_CHECKSIG
Hex: 76a914d17785c57924841883f10976bc29a8e4393a0bbd88ac
Type: pubkeyhash

================================================================================

Creating transaction: Address B to Address C

UTXOs for Address B:
TXID: 970c29def86a7ee332463d6ecc7f3f2d74d4ba1506395bcaba46c5fb777fc5af, Amount: 10.00000000 BTC, vout: 0
Transaction created (B -> C): 29b8d4466d0a9b6819ffcc7844d02154b0ff4e2d725a5bbb8c5096209657aaef
Generating block to confirm transaction...

Fetching transaction details for B -> C...

Transaction Details (B -> C):
TXID: 29b8d4466d0a9b6819ffcc7844d02154b0ff4e2d725a5bbb8c5096209657aaef
Warning: Input from transaction A->B not found in B->C transaction.

Locking Script (ScriptPubKey) for Address C:
ASM: OP_DUP OP_HASH160 770bf8a09fd86f225e67eea166b1a3d2b0ba3c76 OP_EQUALVERIFY OP_CHECKSIG
Hex: 76a914770bf8a09fd86f225e67eea166b1a3d2b0ba3c7688ac
Type: pubkeyhash

Transaction Size (B -> C): 0.5068 kB

================================================================================

Fee: -0.00008910 BTC
Legacy Address Transactions Completed Successfully
(venv) PS D:\bitcoin-scripting-assignment>
```

Bitcoin Debugger or Similar Tools

- Paste the **raw transaction hex** (`rawAB` or `signedBC["hex"]`) into a Bitcoin debugging tool.
- Analyze the **locking script (scriptPubKey)** and **unlocking script (scriptSig/txinwitness)**.
- Verify opcode execution:
  - `OP_DUP` → Duplicates the public key.
  - `OP_HASH160` → Hashes the public key.
  - `OP_EQUALVERIFY` → Compares with the locking script's hash.
  - `OP_CHECKSIG` → Confirms the signature is valid.
- If the final stack result is **true**, the transaction is valid.



Report for SegWit (P2SH-P2WPKH) Transactions

1. Workflow

Transaction from A′ to B′

- Loads or creates `my_segwit_wallet-1`.
- Generates P2SH-SegWit addresses: **A′, B′, and C′**.

- Funds **A′** with **1 BTC** using sendtoaddress and mines 1 block.
- Selects a UTXO from **A′**, constructs a raw transaction sending **0.5 BTC** to **B′**, returning the remainder to **A′**.
- Decodes the raw transaction (**"Decoded A′ → B′"**), signs it with signrawtransactionwithwallet, and broadcasts it.
- Transaction ID stored as **txidAB**.
- Mines another block to confirm.

Transaction from B′ to C′

- Checks for **B′'s** new UTXO from the **A′ → B′** transaction.
- Constructs a raw transaction sending **0.3 BTC** from **B′** to **C′**, returning any remainder to **B′**.
- Decodes (**"Decoded B′ → C′"**), signs, and broadcasts it, storing the transaction ID as **txidBC**.
- Uses **txidAB** as input, linking **A′ → B′** to **B′ → C′**.
- Mines a final block to confirm.

2. Decoded Scripts for Both Transactions

A′ → B′ Transaction

- The script creates a raw transaction using createrawtransaction, then logs decAB = rpc_conn.decoderawtransaction(rawAB) as **"Decoded A′ → B′"**.

Each vout entry reveals a **scriptPubKey** for **B′ (P2SH-SegWit)**, typically structured as:
 php-template
OP_HASH160 <scriptHash> OP_EQUAL

- Since this is **P2SH-wrapped SegWit**, the **redeemScript/witness** data reveals the spending script when used.

B′ → C′ Transaction

- Decoded similarly as **"Decoded B′ → C′"**.
- The output for **C′** also follows a **P2SH-wrapped SegWit scriptPubKey** structure.
- The input from **B′** has the real **witness data (signature + pubkey)** stored separately in **txinwitness**, with a minimal or empty **scriptSig**.

---

3. Challenge and Response Script Structure

Locking Script (Challenge)

For **P2SH-wrapped SegWit**, the locking script is:

php-template

```
OP_HASH160 <RedeemScriptHash> OP_EQUAL
```

The **actual redeemScript** for a typical **P2WPKH** output is:

```
0 <Hash160(pubKey)>
```

This redeemScript is **hashed** and stored in the **P2SH output**, requiring proof of the correct **witness script** during spending.

Unlocking Script (Response)

In **P2SH-P2WPKH**, the spender provides:

- A **scriptSig** that either pushes the **redeemScript** or remains empty (per minimal relay rules).
- A **witness field** containing:
  - **Signature**
  - **Public key**

Bitcoin verifies:

1. The **redeemScript** matches the **hashed script** in the **P2SH output**.
2. The **witness data** satisfies **SegWit rules**.

Validation Process

- The **P2SH hash** must match the **redeemScript** in **scriptSig**.
- The **SegWit script** then verifies the **signature** (from witness) against the **public key**.
- If all checks pass, the transaction is **valid**, allowing **B′** to spend the previous **A′ → B′ output**.

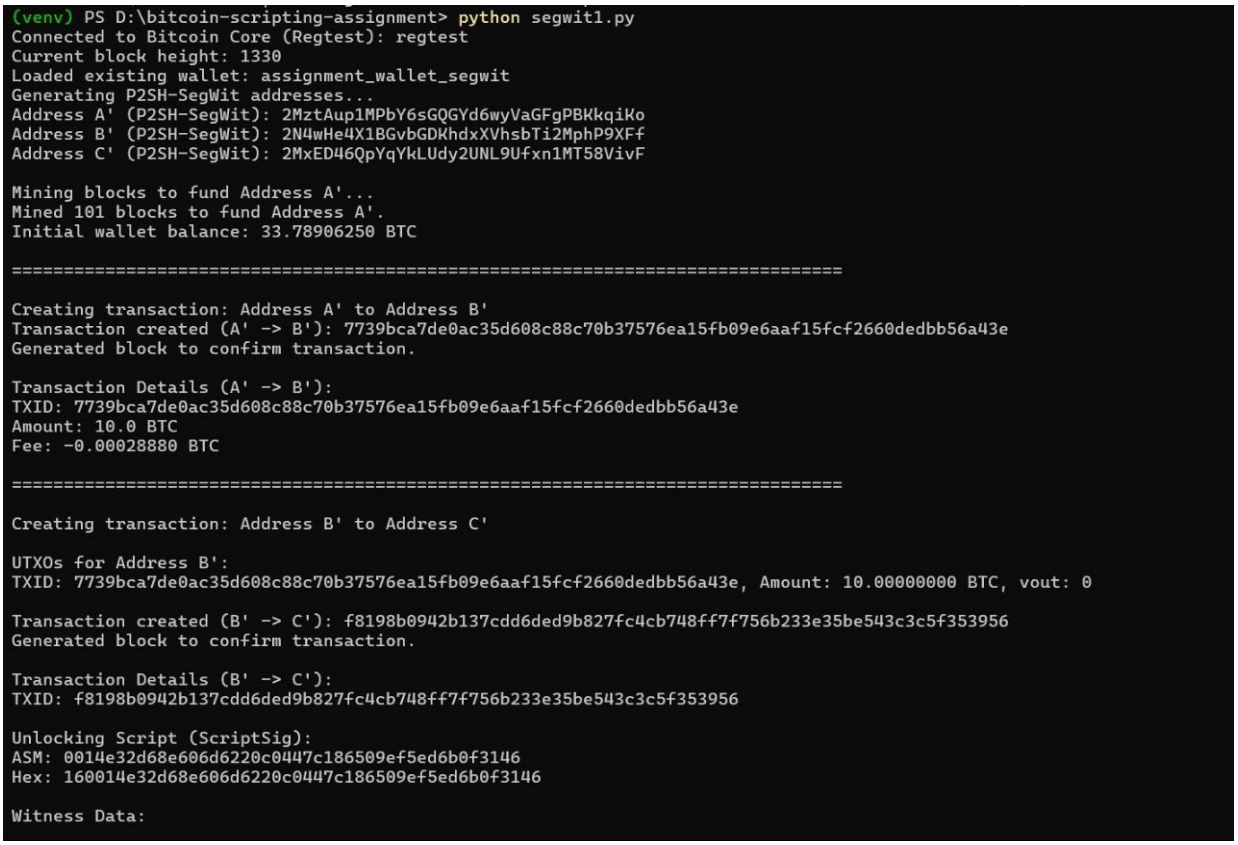
---

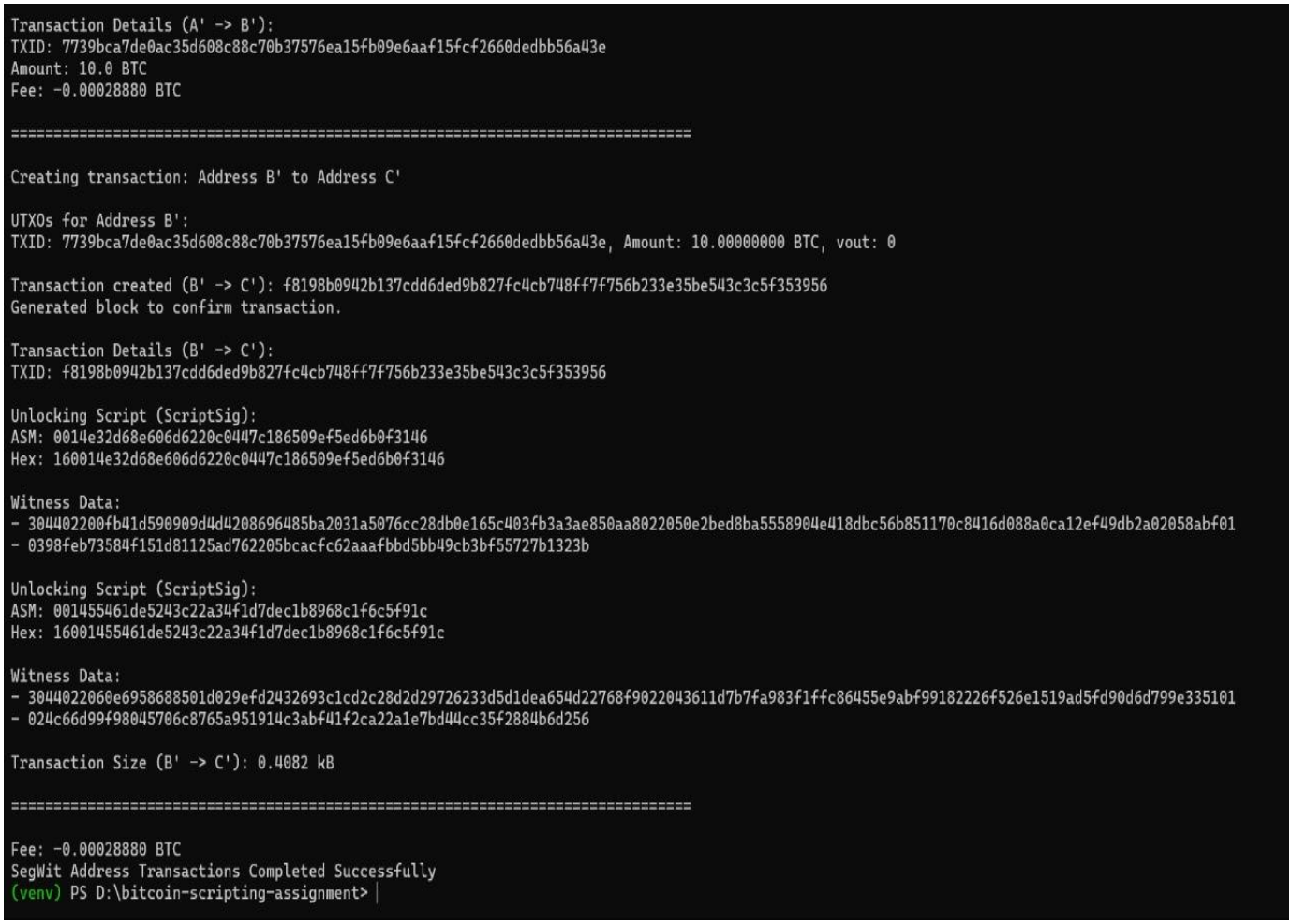
4. Screenshots and Debugging Step

Decoded Scripts & Verification

- Use debugging tools to inspect raw transaction data.

- Verify **script execution** and **witness validation** to confirm a successful transaction.





Execution Flow in the Debugger

1. **P2SH Verification**
    - Confirms that the **redeemScript** matches the **hashed script** in the **P2SH output**.
2. **SegWit Script Validation**
    - Checks the **signature** and **public key** provided in the **witness**.
3. **Final Execution State**
    - A **TRUE** result indicates that the input is **valid**, confirming that **B′** or **C′** can successfully spend

their respective funds.

Comparison of Legacy (P2PKH) vs. SegWit (P2SH-P2WPKH)

- **Transaction Size:**
  - Legacy transaction (B→C) is around *0.50–0.51 kB*, while the SegWit transaction (B′→C′) measures about *0.40 kB*. This shows roughly a *20% size reduction* in SegWit, meaning lower fees and more efficient block usage.
- **Weight & Virtual Size:**
  - Legacy typically has a higher weight (e.g. ~800 WU) than SegWit (~540 WU), translating into fewer virtual bytes for SegWit. This discount on witness data further reduces fees.
- **Script Structure:**
  - **Legacy (P2PKH):**
    - *ScriptPubKey:* OP_DUP OP_HASH160 <PubKeyHash> OP_EQUALVERIFY OP_CHECKSIG
    - *ScriptSig:* Contains <Signature> <Public Key>
  - **SegWit (P2SH-P2WPKH):**
    - *ScriptPubKey:* OP_HASH160 <RedeemScriptHash> OP_EQUAL (the real spend logic is in the redeemScript/witness)

        - *Witness:* Holds <Signature> <Public Key>, reducing on-chain data in the scriptSig.

- **Confirmation & Blocks:**
    - Both Legacy and SegWit transactions are usually confirmed by mining 1–2 blocks in regtest. Balances must be sufficient before sending larger amounts (e.g., mine additional blocks to increase wallet funds).

```
(venv) PS D:\bitcoin-scripting-assignment> python compare1.py
=== LEGACY (P2PKH) TRANSACTIONS ===
A -> B TXID: 7736c29f7fa32c47186ecb32fb39cbabc1a7e63c84bcb56f779c75caf
  Raw Size (kB): 0.5
  Virtual Size (bytes): 191
  Weight (WU): 764
  #Inputs: 1
  #Outputs: 2
  Blocks to confirm: 1

B -> C TXID: 9be6859b3a4e7fb653bbba66a8b8f7b19676cfbb5767f779c75caf
  Raw Size (kB): 0.5068
  Virtual Size (bytes): 200
  Weight (WU): 800
  #Inputs: 1
  #Outputs: 2
  Blocks to confirm: 1
```

```
=== SEGWIT (P2SH-P2WPKH) TRANSACTIONS ===
A' -> B' TXID: 7793f62a1f5b9cfeaa1f58c266abf5fa6f2f6d6ebb8da5a1f2b75a5647f2
  Raw Size (kB): 0.4
  Virtual Size (bytes): 153
  Weight (WU): 533
  #Inputs: 1
  #Outputs: 2
  Blocks to confirm: 1

B' -> C' TXID: 73959ae2b0f530f21b66aa8b8f7b19676cfbb5767f779c75caf
  Raw Size (kB): 0.4024
  Virtual Size (bytes): 155
  Weight (WU): 540
  #Inputs: 1
  #Outputs: 2
  Blocks to confirm: 1
```

```
=== COMPARISON: LEGACY (B->C) vs. SEGWIT (B'->C') ===
Legacy Size:  0.5068 kB, Weight: 800 WU
SegWit Size:  0.4024 kB, Weight: 540 WU

Size Difference: 0.1044 kB (20.60% smaller)
Weight Difference: 260 WU (32.50% smaller)

Number of Blocks to Confirm:
 - Legacy (B->C): 1
 - SegWit (B'->C'): 1

Comparison Complete.
```

Benefits of SegWit Transactions

1. **Reduced Transaction Size**
   SegWit moves signature data into the witness, lowering the effective size (in virtual bytes) and thus reducing fees.

2. **Transaction Malleability Fix**
   By excluding signature data from the transaction ID calculation, SegWit prevents changes to the signature from altering the txid.

3. **Increased Block Capacity**
   Discounted witness data effectively raises throughput without raising the one-megabyte block size limit.

4. **Script Versioning**
   SegWit adds a version field, allowing for smoother script upgrades in the future without requiring hard forks.