Bitcoin Transaction Preliminaries: UTXOs

- Look at a transaction
- What is a UTXO?
  - Unspent Transaction Output
  - Every block height has a different UTXO set
- What does a UTXO consist of?
  - Amount
  - How it can be spent again
- What are transaction inputs?
  - Identified UTXOs
  - With proofs that they may be spent
- What are transaction outputs?
  - Amounts & spending conditions
- => A transaction destroys and creates UTXOs
- Every transaction?

Bitcoin Transaction Overview

Serialized TX (Over-the-wire Format)

❶ [version] 4 Bytes, LE
02000000

[number of inputs] 1 Byte, LE
01
[#0: previous txid] 32 Bytes, LE
e9c55a9a1ff2d62663f24157a85d204a0ee6008f4cdd913bc9
16e84fc1e605
[#0: previous output index] 4 Bytes, LE
00000000
❷ [#0: input script]
6b483045022100fa510547f5906c488d946a7e0cfe9d
e3e99e4151ffc8ba17edd5bc916dd5667502205628c3
5a06efde5ed6ed876cef671ba7a5392a931d48bfdf64
18525ea0b382760121029d4f145f18762a2397ceac36
1033b63904489f90e2a0b69882c9c2d0017bdd0a
[#0: input sequence] 4 Bytes
ffffffff

[number of outputs] 1 Byte
01
[#0: amount] 8 Bytes, LE
❸ 6003b80700000000
[#0: output script]
1976a91442b9b7745ec8b14788f0ca7ac28150782351
d44788ac

❹ [locktime] 4 Bytes, LE
00000000

Basic Bitcoin Transaction Structure:

- S=[version] [inputs] [outputs] [locktime]
- TXID=sha256(sha256((S)) (little-endian)

1) Version Field

- Mostly unused, any value can be applied (historically, default=1)
- For relative timelocks, must be set to >= 2

2) Inputs

- Each input uniquely references an unspent output
- Each valid input script unlocks the referenced output

3) Outputs

- Output amount describes the amount spendable by output
- Output script defines conditions for spending
- Transaction fee is the difference between output and referenced input amounts. Fee must be positive amount.

4) Locktime

- Absolute time from which on the transaction can be broadcast
- See absolute transaction timelocks

Transaction **Verification**

```
Input Script
Length of Script
6b (117 Bytes)
Push 72 Bytes to Stack
48 (72 in decimal)
TX Endorsement (72Bytes)
3045022100fa510547f5906c488d946a7e0cfe9de3e99e41..
.df6418525ea0b3827601
Push 33 Bytes to Stack
21 (33 in decimal)
Compressed Public Key (33Bytes)
029d4f145f18762a2397ceac361033b63904489f90e2a0b6
9882c9c2d0017bdd0a

Output Script
Length of Script
19 (25 in decimal)
OP_DUP
76
OP_HASH160
a9
Push 20 Bytes to Stack
14 (20 in decimal)
Public Key Hash (20 Bytes)
42b9b7745ec8b14788f0ca7ac28150782351d447
OP_EQUALVERIFY
88
OP_CHECKSIG
ac
```

In general, spending a Bitcoin output means providing valid unlocking arguments in the input script of the spending transaction.

Bitcoin Script Code

- Stack-based scripting operations.
- Input & Output scripts are both evaluated sequentially by the Bitcoin Script Machine.

Output Script (Locking Script)

- Generally, output script can describe anything.
- Usually, the output script checks for valid endorsement/signatures in the input scripts.

Input Script (Unlocking Script)

- Usually provides endorsement/signature of spending of output referenced in input.

Script

- Simple data structure: Stack. No loops: DoS protection
  - Push data to the stack: `0x47` `[71-byte-pubkey]`
  - Manipulate stack: `OP_DROP`, `OP_DUP`
  - Control flow: `OP_IF`
  - Do arithmetic: `OP_ADD`
  - Crypto: `OP_HASH160`, `OP_CHECKSIG`
  - Check timelocks: `OP_CHECKLOCKTIMEVERIFY`, `OP_CHECKSEQUENCEVERIFY`
- Script verification
  - Run input script ("scriptSig")
  - Copy stack
  - Run output script ("scriptPubKey")
  - Success == non-zero top-item

Some Script examples

- Satisfy the output script `3 OP_ADD 5 OP_EQUAL`
- Control flow: Satisfy the output script

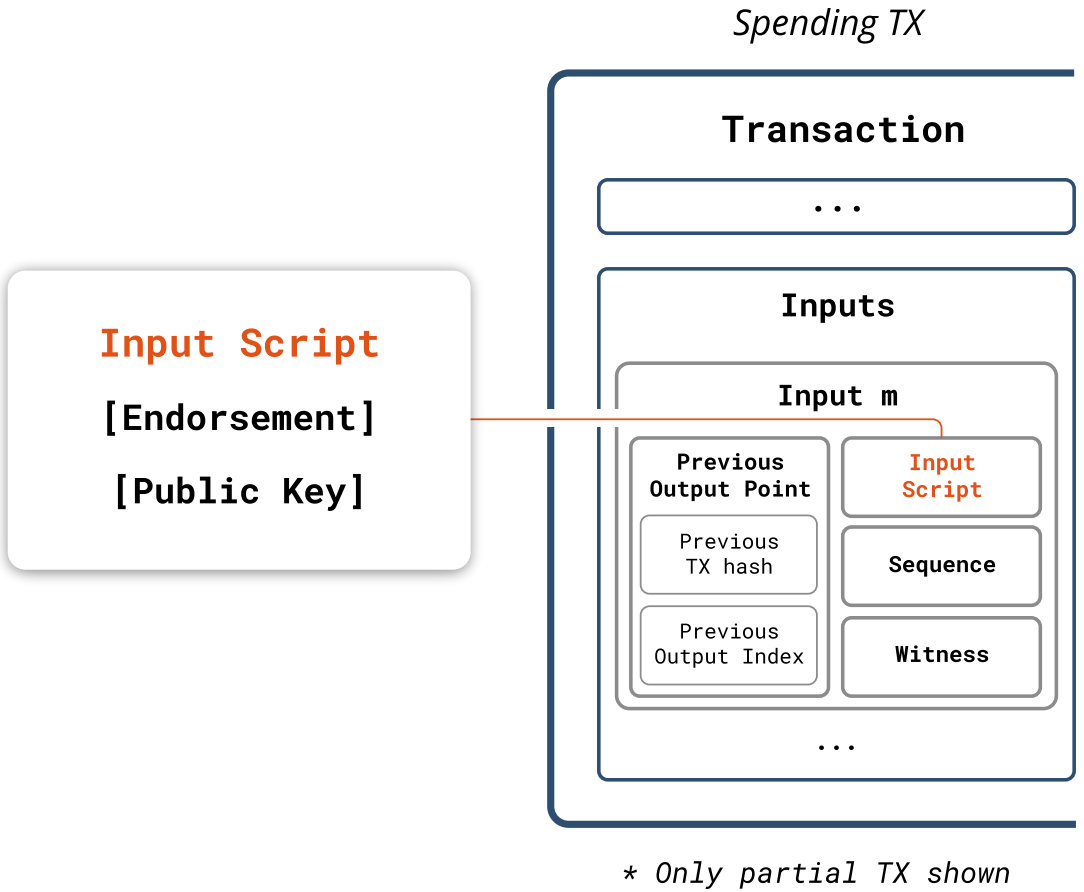
  ```
  OP_IF 1 2 OP_EQUALVERIFY OP_ELSE 1 OP_EQUALVERIFY OP_ENDIF
  ```

- What does this script do?

  ```
  OP_2DUP OP_EQUAL OP_NOT OP_VERIFY OP_SHA1 OP_SWAP OP_SHA1 OP_EQUAL
  ```

  - SHA-1 collision prize was claimed in 2017
  - Others still open

P2PKH Output Scripts



A basic wallet will send to a P2PKH(destination public key hash) output when it sends funds to a regular Bitcoin address.

Output Script (Locking Script)

- Input script must provide valid public key preimage.
- Input script must provide valid endorsement.

Next: P2PKH Script Verification

- Bitcoin script machine must run both input & previous output scripts and verify outcome.
- Top script machine stack element must be non-zero to be valid.
- If script runs are successful for all inputs, and the referenced output is unspent, the transaction is valid and can be broadcast.

P2PKH Script Evaluation

Input Script

[Endorsement]

[Public Key]

Output Script

OP_DUP

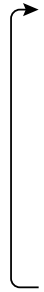
OP_HASH160

[Public Key Hash']

OP_EQUALVERIFY

OP_CHECKSIG

Script Machine Stack



1

Output script run must end with a non-zero element on stack, in order for a valid spending of output script.

In our example:
The input script successfully unlocks/spends the output script.

[Data] push operator

- Represents data bytes to be pushed onto stack

OP_DUP

- Duplicate top member of stack

OP_HASH160

- Duplicate top member of stack

OP_EQUALVERIFY

- Verifies if top 2 stack elements are equal.
  - If positive, nothing is output to stack and script run continues.
  - If negative, script run fails.

OP_CHECKSIG

- Checks whether signature & public key are valid, returns 1 or 0 to stack.

Endorsement Check Operations

OP_CHECKSIG

OP_CHECKSIGVERIFY

OP_CHECKMULTISIG

OP_CHECKMULTISIGVERIFY

Script Machine Stack
Before OP_CHECKMULTSIGVERIFY

[3]

[Public Key 3]

[Public Key 2]

[Public Key 1]

[2]

[TX Endorsement 2]

[TX Endorsement 1]

[0]

Script Machine Stack
After OP_CHECKMULTSIGVERIFY

Nothing or Fail

Checksig opcodes validate that the transaction is signed by the private key corresponding to the public key on the stack.

CheckSig/CheckVerify Operations

- Usually part of an ouput script.
- Checks endorsement(s) against public key(s).
- Checks that endorsement signs correct transaction data.

Next: Transaction Endorsements

- An endorsement signs off on all or parts of a transaction.
- An endorsement is specific to each transaction input.
- Each input spends its referenced output individually, with a separate endorsement.

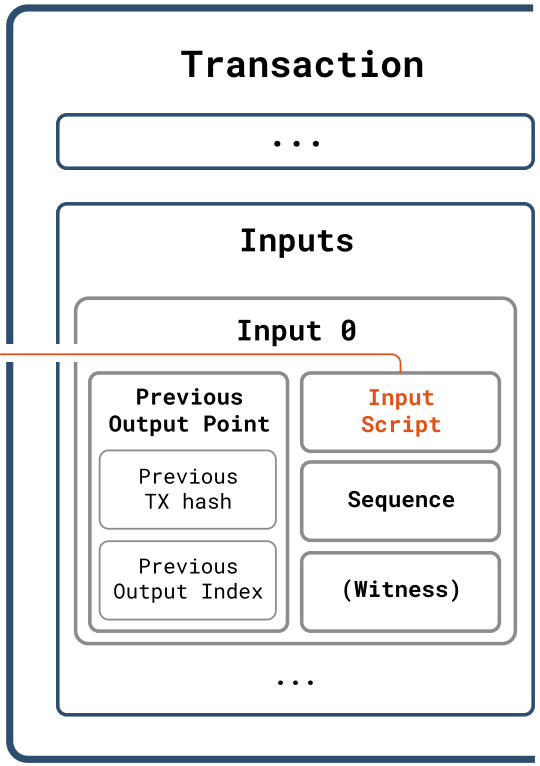
Building an **Endorsement**

❸

Input Script

[TX Endorsement]

[Public Key]



** Only partial TX shown*

1) Build TX w/o input script

- The transaction is populated with all elements which are signed by the endorsement.
- For Sighash ALL, this includes all the TX elements except for the input script, which is left empty.
- Except at the input that is being signed right now, where the previous output script is placed.
  - Why? Not even Pieter Wuille knows

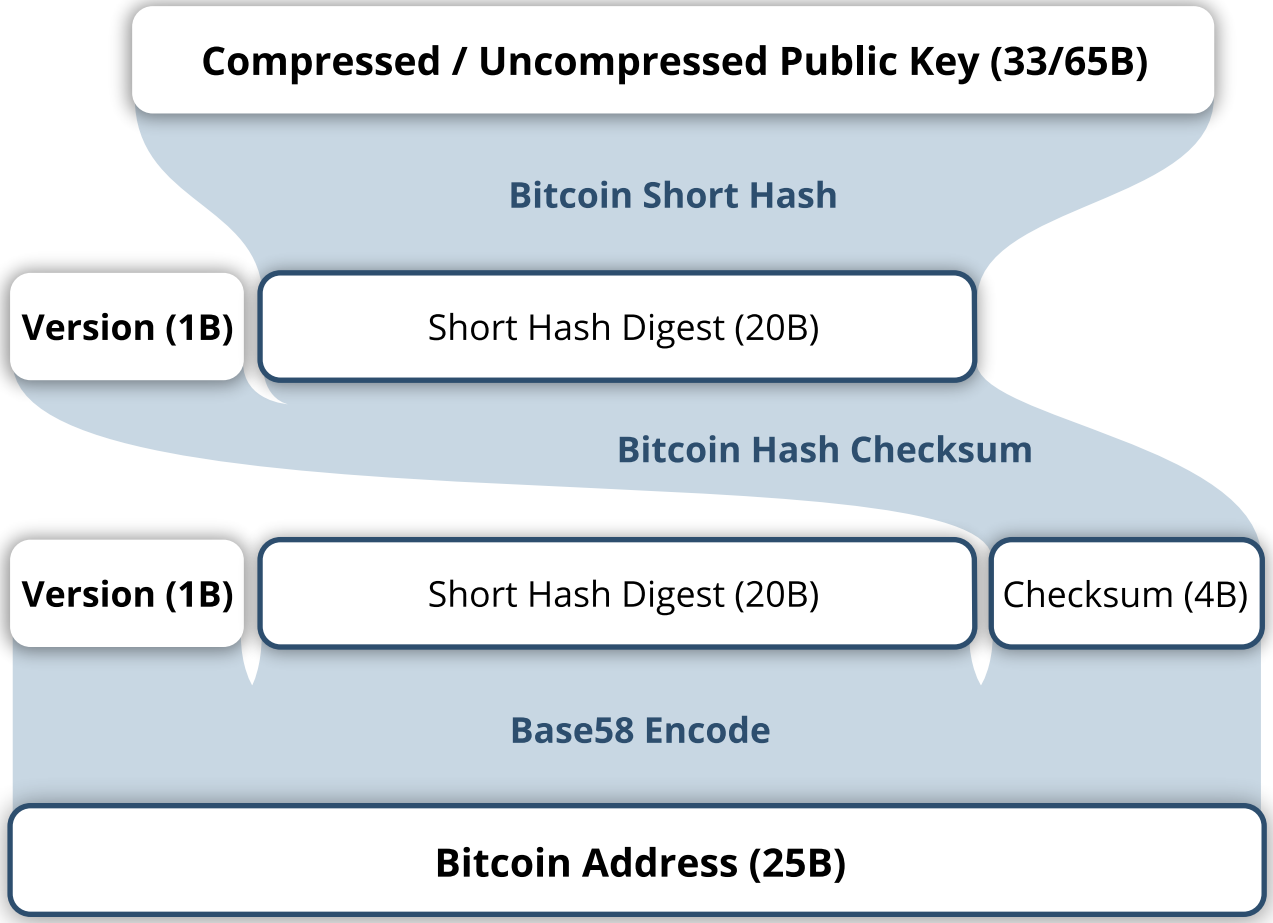
2) Sign Signature Hash

- The serialized transaction is appended with the sighash marker, and hashed.
- The endorsement is a DER encoded signature thereof.

3) Complete Input Script & Broadcast

- With a endorsement constructed, the transaction can now be completed with the valid input script and broadcast on the network.

Public Key to **P2PKH Address**



Bitcoin Short Hash

- `RIPEMD160(SHA256(public key))`
- Compressed or uncompressed public key point.

Bitcoin Hash Checksum

- `double SHA256(input)`
- First 4 bytes of digest.
- Version prefix (Mainnet/Testnet)
  - 0x00/0x6F

Base58 Encoding

- Bitcoin Address:
  - Mainnet: begins with 1
  - Testnet: begins with m or n