SegWit Bitcoin Transaction Report

1. Transaction Workflow

Transaction A → B

- A transaction was broadcasted from **Address A** to **Address B** and confirmed on the blockchain.
- The **txid** for this transaction is stored in the variable **txid_A_to_B** in the script `segwitwallet1.py`.
- This transaction creates a **UTXO** for **Address B**, which is later used as input for the next transaction.

Transaction B → C

- A transaction from **Address B** to **Address C** was executed using the **UTXO** from the previous step.
- The **txid** for this transaction is stored in the variable **txid_B_to_C** in the script `segwitwallet2.py`.
- This transaction references the previous **UTXO** and transfers funds from **B** to **C**.

2. Decoded Scripts for Both Transactions

Transaction A → B

- **ScriptPubKey (Locking Script)**:

1. OP_0 <32-byte public key hash>

ScriptSig & Witness (Unlocking Script)

- The witness structure remains the same as the first transaction:
    1. **Signature**
    2. **Public Key**
- The witness field is used to provide the unlocking data to spend the UTXO from **Address B**.

3. Structure of Challenge and Response Scripts

Challenge Script (ScriptPubKey)

- This script is stored in the **UTXO** and determines the conditions to spend the output.
- In SegWit transactions, the challenge script is stored in **scriptPubKey**.
- **Example:**

2. OP_0 <20-byte public key hash>

   This ensures that only the owner of the corresponding private key can unlock and spend the coins.

Response Script (ScriptSig & Witness)

- In SegWit transactions, the **witness data** provides the required proof to spend the locked funds.
- The response script contains:
  - **Digital signature** proving ownership
  - **Public key** matching the public key hash in scriptPubKey

4. Validation Using Bitcoin Debugger

To verify the correctness of the challenge and response scripts, we use the **Bitcoin Debugger (btcdeb)**:

Steps to Validate the Scripts

1. **Extract the challenge script** from `scriptPubKey` of the previous transaction.
2. **Extract the response script** (witness data) from the input of the new transaction.
3. **Run the Bitcoin Debugger** to execute the challenge and response scripts together.
4. **Ensure that the execution completes successfully**, meaning the transaction is valid.
5.