Report

Blockchain Assignment-2

This report focuses on three key procedures:

1. Setting up **bitcoind**
2. Making a transaction using Legacy (P2PKH) addresses
3. Making a transaction using SegWit (P2SH-P2WPKH) addresses

Setting up bitcoind

- **Installation**: Download and install Bitcoin Core (`bitcoind`), the Bitcoin Debugger, and any necessary Python packages.
- **Configuration**: Edit the `bitcoin.conf` file, typically located in the `AppData/Roaming/Bitcoin` folder, to include parameters such as `regtest=1`, `rpcuser`, `rpcpassword`, and `rpcport`.
- **Starting the Server**:
  bitcoind -server -regtest -rpcport=8000
  This launches a Bitcoin node in Regtest mode, listening on port 8000.

---

2. LEGACY TRANSACTIONS

- Script 1 (Legacy):
  - Connects to the bitcoind RPC interface.
  - 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 shows decoded scripts (locking/unlocking).
- Sends from B → C, decoding and analyzing scripts again.

---

3. SEGWIT TRANSACTIONS

- Script 2 (SegWit):
  - Connects to the bitcoind RPC interface.
  - Creates or loads a different 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 shows the challenge script for B'.
  - Sends from B' → C', decoding and analyzing the scripts again.

HANDLING INSUFFICIENT FUNDS

If you see an "Insufficient funds" error when running either script:

1. Get a new address from your loaded wallet: **bitcoin-cli -regtest -rpcuser=Harsh -rpcpassword=r123 -rpcport=8000 -rpcwallet="my_segwit_wallet-1" getnewaddress**

   Suppose this returns an address <A>.

2. Mine 101 blocks to that address: **bitcoin-cli -regtest -rpcuser=Harsh -rpcpassword=r123 -rpcport=8000 generatetoaddress 101 <A>**

This awards enough block rewards to your wallet to ensure sufficient balance for subsequent transactions.

REPORT FOR LEGACY:

1. Workflow:

- **Transaction from A to B**
    - After creating wallet **mywallet-2**, the script generates three new **legacy** addresses: **A**, **B**, and **C**.
    - It funds **A** by calling **sendtoaddress(A, 1)** and mines 1 block with **generatetoaddress**.
    - The script selects an unspent transaction output (UTXO) belonging to **A**, constructs a raw transaction sending **0.5 BTC** to **B**, and returns the remainder to **A** (minus a small fee).
    - This raw transaction is decoded with **decoderawtransaction**, then signed using **signrawtransactionwithwallet**, and finally broadcast with **sendrawtransaction**. The transaction ID is stored in **txidAB**.
- **Transaction from B to C**
    - The script mines another block, then checks for the UTXO now belonging to **B** (created by the **A** → **B** transaction).
    - It creates and signs another raw transaction to send **0.3 BTC** from **B** to **C**, returning any leftover back to **B**.
    - This raw transaction is signed, broadcast, and stored in **txidBC**.
    - Because **B**'s UTXO came from the **A** → **B** transaction, you see **txidAB** used as input for the **B** → **C** transaction.

---

2. Decoded Scripts for Both Transactions:

- **A → B Transaction**
    - Decoded via **decodedAB = rpc_connection.decoderawtransaction(rawAB)**.
    - Typically, each output includes a `scriptPubKey` (locking script) that looks like: **OP_DUP OP_HASH160 <pubKeyHash> OP_EQUALVERIFY OP_CHECKSIG**.
    - The destination address **B** has one of these `scriptPubKey` entries in **decodedAB["vout"]**.
- **B → C Transaction**
    - Decoded via **decodedBC = rpc_connection.decoderawtransaction(signedBC["hex"])**.
    - Similar structure, but now **B**'s UTXO is the input. The script locking that output was from the **A → B** transaction.
    - The unlocking script (`scriptSig`) or witness data references **B**'s private key signature plus public key.

---

3. Challenge and Response Script Structure:

- **Challenge (Locking Script / scriptPubKey)**
    - In P2PKH, it is typically:
      OP_DUP OP_HASH160 <Hash160(pubKey)> OP_EQUALVERIFY OP_CHECKSIG
    - This means the spender must provide a public key that hashes to `<Hash160(pubKey)>` and a valid signature to pass **OP_CHECKSIG**.
- **Response (Unlocking Script / scriptSig)**

    - For P2PKH, the spender includes:
      [signature] [public key]
    - When combined with the locking script, Bitcoin verifies that the provided public key matches the hash in the locking script and that the signature is valid for the transaction data.
- **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.

---

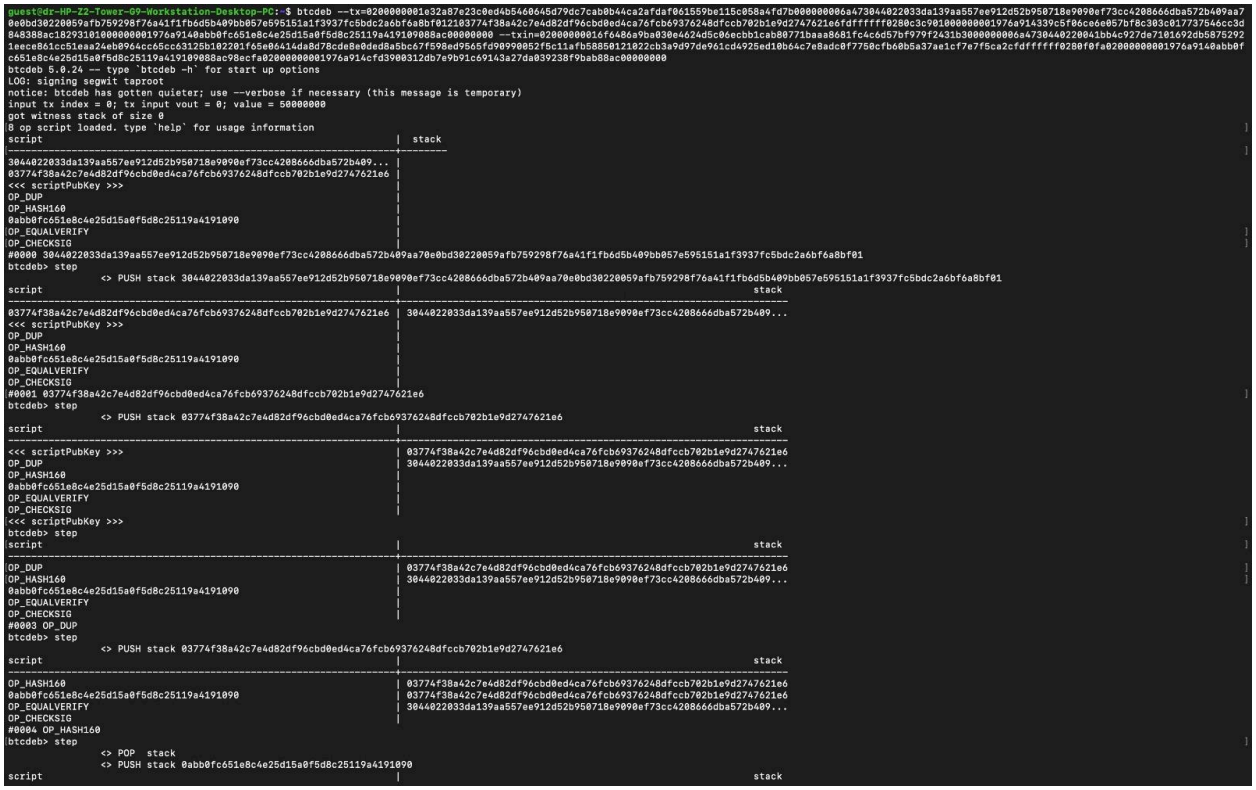
4. Screenshots and Debugging Steps:

- **Decoded Scripts:**
  FOR LEGACY :

```
PS C:\Users\Asus\Desktop\Blockchain> python script1.py
INFO:root:Addresses: A=mgDPjGBZZJRpf9AdKMko76EfV73HUS1aS3, B=myGTEtU58h5wu82LetcBuj8fS75T5kuGsG, C=mwMezTwwUwKU6uQ54re1bjktujd7aUgMcj
INFO:root:Funded A with txid=adf86ac49d292a41cc91e808bed58f5201d268b47c487582d0d55058e231b666
INFO:root:Decoded raw A->B: {
  "txid": "380624e914149fc259ace0b1d5273cd31dca83470f46aa5f99e638cedff8bb79",
  "hash": "380624e914149fc259ace0b1d5273cd31dca83470f46aa5f99e638cedff8bb79",
  "version": 2,
  "size": 119,
  "vsize": 119,
  "weight": 476,
  "locktime": 0,
  "vin": [
    {
      "txid": "adf86ac49d292a41cc91e808bed58f5201d268b47c487582d0d55058e231b666",
      "vout": 1,
      "scriptSig": {
        "asm": "",
        "hex": ""
      },
      "sequence": 4294967293
    }
  ],
  "vout": [
    {
      "value": "0.50000000",
      "n": 0,
      "scriptPubKey": {
        "asm": "OP_DUP OP_HASH160 c2b42fe5a75dc71004fcd0c4556a25242f03ad2e OP_EQUALVERIFY OP_CHECKSIG",
        "desc": "addr(myGTEtU58h5wu82LetcBuj8fS75T5kuGsG)#fe7d3aww",
        "hex": "76a914c2b42fe5a75dc71004fcd0c4556a25242f03ad2e88ac",
        "address": "myGTEtU58h5wu82LetcBuj8fS75T5kuGsG",
        "type": "pubkeyhash"
      }
    },
    {
      "value": "0.49999000",
      "n": 1,
      "scriptPubKey": {
        "asm": "OP_DUP OP_HASH160 07a5dec7812e1487fd209fdca4d94492b86fceba OP_EQUALVERIFY OP_CHECKSIG",
        "desc": "addr(mgDPjGBZZJRpf9AdKMko76EfV73HUS1aS3)#j0cs8wkr",
        "hex": "76a91407a5dec7812e1487fd209fdca4d94492b86fceba88ac",
        "address": "mgDPjGBZZJRpf9AdKMko76EfV73HUS1aS3",
        "type": "pubkeyhash"
      }
```

```
INFO:root:Broadcasted A->B, txid=a3a8a14bf3ec69f00e176d0faa1c8c6afdb571ee9e70f1fd882dc498e5a418cb
INFO:root:Broadcasted B->C, txid=0d51746efe17d9bddbb375753c3b54fcd53fad157524e434c5a666079414ca80
INFO:root:Decoded raw B->C: {
  "txid": "0d51746efe17d9bddbb375753c3b54fcd53fad157524e434c5a666079414ca80",
  "hash": "0d51746efe17d9bddbb375753c3b54fcd53fad157524e434c5a666079414ca80",
  "version": 2,
  "size": 225,
  "vsize": 225,
  "weight": 900,
  "locktime": 0,
  "vin": [
    {
      "txid": "a3a8a14bf3ec69f00e176d0faa1c8c6afdb571ee9e70f1fd882dc498e5a418cb",
      "vout": 0,
      "scriptSig": {
        "asm": "304402200df0957dc727cbedee196de3661418b17f9ffa0e648e5f49ecc0498f6e2c28e90220195f2d6f9dfd75a1dde843f22d27f51ecdf41d25b54d8b358ce856c0c96ec80c[A
LL] 038a43bf75a96c4e1221157945edc0e445c06ce85f6e18511af4924bf37779e3d1",
        "hex": "47304402200df0957dc727cbedee196de3661418b17f9ffa0e648e5f49ecc0498f6e2c28e90220195f2d6f9dfd75a1dde843f22d27f51ecdf41d25b54d8b358ce856c0c96ec80c
0121038a43bf75a96c4e1221157945edc0e445c06ce85f6e18511af4924bf37779e3d1"
      },
      "sequence": 4294967293
    }
  ],
  "vout": [
    {
      "value": "0.30000000",
      "n": 0,
      "scriptPubKey": {
        "asm": "OP_DUP OP_HASH160 adbfd329d811c956dfd03d168644341338cc6309 OP_EQUALVERIFY OP_CHECKSIG",
        "desc": "addr(mwMezTwwUwKU6uQ54re1bjktujd7aUgMcj)#4ucdgstf",
        "hex": "76a914adbfd329d811c956dfd03d168644341338cc630988ac",
        "address": "mwMezTwwUwKU6uQ54re1bjktujd7aUgMcj",
        "type": "pubkeyhash"
      }
    },
    {
      "value": "0.19999000",
      "n": 1,
      "scriptPubKey": {
        "asm": "OP_DUP OP_HASH160 c2b42fe5a75dc71004fcd0c4556a25242f03ad2e OP_EQUALVERIFY OP_CHECKSIG",
        "desc": "addr(myGTEtU58h5wu82LetcBuj8fS75T5kuGsG)#fe7d3aww",
        "hex": "76a914c2b42fe5a75dc71004fcd0c4556a25242f03ad2e88ac",
        "address": "myGTEtU58h5wu82LetcBuj8fS75T5kuGsG",
        "type": "pubkeyhash"
```

- **Bitcoin Debugger or Similar Tools:**
    - You can copy the raw transaction hex (**rawAB** or **signedBC["hex"]**) into a Bitcoin debugging site or tool.
    - Step through the challenge (locking script) and the response (signature + pubkey) in the scriptSig or `txinwitness`.
    - Confirm each opcode executes successfully:
        - **OP_DUP** duplicates the public key,
        - **OP_HASH160** applies the hash,
        - **OP_EQUALVERIFY** compares it with the hash in the locking script,
        - **OP_CHECKSIG** verifies the signature is correct.
    - The final stack result should be *true*, indicating the transaction is valid.



Report for SegWit (P2SH-P2WPKH) Transactions Using the Provided Script

---

1. Workflow

- **Transaction from A′ to B′**
    - The script creates or loads a wallet named **my_segwit_wallet-1**.
    - It generates three **P2SH-SegWit** addresses: **A′**, **B′**, and **C′**.
    - It funds **A′** with **1 BTC** by calling **sendtoaddress** and mines 1 block to confirm.
    - A UTXO from **A′** is then used to create a raw transaction sending **0.5 BTC** to **B′**, returning any remainder to **A′**.
    - The script decodes that raw transaction (logged as “Decoded A'->B'”) and signs it with **signrawtransactionwithwallet**.
    - It is broadcast to the regtest network (logged as **txidAB**).
    - Another block is mined to confirm the transaction.
- **Transaction from B′ to C′**
    - The script checks for a new UTXO belonging to **B′**, created by the **A′ → B′** transaction.
    - It constructs a raw transaction to send **0.3 BTC** from **B′** to **C′**, returning leftover to **B′**.
    - The script decodes the raw transaction (logged as “Decoded B'->C'”), signs, and broadcasts it, storing the transaction ID in **txidBC**.
    - The newly created transaction uses the **B′** output from **txidAB** as input, so that’s the chain from **A′ → B′** into **B′ → C′**.
    - Finally, it mines another block to confirm this second transaction.

---

2. Decoded Scripts for Both Transactions

- **A′ → B′ Transaction**
    - The script obtains a raw transaction using **createrawtransaction**, then logs **decAB = rpc_conn.decoderawtransaction(rawAB)** as “Decoded A'->B'.”
    - You can see each **vout** describing **scriptPubKey** for **B′** (P2SH-SegWit). The `scriptPubKey` typically appears as:
      OP_HASH160 <scriptHash> OP_EQUAL
    - Because this is P2SH-wrapped SegWit, the spending script is revealed in the redeemScript/witness data upon spending.
- **B′ → C′ Transaction**
    - Similarly decoded as “Decoded B'->C'.”
    - The output for **C′** also has a P2SH-wrapped SegWit `scriptPubKey`.
    - The input from **B′** is structured so that the real witness (signature + pubkey) is stored separately in `txinwitness`. The scriptSig is minimal or empty.

---

3. Challenge and Response Script Structure

- **Locking Script (Challenge)**
    - In **P2SH**-wrapped SegWit, the on-chain locking script is typically:
      OP_HASH160 <RedeemScriptHash> OP_EQUAL
    - The actual redeemScript for a typical single-sig SegWit output (P2WPKH) is:
      0 <Hash160(pubKey)>
    - This redeemScript is hashed, placed in the P2SH output. The challenge is “prove you have the correct witness script that

hashes to `<RedeemScriptHash>` and produce a valid witness."

- **Unlocking Script (Response)**
    - In **P2SH-P2WPKH**, the spender provides:
        1. A scriptSig that pushes the redeemScript, or is empty (depending on minimal relay rules).
        2. A **witness** field containing:
            - The **signature**
            - The **public key**
        3. Bitcoin checks that the redeemScript matches the hashed script in the P2SH output, and that the witness data is valid under the SegWit rules.
- **Validation**
    - Once the script is assembled, Bitcoin sees that the P2SH hash matches the redeemScript provided in scriptSig.
    - Then the SegWit script checks the signature (in the witness) against the public key.
    - If everything passes, the transaction is valid. This is how the **B′** address can be spent in the second transaction, referencing the previous **A′** → **B′** output.

---

4. Screenshots and Debugging Steps

- **Decoded Scripts**

```
PS C:\Users\Asus\Desktop\Blockchain> python script2.py
INFO:root:A'=2NA3uevaAwVfWkyoSn7jCKjetgtCfTf639Y, B'=2NG92nyMrZUhBUN72rXUem7u1TbmYuyHhVY, C'=2NFzY96zjiJY8uwgAHNhUmqfQvYPGNpYk3j
INFO:root:Funded A' with txid=559a645d64da6caa8ad18a571c387c779b3ccbf2dbeb7bf3fbd18a7739b53dd5
INFO:root:Decoded A'->B': {
  "txid": "76948a6b2b1f17995aab5f2419ce1de51fb99cfb0c1648db3a72c3aa48d4e2de",
  "hash": "76948a6b2b1f17995aab5f2419ce1de51fb99cfb0c1648db3a72c3aa48d4e2de",
  "version": 2,
  "size": 115,
  "vsize": 115,
  "weight": 460,
  "locktime": 0,
  "vin": [
    {
      "txid": "559a645d64da6caa8ad18a571c387c779b3ccbf2dbeb7bf3fbd18a7739b53dd5",
      "vout": 0,
      "scriptSig": {
        "asm": "",
        "hex": ""
      },
      "sequence": 4294967293
    }
  ],
  "vout": [
    {
      "value": "0.50000000",
      "n": 0,
      "scriptPubKey": {
        "asm": "OP_HASH160 fb1f0d57c23d705e855c8a6a2520d4b796a6e1ff OP_EQUAL",
        "desc": "addr(2NG92nyMrZUhBUN72rXUem7u1TbmYuyHhVY)#d6494ulp",
        "hex": "a914fb1f0d57c23d705e855c8a6a2520d4b796a6e1ff87",
        "address": "2NG92nyMrZUhBUN72rXUem7u1TbmYuyHhVY",
        "type": "scripthash"
      }
    },
    {
      "value": "0.49999000",
      "n": 1,
      "scriptPubKey": {
        "asm": "OP_HASH160 b856422b3c707f7066c5a1af175c56bb4d16320b OP_EQUAL",
        "desc": "addr(2NA3uevaAwVfWkyoSn7jCKjetgtCfTf639Y)#t6j2e857",
        "hex": "a914b856422b3c707f7066c5a1af175c56bb4d16320b87",
        "address": "2NA3uevaAwVfWkyoSn7jCKjetgtCfTf639Y",
        "type": "scripthash"
      }
```

```
    }
  ]
}
INFO:root:A'->B' broadcast, txid=6d5fd6cbfb1f2b918ba3648a20676174735c72f5edd015caf73f9e1324af474f
INFO:root:Decoded B'->C': {
  "txid": "5a865133349c63e5f9fc95f1ef3a17785ded0a899014c4f044af9c665d856190",
  "hash": "5a865133349c63e5f9fc95f1ef3a17785ded0a899014c4f044af9c665d856190",
  "version": 2,
  "size": 115,
  "vsize": 115,
  "weight": 460,
  "locktime": 0,
  "vin": [
    {
      "txid": "6d5fd6cbfb1f2b918ba3648a20676174735c72f5edd015caf73f9e1324af474f",
      "vout": 0,
      "scriptSig": {
        "asm": "",
        "hex": ""
      },
      "sequence": 4294967293
    }
  ],
  "vout": [
    {
      "value": "0.30000000",
      "n": 0,
      "scriptPubKey": {
        "asm": "OP_HASH160 f983ce624e050ec71b558b56c4662bb266b4abc7 OP_EQUAL",
        "desc": "addr(2NFzY96zjiJY8uwgAHNhUmqfQvYPGNpYk3j)#rh3qs6gt",
        "hex": "a914f983ce624e050ec71b558b56c4662bb266b4abc787",
        "address": "2NFzY96zjiJY8uwgAHNhUmqfQvYPGNpYk3j",
        "type": "scripthash"
      }
    },
    {
      "value": "0.19999000",
      "n": 1,
      "scriptPubKey": {
        "asm": "OP_HASH160 fb1f0d57c23d705e855c8a6a2520d4b796a6e1ff OP_EQUAL",
        "desc": "addr(2NG92nyMrZUhBUN72rXUem7u1TbmYuyHhVY)#d6494ulp",
        "hex": "a914fb1f0d57c23d705e855c8a6a2520d4b796a6e1ff87",
        "address": "2NG92nyMrZUhBUN72rXUem7u1TbmYuyHhVY",
        "type": "scripthash"
```

- **Execution Flow in the Debugger**

- For P2SH, it verifies `redeemScript` matches the hashed script in the P2SH output.
- Then the SegWit script portion checks the signature and public key in the witness.
- A successful final state (TRUE) indicates the input is valid, confirming that B′ or C′ can spend their respective funds.