**Bitcoin Scripting Assignment Report**

**Team members:**

| **Name** | **Roll no.** |
| --- | --- |
| **Ayush Kumar** | **230001012** |
| **Hemsai** | **230001079** |
| **Shubham prajapati** | **230005047** |

**1. Introduction**

This report presents an in-depth analysis of Bitcoin transaction scripting using both Legacy (P2PKH) and SegWit (P2SH-P2WPKH) address formats. We aim to comprehensively investigate the transaction creation process, validate Bitcoin transactions, compare transaction sizes, and analyze the underlying scripts that power these transactions. This analysis provides insights into the technical improvements SegWit brings to the Bitcoin network.

**2. Part 1: Legacy Address Transactions**

**2.1 Environment Setup**

We configured our testing environment as follows:

* Installed and configured bitcoind on the system
* Updated bitcoin.conf with the following essential parameters:
  + paytxfee=0.0001
  + fallbackfee=0.0002
  + mintxfee=0.00001
  + txconfirmtarget=6
  + rpcusername= username
  + rpcpassword=password
* Started bitcoind in regtest mode to create a controlled testing environment

**2.2 Transactions and Scripts**

**Step 1: Create and Fund Addresses**

* Generated three Legacy addresses (A, B, and C) using the Bitcoin Core wallet
* Funded Address A using the sendtoaddress command to create initial UTXO

**Step 2: Create Transaction from A to B**

* Created a raw transaction transferring funds from Address A to Address B
* Decoded the raw transaction to analyze the ScriptPubKey of Address B
* Signed the transaction with the private key of Address A and broadcast it to the network

**Step 3: Create Transaction from B to C**

* Retrieved unspent transaction outputs (UTXOs) for Address B using listunspent
* Created and broadcast a transaction moving funds from Address B to Address C
* Extracted and analyzed the ScriptSig to understand the challenge-response mechanism

**2.3 Results and Analysis**

**Transaction IDs:**

* Txid (Funding A): 71be9e1fb0daba1e858ae0b4519b46730caf366dee42b7ef5447e98d72d2353a
* Txid (A → B): 9a6294fe3e452d81e4c23ff6ff751bc6bf7f280593db3e996b0448148a779168
* Txid (B → C): 7c1f800d3b557fef176715ebd1815e3f03bdfb3b46e81ca4dcc0782d2296bdee

**Challenge Script (ScriptPubKey for B):**

* Hex: 76a914adcd7a75014356f74d8b39c7ef098e374f5e4cdd88ac
* Decoded:

{

"asm": "OP\_DUP OP\_HASH160 adcd7a75014356f74d8b39c7ef098e374f5e4cdd OP\_EQUALVERIFY OP\_CHECKSIG",

"desc": "addr(mwMwM8J5Yc72aoebzABdVnPKhR8jX1EhcT)#dm6xnx2j",

"address": "mwMwM8J5Yc72aoebzABdVnPKhR8jX1EhcT",

"type": "pubkeyhash",

"p2sh": "2N8D1ZfH11zAdsUpqgb1dPPzDvavpsYRXR6",

"segwit": {

"asm": "0 adcd7a75014356f74d8b39c7ef098e374f5e4cdd",

"desc": "addr(bcrt1q4hxh5agpgdt0wnvt88r77zvwxa84unxa5l3vsv)#c4eh5mfq",

"hex": "0014adcd7a75014356f74d8b39c7ef098e374f5e4cdd",

"address": "bcrt1q4hxh5agpgdt0wnvt88r77zvwxa84unxa5l3vsv",

"type": "witness\_v0\_keyhash",

"p2sh-segwit": "2MtdoS7SPZA1pnVNq5Tgt2Q82s6PLkPA3eu"

}

}

**Response Script (ScriptSig):**

* Hex: 473044022001e4174a2f98d3a969a2ee9e4b412665c64f1a93465db367047967e424e9a82202201fd2a6f69af131e130be5ffb62568c8e355b5541f69520d3903cea5577a0da2401210345479c6aab6019dce2e66c793cd15adf7c974eec394698589b0992e59cca598f
* Decoded:

{

"asm": "3044022001e4174a2f98d3a969a2ee9e4b412665c64f1a93465db367047967e424e9a82202201fd2a6f69af131e130be5ffb62568c8e355b5541f69520d3903cea5577a0da2401 0345479c6aab6019dce2e66c793cd15adf7c974eec394698589b0992e59cca598f",

"desc": "raw(473044022001e4174a2f98d3a969a2ee9e4b412665c64f1a93465db367047967e424e9a82202201fd2a6f69af131e130be5ffb62568c8e355b5541f69520d3903cea5577a0da2401210345479c6aab6019dce2e66c793cd15adf7c974eec394698589b0992e59cca598f)#lakh0jpd",

"type": "nonstandard",

"p2sh": "2N1wGYisV4pNcbvFPcZw2k7mZ3j4CB3q8gR",

"segwit": {

"asm": "0 93f426fc9d4fcc6071f7dc2288d32ffcc03dbb875907cd5bf70c74469b98659e",

"desc": "addr(bcrt1qj06zdlyaflxxqu0hms3g35e0lnqrmwu8tyru6klhp36ydxucvk0q9te5ps)#5g7fw07x",

"hex": "002093f426fc9d4fcc6071f7dc2288d32ffcc03dbb875907cd5bf70c74469b98659e",

"address": "bcrt1qj06zdlyaflxxqu0hms3g35e0lnqrmwu8tyru6klhp36ydxucvk0q9te5ps",

"type": "witness\_v0\_scripthash",

"p2sh-segwit": "2MyxTr1YKj2HG2dAWnnM9aQe2Yup5KX3rZJ"

}

}

**Screenshots of decoded scripts**

*Challenge script:*

A computer screen with white text

AI-generated content may be incorrect.

*Response script:*

A black screen with white text

AI-generated content may be incorrect.

**Steps for debugging script:**

*Step 1:* SSH login

* Username: guest
* IP Address: 10.206.4.201
* Password: root1234

*Step 2:* Debug using Bitcoin script debugger

* Execute: btcdeb -v 'Script' in terminal
* Where Script = RS+CS (Response Script + Challenge Script)
* Here + denotes String Concatenation (don’t use it in between)
* Enter **step** to get steps(repeat it until its verified)

**Screenshot of debugging:**

A black screen with white text

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A black screen with white text

AI-generated content may be incorrect.

**Structure and Validation Process**

*Structure of Challenge Script:* OP\_DUP OP\_HASH160 <pubKeyHash> OP\_EQUALVERIFY OP\_CHECKSIG

*Structure of Response Script:* <signature> <publicKey>

*Validation Process:*

1. The unlocking script (Response Script) is executed first, pushing the signature and public key onto the stack
2. The locking script (Challenge Script) is then executed, which:
   * Duplicates the public key (OP\_DUP)
   * Hashes the public key (OP\_HASH160)
   * Compares the hash with the stored pubKeyHash (OP\_EQUALVERIFY)
   * Verifies the signature against the public key (OP\_CHECKSIG)
3. If all conditions are met and the final result is "true," the transaction is considered valid

This multi-step validation ensures that only the holder of the correct private key can produce the appropriate signature to unlock the funds, providing the foundational security mechanism for Bitcoin transactions.

**3. Part 2: P2SH-SegWit Address Transactions**

**3.1 Setup and Transactions**

* Created three SegWit addresses (A', B', and C') using P2SH-P2WPKH format
* Funded address A' and performed analogous transactions as in Part 1
* Examined the structural differences in transaction formats

**3.2 Results and Analysis**

**Transaction IDs:**

* Funding A': 052181e267426a90779f96cfc3f519318fa214b303382adb4348cb3b03ce4e69
* A' → B': 1dfdab917b626ad665e0e50b23f2eabe65dddc141c8c2c074ee4efa31a92687c
* B' → C': c013e884d138097791a1397c171c912f60082e3b3847f6c20a48dc9a0a8de4bd

**Decoded Scripts**

In SegWit transactions (P2SH-P2WPKH), the locking and unlocking mechanisms differ significantly from Legacy transactions. Instead of storing unlocking scripts directly in ScriptSig, SegWit transactions use a separate witness field, which fundamentally changes how transaction data is structured and validated.

**Challenge Script (ScriptPubKey for Address B'):**

* Hex: a914b08597f3057fa8e0452f3b4acc73c9ebc98cb9d887
* Decoded:

{

"asm": "OP\_HASH160 b08597f3057fa8e0452f3b4acc73c9ebc98cb9d8 OP\_EQUAL",

"desc": "addr(2N9LaxHhrvo9vrpV1VgzpWqt4WmCc9AreCU)#zfg8dnng",

"address": "2N9LaxHhrvo9vrpV1VgzpWqt4WmCc9AreCU",

"type": "scripthash"

}

**Response Script (Witness Data for Unlocking):**

* ScriptSig (redeemScript): 1600142415b92034bf272e2505a807cc7c90006b6f05b9
* Witness: ['304402201f1a9f3c5ec10806ba3392b4580111aa26dacd4ed94d44feec35d18ffa1ca0670220181a485c376e6e15a11ed47d9e60c4754f70066dc5860bab0180d7e3df32480701', '03179147eb30520101ed514f52b3974de1dd2290465e35e011f7dfa226bd872262']
* Decoded:

{

"asm": "00142415b92034bf272e2505a807cc7c90006b6f05b9",

"desc": "raw(1600142415b92034bf272e2505a807cc7c90006b6f05b9)#yk5n9cht",

"type": "nonstandard",

"p2sh": "2N6cjSeoM6ZCYB6t4q6fYp5eNfSxjWvWtEH",

"segwit": {

"asm": "0 1835fe8c707ea15fb6dfa65e458c1a23e1a74370aebe2d0865411348284f19d3",

"desc": "addr(bcrt1qrq6larrs06s4ldkl5e0ytrq6y0s6wsms46lz6zr9gyf5s2z0r8fshkhzw3)#kksawq7l",

"hex": "00201835fe8c707ea15fb6dfa65e458c1a23e1a74370aebe2d0865411348284f19d3",

"address": "bcrt1qrq6larrs06s4ldkl5e0ytrq6y0s6wsms46lz6zr9gyf5s2z0r8fshkhzw3",

"type": "witness\_v0\_scripthash",

"p2sh-segwit": "2MsUd8Q33knYdFZZutEhKNHU7SDrhNTxgtv"

}

}

**Screenshots of decoding scripts**

*Challenge Script:*

A computer screen shot of a black screen

AI-generated content may be incorrect.

*Response Script:*

A computer code on a black background

AI-generated content may be incorrect.

**Screenshot of Bitcoin debugger validating transactions:**

A screenshot of a computer

AI-generated content may be incorrect.

A black screen with white text

AI-generated content may be incorrect.

**Structure And Validation**

*Structure of Challenge Script:* OP\_HASH160 <Hash(redeem\_script)> OP\_EQUAL

*Structure of Response Script:*

* scriptSig: Contains the redeem script (revealing the spending conditions)
* Witness Field: Contains the signatures and additional data required to satisfy the redeem script

**Validation Process:**

1. **Redeem Script Extraction:** The node extracts the redeem script from the input's scriptSig.
2. **Hash Verification:** It computes HASH160(redeem\_script) and compares it with the hash embedded in the challenge script (output). A match confirms the correct redeem script is provided.
3. **Script Execution:** The redeem script is executed with the unlocking data (from the witness). The transaction is considered valid if the script returns a true value.

This segregated witness structure provides both technical advantages and enhanced security features compared to legacy transactions.

**4. Part 3: Comparison and Analysis**

**4.1 Transaction Size Comparison**

| **Type** | **Transaction Size (Vbyte)** | **Transaction Weight** |
| --- | --- | --- |
| **Legacy (P2PKH)** | **225** | **900** |
| **SegWit (P2SH-P2WPKH)** | **166** | **661** |

SegWit transactions are notably smaller due to the segregation of witness data from the transaction body. This separation reduces the overall size of the transaction, as witness data is counted differently in the block weight calculation. The reduction in size directly translates to lower transaction fees and increased blockchain capacity.

**4.2 Script Structure Comparison**

**Legacy (P2PKH):**

* Challenge Script Size: 25 Vbytes (100 weight units)
* Response Script Size: 106 Vbytes (424 weight units)
* Uses ScriptPubKey with format: OP\_DUP OP\_HASH160 <PubKeyHash> OP\_EQUALVERIFY OP\_CHECKSIG
* ScriptSig contains full signature and public key, increasing transaction size

**SegWit (P2SH-P2WPKH):**

* Challenge Script Size: 23 Vbytes (92 weight units)
* Response Script Size: 23 Vbytes (92 weight units)
* Uses ScriptPubKey with format: OP\_HASH160 <RedeemScriptHash> OP\_EQUAL
* ScriptSig contains only the redeem script, while the witness field (counted at a discount) contains the signature and public key

**4.3 Benefits of SegWit**

1. **Efficiency and Cost:**
   * Reduces transaction size and corresponding fees
   * More transactions can fit into each block, increasing network throughput
2. **Security Improvements:**
   * Eliminates transaction malleability issues by moving signatures outside the transaction ID calculation
   * Provides a cleaner separation of signature data from transaction data
3. **Future Scalability:**
   * Enables second-layer solutions like the Lightning Network
   * Provides a framework for future protocol upgrades without requiring hard forks

**5. Conclusion**

This assignment successfully demonstrated the process of creating, analyzing, and comparing Bitcoin transactions using both Legacy and SegWit address formats. Through practical implementation and detailed script analysis, we have shown that SegWit transactions offer significant advantages in terms of efficiency, cost, and security.

The segregation of witness data not only reduces transaction sizes and associated fees but also eliminates transaction malleability, which was a critical issue in legacy Bitcoin transactions. This improvement has paved the way for second-layer scaling solutions and other protocol enhancements.