DSRV©

Bitcoin: The Theoretical Minimum

in-house seminar

Jeongho Jeon

2025-05-XX

DSRV (All That Node, Custody, Payments, Validator, WELLDONE Studio)

version 20251029

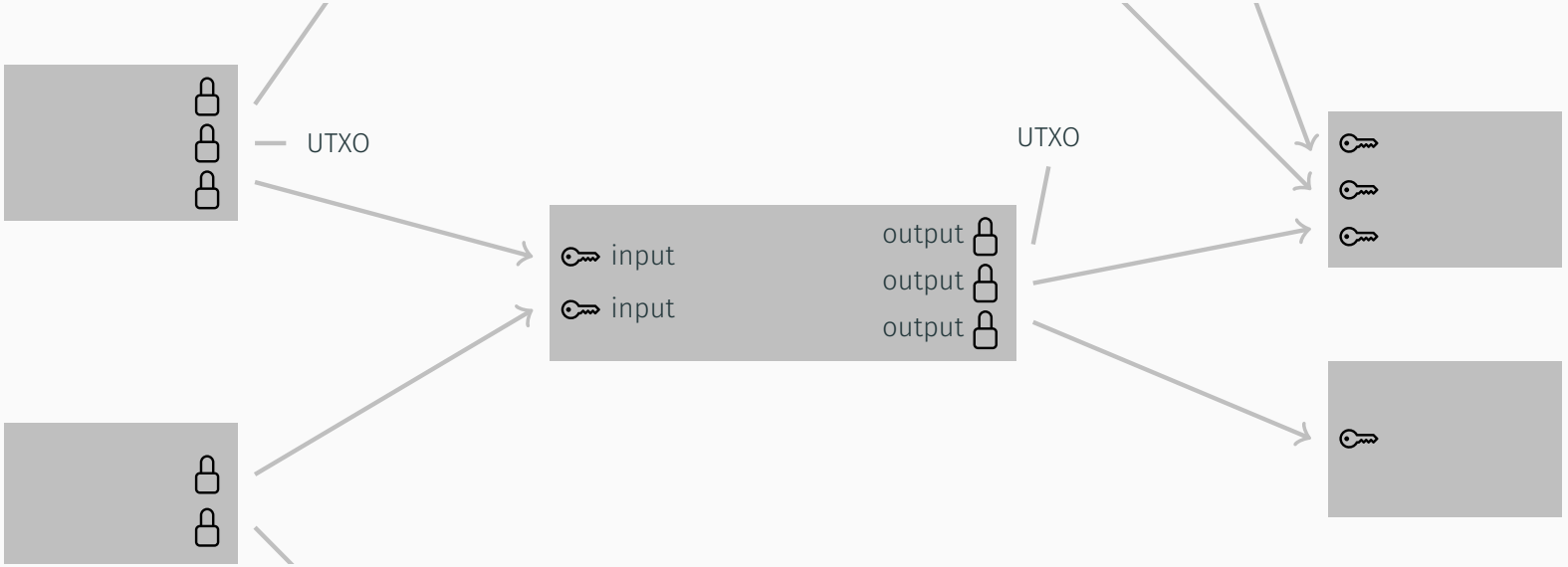
UTXO model

Bitcoin and its forks use the UTXO model, while account-based models, like a bank balance, are easier to reason about. Account-based systems record balances in one place, making them better suited for smart contracts, as seen in Ethereum.

In the UTXO model, a user's coins are scattered across many outputs. Wallets sum them to show the total balance. Unlike the account-based model, UTXO allows parallel transfers, e.g., transfering coins from A, B and C to D, E and F in one transaction. Though harder to handle in smart contract engines, UTXO is well-suited for privacy coins.

A blockchain *state* stores either UTXOs or account balances, depending on the model.

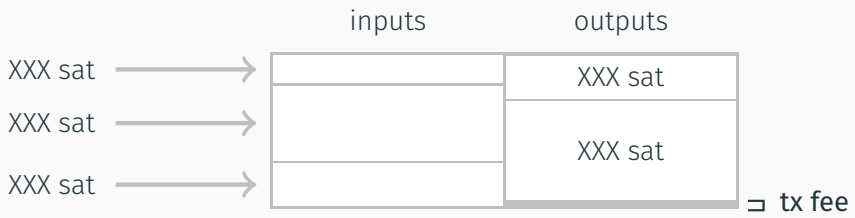
Bitcoin transaction



A transaction consists of inputs and outputs. Think of it as melting old coins (inputs) and minting new ones (outputs). Each input references an **U**nspent **T**rans(**X**)action **O**utput (UTXO) using the previous transaction hash and output index.

Outputs were locked with conditions, like a recipient address and related signature. Inputs provide the unlocking key, typically a digital signature. New outputs are created with new locks, defining who can spend the coins next.

no explicit fee



Bitcoin transactions don't explicitly show the fee. The fee is calculated as total input amount minus total output amount. Miners collect the fee as a reward. While outputs show coin amounts, input values must be traced back to previous transactions.

Bitcoin fees are measured in sat/vB (satoshis per virtual byte, see SegWit for 'virtual'). Miners collect transactions into 1 MB blocks and prefer txs with higher fees per byte. Fees are not based on the coin amount, but on transaction size. Transaction size increases with more inputs and outputs.

Coins that cost more to spend than they're worth are called *dust*. Wallet developers avoid selecting inputs that would leave dust. SDKs don't handle this automatically.

change

It's rare for inputs to exactly match outputs plus fees. The leftover amount output is sent back to the sender as *change*.

Bitcoin recommends changing addresses with each use. Unlike MetaMask, which shows one address at a time, Bitcoin wallets often display 20+ addresses. These addresses are derived from the same mnemonic, but appear unrelated to outsiders.

transaction fee bumping

Two common strategies exist to handle transactions waiting in the mempool, often due to low fees.

Replace By Fee (RBF) A transaction with an input's "sequence" field below `0xfffffffe` is replaceable. To replace a transaction, create a new one that spends at least one of the same UTXOs with a higher fee. Fees can be increased by including more inputs or lowering output values.

Child Pays for Parent (CPFP) Without replacing the original transaction, you can attach a high-fee child transaction that spends its output. To claim the higher fee, miners are incentivized to include the low-fee parent transaction along with the high-fee child one in the same block. Even recipients can use this method to accelerate confirmation.

coinbase transaction

Bitcoin transactions can have any number of inputs and outputs. For example, a mining pool reward distribution transaction may have 1 input and over 100 outputs.

Some special transactions create new bitcoins without spending any inputs. These are *coinbase transactions*, which reward miners. Each block starts with a coinbase transaction. It has one dummy input that isn't interpreted. It is typically used for extranonce and mining pool names. Satoshi's famous message, *The Times 03/Jan/2009 ... bailout for banks.*, was here.

The coinbase transaction's output amount equals the block reward plus the total fees from all transactions in the block. SegWit introduced an additional zero-valued output carrying the wtxid (txid with witness) Merkle tree root.

Bitcoin script

To spend a UTXO, 🔑 (scriptSig) + 🔒 (scriptPubKey) are executed in order. If the result is true, the spend is allowed. Let's take a legacy P2PKH (Pay to Public Key Hash) transaction as an example.

🔑 (new tx's input) - <sig> <pubKey>
🔒 (old tx's output) - `OP_DUP` `OP_HASH160` <pubKeyHash> `OP_EQUALVERIFY` `OP_CHECKSIG`

As commands are executed in order, the stack looks like this:

command	stack
<sig>	<sig>
<pubKey>	<sig> <pubKey>
`OP_DUP` (duplicate the last)	<sig> <pubKey> <pubKey>
`OP_HASH160` (hash the last)	<sig> <pubKey> <pubKeyHash>
<pubKeyHash>	<sig> <pubKey> <pubKeyHash> <pubKeyHash>
`OP_EQUALVERIFY` (fail if not equal)	<sig> <pubKey>
`OP_CHECKSIG` (check the last's signature)	true

With an invalid signature, the script fails and the UTXO cannot be spent.

standard transaction types

Most nodes only relay and mine standard transaction types. Transactions that create dust may be rejected or not mined. In addition to P2PKH, standard Bitcoin transaction types include:

P2PH (Pay to Public Key) This output exposed the public key and was used briefly in the early days.

🔑 (new tx's input) - <sig>
🔓 (old tx's output) - <pubKey> `OP_CHECKSIG`

`OP_RETURN` `OP_RETURN` creates an unspendable output. It can store arbitrary data on Bitcoin, which Bitcoin does not interpret. This enabled overlay protocols like Colored Coins and Mastercoin. Some believe these were early forms of NFT. Since it's unspendable, it has no amount.

🔑 (new tx's input) - unspendable
🔓 (old tx's output) - `OP_RETURN` <arbitrary data>

standard transaction types

P2SH (Pay to Script Hash) With `OP_IF` and `OP_ELSE`, complex conditions can be written, for example, requiring 2-of-3 guardian signatures until the heir becomes an adult. But this increases output size and UTXO set. To address this, P2SH was introduced: only a script hash is stored in the output, and the full script is revealed when spending.

🔑 (new tx's input) - `OP_HASH160` <scriptHash> `OP_EQUAL`
🔒 (old tx's output) - <$param_1$> ...<$param_N$> <script>

The script may use `OP_CHECKMULTISIG` to require multiple signatures for spending. Taproot replaces this with more efficient `OP_CHECKSIGADD`.

🔑 (new tx's input) - 0(dummy) <sig_1> <sig_2>
🔒 (old tx's output) - 2 <$pubKey_1$> <$pubKey_2$> <$pubKey_3$> 3 `OP_CHECKMULTISIG`

🔑 (new tx's input) - <sig_3> dummy <sig_1>
🔒 (old tx's output) - <$pubKey_1$> `OP_CHEKCSIG` <$pubKey_2$> `OP_CHEKCSIGADD` <$pubKey_3$> `OP_CHEKCSIGADD` 2 `OP_GREATERTHANOREQUAL`

SegWit (Segregated Witness)

SegWit soft fork was introduced in 2017 after the blocksize war. It moved bulky signatures to a new “witness” section, counted at ¼ bytes. This allowed more transactions per 1MB block, reduced fees, and led to the concept of vbytes: 1 vbyte = 4 weight.

If a node assumes transactions were already validated by others, it doesn’t need the witness to understand its effects. This allows for lighter relaying and storage.

Also, since random signatures no longer affect the txid (tx hash), the malleability issue is resolved, enabling dependency-sensitive L2 solutions like Lightning Network.

Finally, SegWit made it easier to introduce new features through version upgrades.

SegWit (Segregated Witness)

SegWit introduced P2WPKH and P2WSH—the SegWit versions of P2PKH and P2SH. A P2WPKH script looks like this:

🔑 (new tx's input) - empty (<sig> <pubKey> in witness)
🔒 (old tx's output) - 0(version) <pubKeyHash>

SegWit uses lowercase Bech32 addresses (same as Cosmos) that start with 'bc1', visually distinct from legacy mixed-case Base58Check addresses that start with '1' or '3'.

Taproot

Taproot soft fork (2021) introduced support for easily aggregatable Schnorr signatures. It uses SegWit version 1 and Bech32m addresses. Spend conditions are organized in a Merkle tree, revealing only the script used—improving block space and privacy. P2TR (Pay to Taproot) outputs can be spent in one of two ways.

🔒 (old tx's output) - 1(version) <taprootAddress>

key path spending with one aggregated signature (implying consent from all parties)

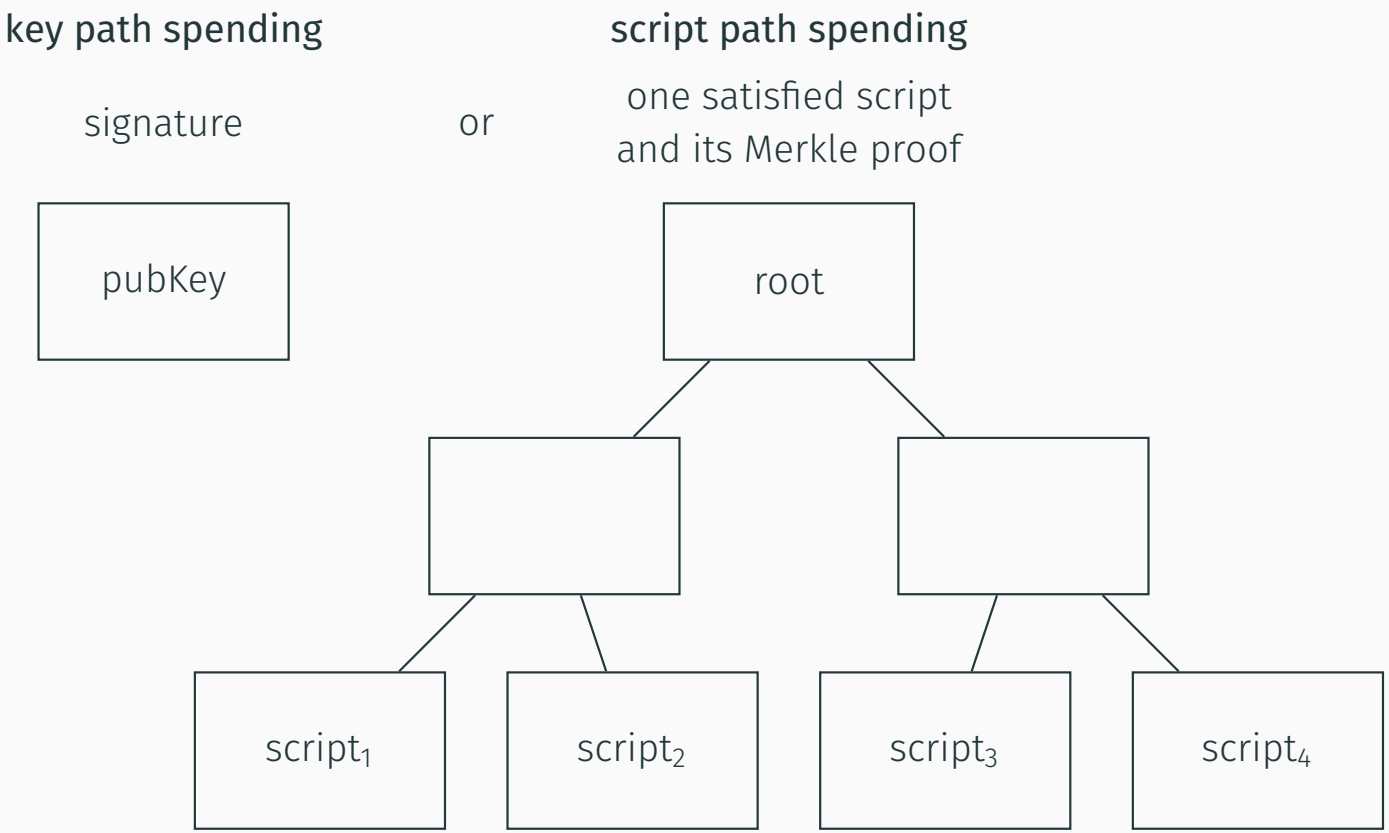
🔑 (new tx's input) - empty (<sig> in witness)

script path spending with the satisfying script and its Merkle proof. Check if a Taproot address Q can be created from The Merkle tree root m calculated from the Merkle proof.

$$Q = P + hash(P||m) \cdot G$$

🔑 (new tx's input) - empty (<param$_1$> ...<script> <controlBlock> in witness)
<controlBlock> contains internal public key P and Merkle proof.

Taproot


key path spending
signature
or
script path spending
one satisfied script
and its Merkle proof
pubKey
root
script1
script2
script3
script4

Babylon

Bitcoin uses PoW and has no native staking. However, ideas emerged to earn yield on idle BTC—Babylon is one example. Unlike typical protocols that lock tokens like bridges, Babylon promotes self-custodial staking and on-demand unbonding.

Most staking operations, including rewards, happen on the Babylon chain (Cosmos SDK + Cosmwasm engine). There are daemon programs monitor Babylon Bitcoin staking transactions.

Babylon uses Taproot transactions on the Bitcoin chain. Since a public key that no one knows the private key for is used, key path spending is not possible, and only script path spending is allowed.

Babylon

The script requires signatures from the staker (user), finality provider (validator), and covenant committee (which is selected by governance and requires a quorum of 6 out of 9 members). When creating a staking transaction, the staker must pre-sign unbonding transaction and slashing transaction in advance.

Three scripts are used in Taproot transactions:

Timelock path after the staking period ends

<stakerPubKey> OP_CHECKSIGVERIFY <timelockInBlocks> OP_CSV

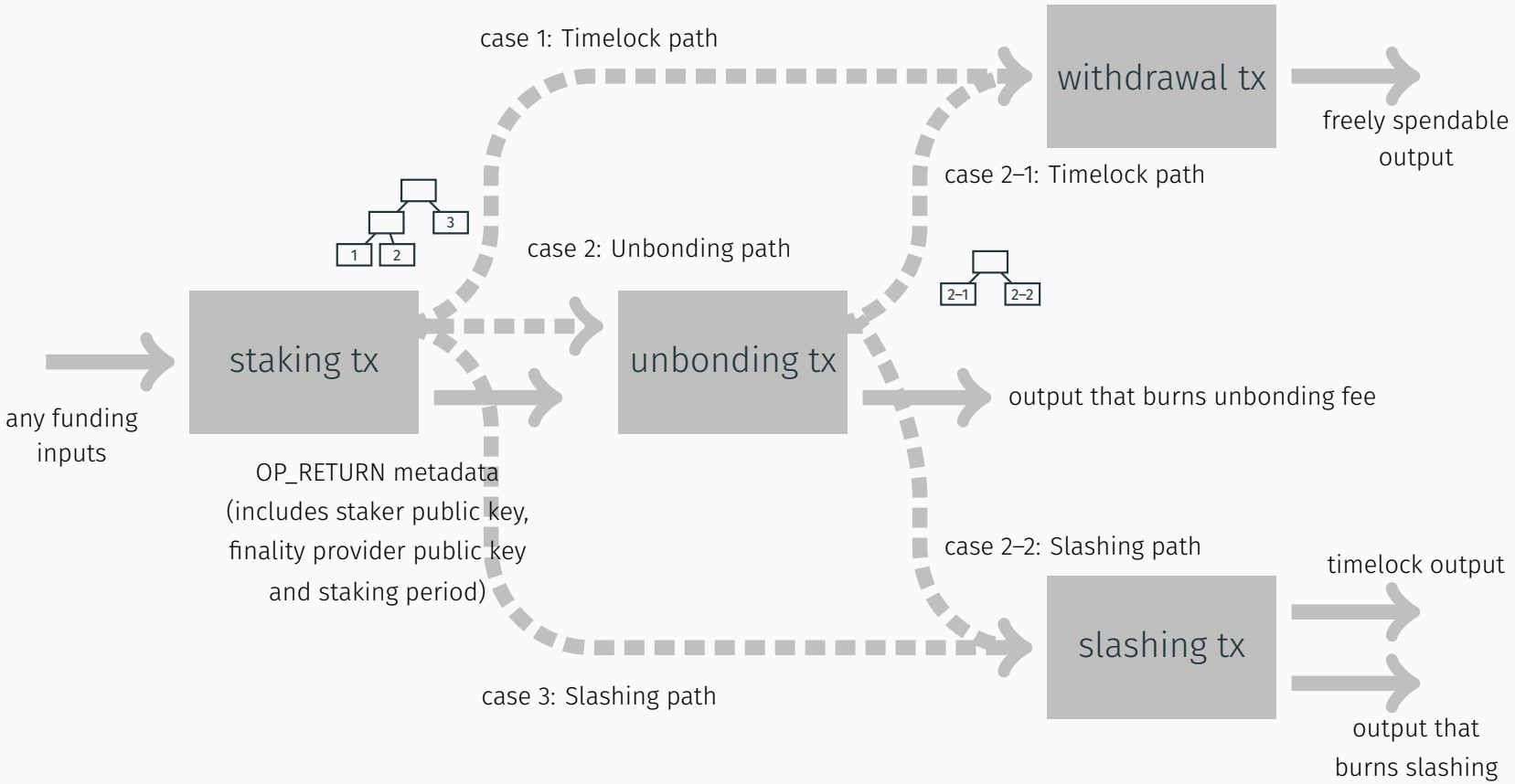
Unbonding path withdrawal during the staking period

<stakerPubKey> OP_CHECKSIGVERIFY
<covenantPubKey$_1$> OP_CHECKSIG ... OP_CHECKSIGADD <n> OP_NUMEQUAL

Slashing path upon a slashing event. When double signing occurs, the finality provider private key is revealed, allowing anyone to generate an finality provider signature.

<stakerPubKey> OP_CHECKSIGVERIFY
<finalityProviderPubKey> OP_CHECKSIGVERIFY
<covenantPubKey$_1$> OP_CHECKSIG ... OP_CHECKSIGADD <n> OP_NUMEQUAL

Babylon Bitcoin transactions



topics not covered

- atomic swap and Lightning Network
- soft fork activation
- Inscriptions NFT
- selfish mining
- script hash types
- ...