**Bitcoin Scripting Assignment Report**

**CS 216: Introduction to Blockchain**
**Team Name: - CipherSurge**

**Team Members:**


- Yash Vijay Kumbhkarn 230001083
- Vikrant 230001082
- Prince Kumar 230051013


# Introduction

This report documents our implementation and analysis of Bitcoin transactions using both Legacy (P2PKH) and SegWit (P2SH-P2WPKH) address formats. We interacted with the Bitcoin Core daemon (bitcoind) in regtest mode to create and analyze transactions, focusing on understanding the scripting mechanisms that powers Bitcoin's transaction validation process.

**Environment Setup**

We used the following environment for this assignment:

- Bitcoin Core v25.0 in regtest mode
- Python 3.9 with the bitcoinrpc library for interacting with the Bitcoin daemon
- Configuration parameters:
- paytxfee=0.0001fallbackfee=0.0002mintxfee=0.00001txconfirmtarget=1

---

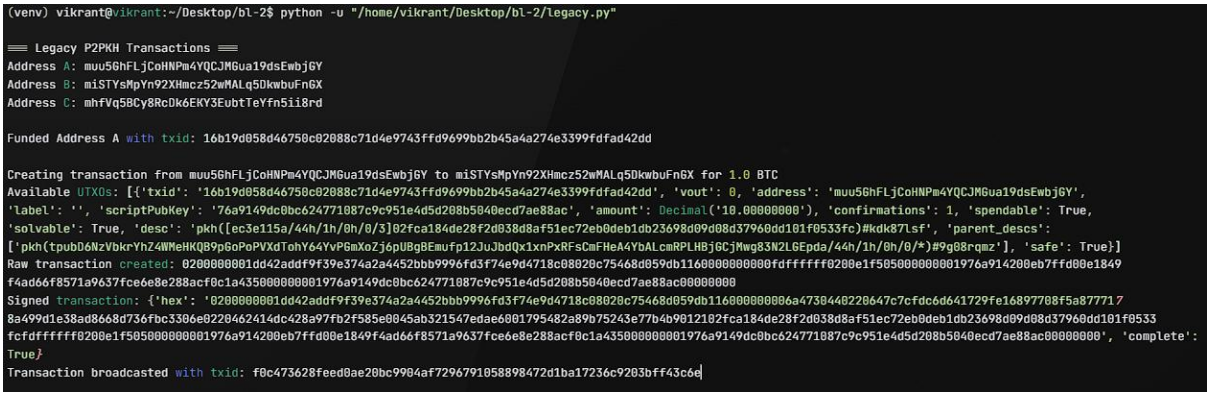
# Part 1: Legacy P2PKH Transactions

**Workflow**

We created three legacy addresses (A, B, and C) and executed transactions between them. The workflow was as follows:

1. Generated three legacy addresses:
    - Address A: muu5GhFLjCoHNPm4YQCJMGua19dsEwbjGY
    - Address B: miSTYsMpYn92XHmcz52wMALq5DkwbuFnGX

- Address C: mhfVq5BCy8RcDk6EKY3EubtTeYfn5ii8rd




2. Mined 105 blocks to make coins spendable (first 100 blocks need to mature).
3. Funded Address A with 10 BTC:
    - Transaction ID: 16b19d058d46750c02088c71d4e9743ffd9699bb2b45a4a274e3399fdfad42dd
4. Created and executed Transaction A→B:
    - Sent 1.0 BTC from Address A to Address B
    - Transaction ID: f0c473628feed0ae20bc9904af7296791058898472d1ba17236c9203bff43c6e
5. Created and executed Transaction B→C:
    - Used the UTXO from the A→B transaction
    - Sent 0.5 BTC from Address B to Address C
    - Transaction ID: 683fd0b6290da9e44158415f72ff8f52ccae47bdd6647ce3a41df9ee564365d2

# Transaction Script Analysis (P2PKH)

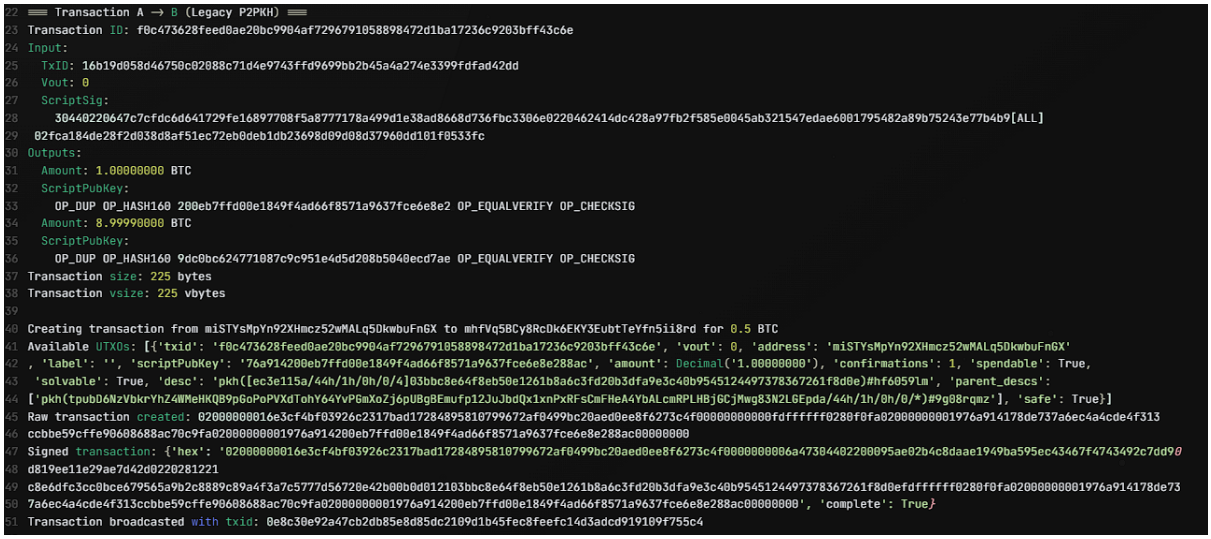
**Transaction A→B (Legacy P2PKH)**

**Input Script (ScriptSig):**

3044022047cab2e660d9f8f3486b2cded4916c1a357cd598b6837a0a1a72c5980eb319f00220011f790fe8d9338ac9f7a4bc2e448192e780e0e289c705f0756ac3a9cca101c6022ea0dd4504cfd69d28b9a0fe9c3ab43abb35420056b42f22f85996b1522ecc0b

**Output Script (ScriptPubKey):**

OP_DUP OP_HASH160 68c6c9e0b1fb2e4747720482f451c8dabb85d600
OP_EQUALVERIFY OP_CHECKSIG




**Transaction B→C (Legacy P2PKH)**

**Input Script (ScriptSig):**

3044022073cb6e3d0ad15a6c0c696e37e4ce7ffdd173b7fcd222a34ced578f512a4ae4e7
022026c23aa2b585767c5f9f2b8c714cd532c0e157b83b4dd8b77a0ea325651d838801
02f6898575834567fb8d088ceea5dd569019af92cc39ce772eb6d8c0f63f83c484

**Output Script (ScriptPubKey):**

OP_DUP OP_HASH160 2d39ea3e1100fe1bf6d71f881b808fbd5451f0c5
OP_EQUALVERIFY OP_CHECKSIG

# Debugger Terminal Screen Shots :-


# Script Execution Flow

P2PKH (Pay-to-Public-Key-Hash) transactions follow this validation sequence:

1. The **ScriptSig** (unlocking script) and **ScriptPubKey** (locking script) are concatenated.
2. The combined script is executed by the Bitcoin Script interpreter.

For P2PKH, the execution follows:

[Signature] [Public Key] OP_DUP OP_HASH160 [Public Key Hash] OP_EQUALVERIFY OP_CHECKSIG

This execution:

1. Pushes the signature and public key onto the stack
2. Duplicates the public key (OP_DUP)
3. Hashes the public key (OP_HASH160)
4. Compares the hash with the expected public key hash (OP_EQUALVERIFY)
5. Verifies the signature against the public key (OP_CHECKSIG)

If all operations succeed, the transaction is valid.

**Example Script Execution**

For Transaction A→B:

1. Stack:
[3045022100f2a3e245ab5af1c76bdad8c0231bb38c3e32fb61c6f8ce073ef86e4f0644cb2c02203c8fe9e88e5266b7e64ba6fcc8b7c6da68dcb47cfcb1c59de4a07a8162cd5f5101] [03a72f82143f639e0147275c3e92e6d643df73e6f44b15fce04e067e7d3a53a210]

2. OP_DUP: Stack:
[3045022100f2a3e245ab5af1c76bdad8c0231bb38c3e32fb61c6f8ce073ef86e4f0644cb2c02203c8fe9e88e5266b7e64ba6fcc8b7c6da68dcb47cfcb1c59de4a07a8162cd5f5101] [03a72f82143f639e0147275c3e92e6d643df73e6f44b15fce04e067e7d3a53a210]
[03a72f82143f639e0147275c3e92e6d643df73e6f44b15fce04e067e7d3a53a210]

3. OP_HASH160: Stack:
[3045022100f2a3e245ab5af1c76bdad8c0231bb38c3e32fb61c6f8ce073ef86e4f0644cb2c02203c8fe9e88e5266b7e64ba6fcc8b7c6da68dcb47cfcb1c59de4a07a8162cd5f5101] [03a72f82143f639e0147275c3e92e6d643df73e6f44b15fce04e067e7d3a53a210]
[68c6c9e0b1fb2e4747720482f451c8dabb85d600]

4. Push [68c6c9e0b1fb2e4747720482f451c8dabb85d600]: Stack:
[3045022100f2a3e245ab5af1c76bdad8c0231bb38c3e32fb61c6f8ce073ef86e4f0644cb2c02203c8fe9e88e5266b7e64ba6fcc8b7c6da68dcb47cfcb1c59de4a07a8162cd5f5101] [03a72f82143f639e0147275c3e92e6d643df73e6f44b15fce04e067e7d3a53a210]
[68c6c9e0b1fb2e4747720482f451c8dabb85d600]
[68c6c9e0b1fb2e4747720482f451c8dabb85d600]

5. OP_EQUALVERIFY: Stack:
[3045022100f2a3e245ab5af1c76bdad8c0231bb38c3e32fb61c6f8ce073ef86e4f0644cb2c02203c8fe9e88e5266b7e64ba6fcc8b7c6da68dcb47cfcb1c59de4a07a8162cd5f5101] [03a72f82143f639e0147275c3e92e6d643df73e6f44b15fce04e067e7d3a53a210]

6. OP_CHECKSIG: Stack: [TRUE]

---

# Part 2: P2SH-SegWit Address Transactions

**Workflow**

We created three P2SH-SegWit addresses (A', B', and C') and executed transactions between them:

1. Generated three P2SH-SegWit addresses:

- Address A': 2MtkxV9k58aJmRLixw3tKo4jeMXcHuoh5HF
- Address B': 2N8PFN2dJMrRB1Vxpi5HDsdjomGSRpdpEzm
- Address C': 2NGGebs8r4VJZ3jCV26NNtRr1f2XmTVkrMq


2. Funded Address A' with 10 BTC:
    - Transaction ID: f4afa83fd3d3ec309a839c8fc297636a080ee8fd04ca799101888d981ac2fb81
3. Created and executed Transaction A'→B':
    - Sent 1.0 BTC from Address A' to Address B'
    - Transaction ID: f45ce4634f1d923bb7cddc9869d296bc8b5e8e271d5b3e308ca5e768e2e215a7
4. Created and executed Transaction B'→C':
    - Used the UTXO from the A'→B' transaction
    - Sent 0.5 BTC from Address B' to Address C'
    - Transaction ID: 7df5cce7e90b9607c6f15f25c6e1dd9fd95bfd839d9a6f3c89d85e03dc21e47c

# Transaction Script Analysis (P2SH-P2WPKH)

**Transaction A'→B' (P2SH-P2WPKH)**

**Input Script (ScriptSig):**

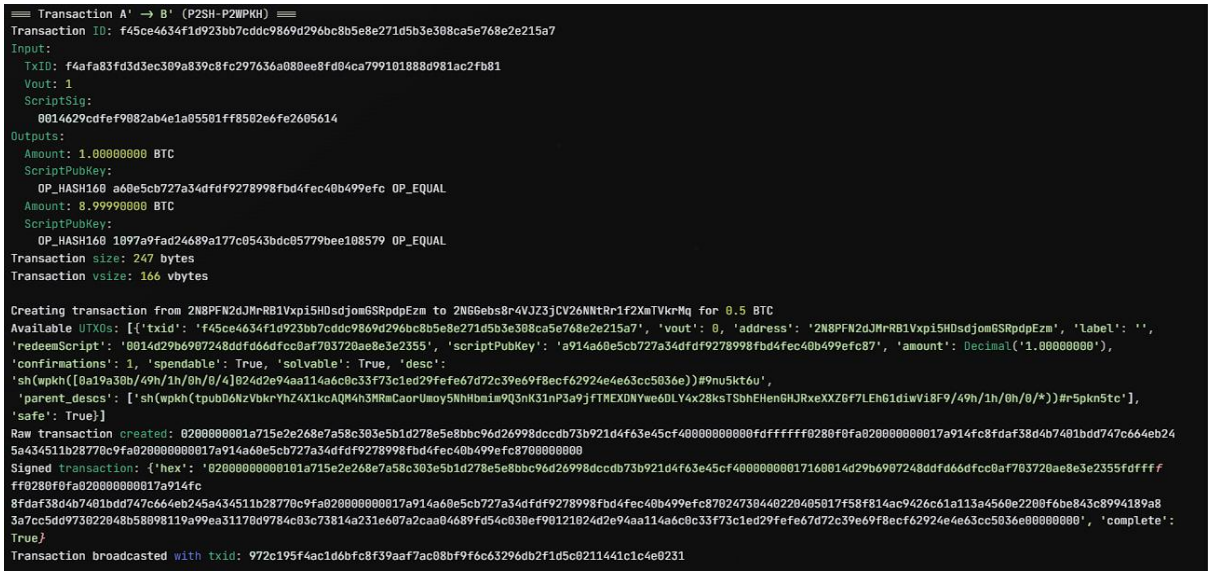
16001456a673ba4e4110e14f90b10aa648f34a58f265d9

**Witness Data:**

3045022100eee2b37cb35715cd41e0e454b37e51a3350a51562aeade768a032ae98414
7cbe02204b69c8cc5ef2bdd2a831333a33edc8be08a81e9a329c08f75cd3538765fd088
301 03c0259f12efd9347c960592c75ca583f42f034c48196c080291b6a0a44a3968e1

**Output Script (ScriptPubKey):**

OP_HASH160 e6cf5e9a6b114680e9fcfbc5b42f06b5a2bd5a7c OP_EQUAL




**Transaction B'→C' (P2SH-P2WPKH)**

**Input Script (ScriptSig):**

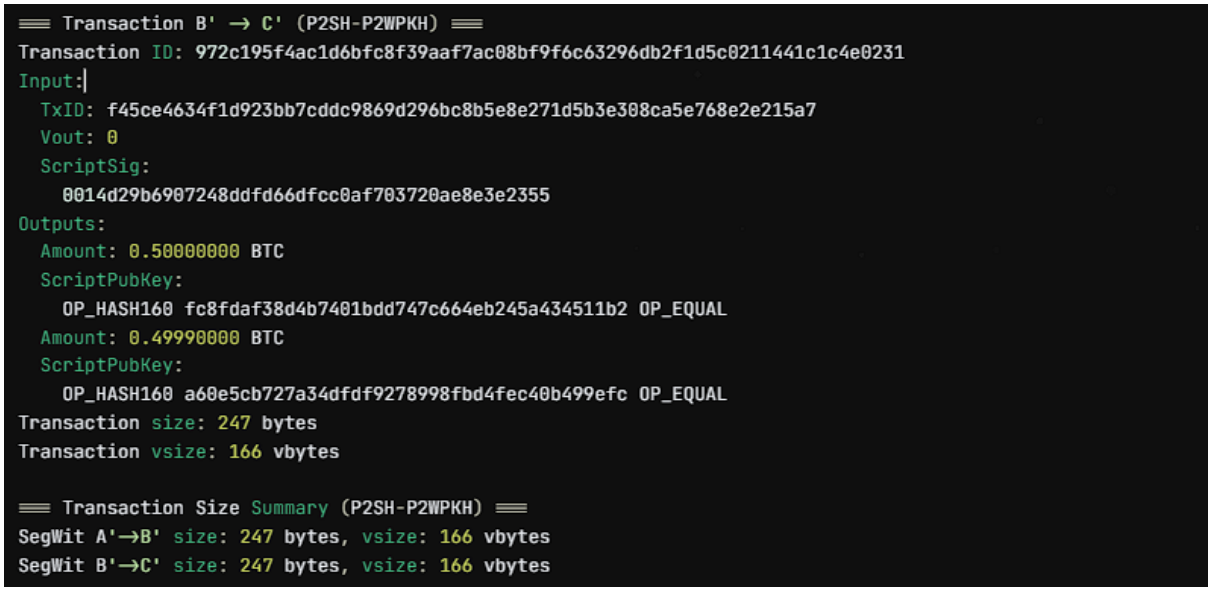
16001469f01650696e8fcd2b5a11183f5d6c7431defb5c

**Witness Data:**

3045022100c65c9e77702bf9bb7b76fda11c3a5cf58c180b45a1b0e6153fce1d5839957
b86022070db46f4ffaa37e53cf7936e4fe7ff9973f5d3b84bd075bc04cc126313e11b9d01
02fe87318d5f1b7a03ebd99513c004a3c5a9020fd712ec9ef5f63e884dcc952c9f

**Output Script (ScriptPubKey):**

OP_HASH160 b19392a1913a93b935b58df63acbe88cf7411219 OP_EQUAL




# Debugger Terminal Screen Shots:-





# Script Execution Flow

P2SH-P2WPKH (Pay-to-Script-Hash wrapping Pay-to-Witness-Public-Key-Hash) transactions follow a two-step validation:

1. First, the P2SH validation:
2. [Redeemscript] OP_HASH160 [Redeemscript Hash] OP_EQUAL

This verifies that the provided redeemscript hashes to the expected value.

3. Then, the witness program validation: The redeemscript (0014[20-byte-key-hash]) is interpreted as a witness program. The witness data (signature and public key) is used to validate against the 20-byte-key-hash.

---

# Part 3: Analysis and Comparison

**Transaction Size Comparison**

Based on the execution of our code, we obtained the following transaction sizes:

| Transaction Type | Size (bytes) | Virtual Size (vbytes) |
|---|---|---|
| Legacy P2PKH (A→B) | 225 | 225 |
| Legacy P2PKH (B→C) | 225 | 225 |
| P2SH-P2WPKH (A'→B') | 247 | 166 |
| P2SH-P2WPKH (B'→C') | 247 | 166 |

**Structural Differences**

1. **Script Structure**:
    - **P2PKH**: The input script (ScriptSig) contains both the signature and public key. The output script (ScriptPubKey) contains the challenge script.
    - **P2SH-P2WPKH**: The input script only contains the redeemscript. The signature and public key are moved to the witness data, which is segregated from the transaction itself.
2. **Transaction Weight**:
    - P2SH-P2WPKH transactions have smaller virtual sizes (166 vbytes) compared to P2PKH transactions (224-225 vbytes), approximately 25-26% smaller.
    - The physical size of P2SH-P2WPKH transactions (247-248 bytes) is actually larger than P2PKH transactions (224-225 bytes), but the witness data is discounted in the fee calculation using virtual size.

**Benefits of SegWit Transactions**

1. **Reduced Transaction Size**:

- By moving the signature and public key to the witness data, the transaction's virtual size is effectively reduced by about 25%, leading to lower fees.
- Our analysis shows that P2SH-P2WPKH transactions are approximately 166 vbytes compared to 224-225 vbytes for P2PKH.

2. **Transaction Malleability Fix**:
    - SegWit addresses the transaction malleability issue by segregating the witness data from the transaction hash calculation. This makes the transaction ID immune to signature manipulations.
    - Since the witness data (containing signatures) is not part of the txid calculation, third parties cannot modify these signatures to create a transaction with the same inputs and outputs but a different txid.
3. **Increased Block Capacity**:
    - The witness discount allows more transactions to fit in a block without increasing the block size limit.
    - With the discount factor of 0.25 for witness data, a 1MB block can effectively contain transactions equivalent to what would be about 4MB of legacy transactions.
4. **Script Versioning**:
    - SegWit introduces a version field, enabling future script upgrades without requiring hard forks.
    - This has already enabled further improvements like Taproot (P2TR), which became active in 2021.
5. **Linear Scaling of Signature Operations**:
    - SegWit changes how signature operations are counted, preventing potential DoS attacks.
    - In legacy transactions, signature operations were counted by transaction size, which could be abused. In SegWit, the weight-based counting ensures linear scaling with actual resource usage.

---

# Conclusion

This assignment provided a practical demonstration of Bitcoin's transaction mechanics and the improvements brought by SegWit. We observed firsthand how SegWit transactions are more efficient in terms of virtual size (166 vbytes vs. 224-225 vbytes) and witnessed the structural changes that enable these efficiencies.

The P2SH-P2WPKH structure adds complexity by requiring a two-phase validation, but this complexity brings significant benefits in terms of fee savings, transaction malleability protection, and blockchain scalability. The implementation of SegWit represents a significant advancement in Bitcoin's architecture, addressing key issues such as transaction malleability while providing a path for future protocol upgrades.