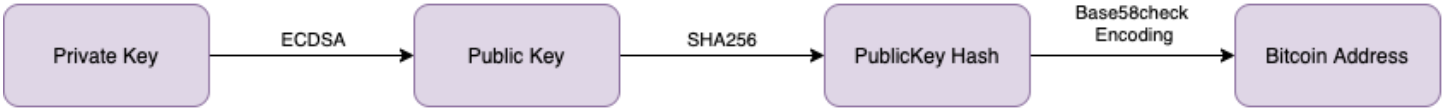
A [Technical] Guide to the BTC Transaction Process

1. **Receiver**: Generate BTC Address



   a. Generate a Private Key.

      Private Keys are any 256-bit number. You could come up with one on your own, download a program that randomly creates one for you, or purchase a hard wallet, such as Trezor or Ledger, to generate one based on some seed code.

   b. Use the Elliptic Curve Digital Signature Algorithm (ECDSA) to generate a corresponding Public Key for the above Private Key.

      Your Private Key uses the Elliptic Curve Digital Signature Algorithm (ECDSA) with the secp265k1 curve to create a corresponding Public Key. The algorithm uses mathematical operations to change the 256-bit private key number into a corresponding string of numbers called the Public Key. You can always use the Private Key to create the Public Key, but you cannot use the Public Key and work backwards to get the Private Key. Public and Private Keys are used to encrypt and decrypt data. You can lock/encrypt data with a Public Key such that the only way to decrypt or unlock the data is with the corresponding private key that created the public key.

   c. Generate a PublicKey Hash

      Your Public Key uses the SHA256 hash function to create a Public Key Hash, or Pay-To-Public-Key-Hash (P2PKH). The Public Key Hash isn't used to encrypt/decrypt data, rather to have a Digest to prove ownership of the Public Key.

   d. Generate a BTC Address

      Pubkey hashes are almost always sent encoded as base58-encoded strings (i.e. Bitcoin Address) containing an address version number, the hash, and an error-detection checksum to catch typos but doesn't need to be.

2. **Receiver**: Transfers the BTC Address to the sender.

3. **Sender**: Decodes the Bitcoin Address back to the standard hash (Public Key Hash or P2PKH)

4. **Sender**: Create a Transaction Output using the above P2PKH.

Sample:

```
{
        “version”: 1,
        “locktime”: 0,
        “vin”: [
                {
                        “txid” : “<TxID>”,
                        “vout”: 0,
                        “scriptSig”: “<scriptSig>”,
                        “sequence”:  4294967295
                }
        ],
        “vout”: [
                {
                        “value”: 0.01500000,
                        “scriptPubKey”: “OP_DUP OP_HASH160 <PubkeyHash> OP_EQUALVERIFY
OP_CHECKSIG”
                },
                {
                        “value”: 0.08450000,
                        “scriptPubKey”: “OP_DUP OP_HASH160 <PubkeyHash> OP_EQUALVERIFY
OP_CHECKSIG”
                }
        ]
}
```

Where,

- TxID (Transaction ID): reference to the Transaction containing the UTXO (Unspent Transaction Output) being spent.
- vout (Output Vector): Index of the UTXO within the Transaction referenced by the TxID
- scripSig: the witness that satisfies the locking script (scriptPubKey) in the UTXO
- sequence: deprecated. Previous versions of Bitcoin Core had a feature which prevented transaction signers from cancelling a time-locked transaction, but a necessary part of this feature was disabled to prevent denial of service attacks. A legacy of this system are four-byte sequence numbers in every input. Sequence numbers were meant to allow multiple signers to agree to update a transaction; when they finished updating the transaction, they could agree to set every input’s sequence number to the four-byte unsigned maximum (0xffffffff), allowing the transaction to be added to a block even if its time lock had not expired.
- locktime (nLockTime): earliest time a transaction can be added to the block chain

Note: In a P2PKH transaction, the signature script (scriptPubKey) contains an secp256k1 signature (sig) and full public key (pubkey):

```
<Sig> <PubKey> OP_DUP OP_HASH160 <PubkeyHash> OP_EQUALVERIFY OP_CHECKSIG
```

5. **Sender**: Broadcasts Transaction to the Blockchain Network.

6. **Network**: Categorizes the above Transaction as a UTXO and adds it to a pool of unconfirmed UTXOs.

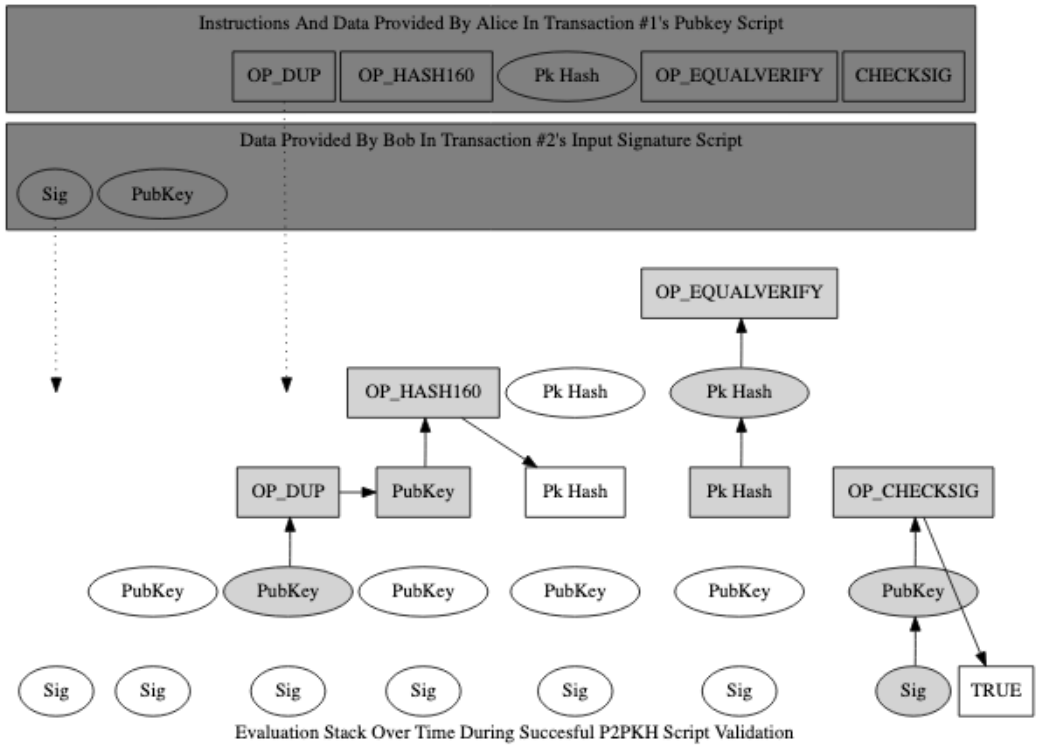
   The unconfirmed pool will continue adding unconfirmed UTXOs until miners are done validating the current block (takes ~10 min).

7. **Miner**: Picks up and validates Transactions from the Pool of Unconfirmed UTXOs (that have been building up while the last Block was being validated)

   The Miner validates the Transaction by:

   1. Making sure the sender has enough funds by adding all of the UTXOs to the Coinbase transaction.

   2. Making sure the scriptSig in the Sender's Input Vector (vin) satisfies Witness Script (scriptPubKey) conditions of the UTXO referenced by the Transaction ID (TxID) and Output Vector (vout).



(Alice initially sent Bob the BTC | Bob is now sending it to someone else…)

   a. The signature (from Bob's signature script) is added (pushed) to an empty stack. Because it's just data, nothing is done except adding it to the stack. The public key (also from the signature script) is pushed on top of the signature.

   b. From Alice's pubkey script, the "OP_DUP" operation is executed. "OP_DUP" pushes onto the stack a copy of the data currently at the top of it—in this case creating a copy of the public key Bob provided.

   c. The operation executed next, "OP_HASH160", pushes onto the stack a hash of the data currently on top of it—in this case, Bob's public key. This creates a hash of Bob's public key.

   d. Alice's pubkey script then pushes the pubkey hash that Bob gave her for the first transaction. At this point, there should be two copies of Bob's pubkey hash at the top of the stack.

   e. Now it gets interesting: Alice's pubkey script executes "OP_EQUALVERIFY".

   f. "OP_EQUALVERIFY" is equivalent to executing "OP_EQUAL" followed by "OP_VERIFY" (not shown).

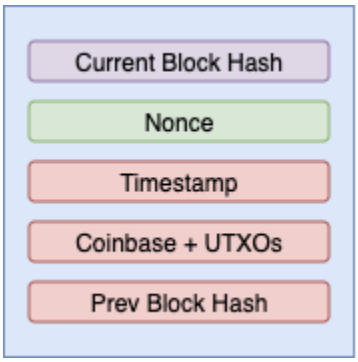
   g. "OP_EQUAL" (not shown) checks the two values at the top of the stack; in this case, it checks whether the pubkey hash generated from the full public key Bob provided equals the pubkey hash Alice provided when she created transaction #1. "OP_EQUAL" pops (removes from the top of the stack) the two values it compared, and replaces them with the result of that comparison: zero (false) or one (true).

h. “OP_VERIFY” (not shown) checks the value at the top of the stack. If the value is false it immediately terminates evaluation and the transaction validation fails. Otherwise it pops the true value off the stack.

i. Finally, Alice’s pubkey script executes “OP_CHECKSIG”, which checks the signature Bob provided against the now-authenticated public key he also provided. If the signature matches the public key and was generated using all of the data required to be signed, “OP_CHECKSIG” pushes the value true onto the top of the stack.

8. **Miner:** Creates a block with all the validated UTXOs from the previous step.

   Block:



9. **Miner:** Validates the Block by solving the Nonce such that the Current Block Hash (hash of the entire block) has a defined number of leading 0s.

10. **Miner:** Broadcasts the Block to the Network so that it can be added to the Blockchain.