

Bitcoin Protocol Deep Dive

BSc Blockchain, Crypto Economy & NFTs

Course Instructor

Module A: Blockchain Foundations

By the end of this lesson, you will be able to:

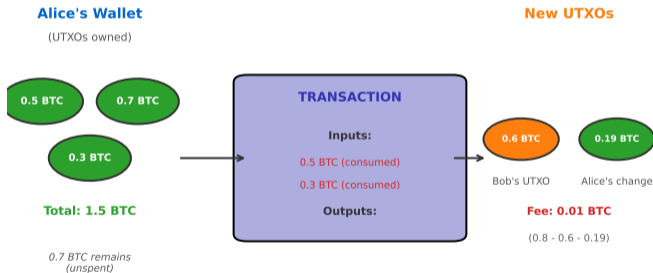
- Explain the UTXO (Unspent Transaction Output) model
- Describe the structure of a Bitcoin transaction
- Understand transaction inputs and outputs
- Recognize different Bitcoin Script types
- Trace the lifecycle of a transaction from creation to confirmation
- Distinguish between legacy and SegWit transaction formats

The UTXO Model: Core Concept

What is a UTXO?

- Unspent Transaction Output = a chunk of bitcoin that can be spent
- Bitcoin does not track account balances (unlike Ethereum)
- Instead, tracks individual “coins” (UTXOs)

UTXO Model: Unspent Transaction Outputs



UTXOs are consumed entirely, change returned as new UTXO

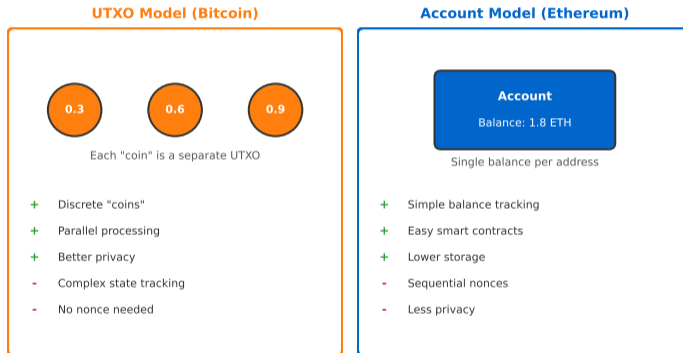
Analogy: Physical Cash

- UTXOs are like bills in your wallet
- You do not have “100 EUR balance” – you have five 20 EUR bills
- To pay 30 EUR: give one 20 EUR bill + one 10 EUR bill
- To pay 25 EUR with a 50 EUR bill: receive 25 EUR change

Key Principles:

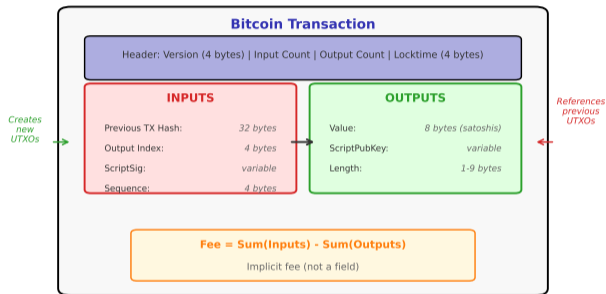
- Each UTXO can only be spent once (consumed entirely)
- Spending a UTXO creates new UTXOs
- Blockchain tracks which UTXOs are unspent
- Your wallet balance = sum of all UTXOs you can spend

State Models: UTXO vs Account-Based



Why Bitcoin Uses UTXO: Easier to verify (check UTXO existence), no global state needed, natural double-spend prevention.

Bitcoin Transaction Anatomy



Transaction Hash (txid):

- Double SHA-256 of entire transaction – unique identifier
- Used to reference transaction in inputs

Each Input Contains:

- 1 **Previous Transaction Hash:** txid of transaction containing UTXO
- 2 **Output Index:** which output from previous transaction (0, 1, 2, ...)
- 3 **ScriptSig (Unlocking Script):** provides signature and public key
- 4 **Sequence Number:** originally for transaction replacement

Example Input:

- Previous tx: 5a3c7b... (Alice received 3 BTC)
- Output index: 0 (first output of that transaction)
- ScriptSig: signature proving Alice owns the UTXO

Multiple Inputs:

- Transaction can have many inputs (combining UTXOs)
- Each input must be signed separately

Each Output Contains:

- 1 **Value:** Amount of satoshis (1 BTC = 100,000,000 satoshis)
- 2 **ScriptPubKey (Locking Script):** conditions to spend this output

Change Outputs:

- When input value i payment amount, create change output
- Change goes back to sender (usually new address for privacy)
- Example: Spend 5 BTC UTXO to send 3 BTC \rightarrow 3 BTC to recipient + 1.999 BTC change

Transaction Fee:

- Fee = Sum of inputs – Sum of outputs
- Not explicitly stated in transaction (implicit)
- Miner collects the difference

What is Bitcoin Script?

- Simple, stack-based programming language
- Not Turing-complete (no loops, limited expressiveness)
- Executed during transaction validation
- Determines whether transaction is valid

How It Works:

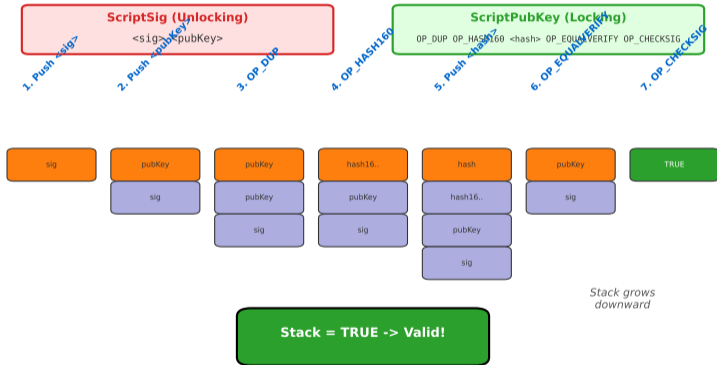
- 1 Combine ScriptSig (from input) + ScriptPubKey (from previous output)
- 2 Execute script operations left to right
- 3 Use a stack (LIFO data structure)
- 4 Transaction valid if final stack value is TRUE

Basic Operations:

- OP_DUP: duplicate top stack item
- OP_HASH160: hash with SHA-256 then RIPEMD-160
- OP_EQUALVERIFY: check equality, fail if not
- OP_CHECKSIG: verify signature against public key

Bitcoin Script Stack Execution

P2PKH: Pay-to-Public-Key-Hash



Why Public Key Hash? Shorter addresses (20 vs 33 bytes), extra security (quantum resistance until spending).

Purpose:

- Allows complex spending conditions (multi-signature, time-locks)
- Hides complexity until spending time
- Sender only needs recipient's P2SH address

ScriptPubKey:

OP_HASH160 <ScriptHash> OP_EQUAL

ScriptSig:

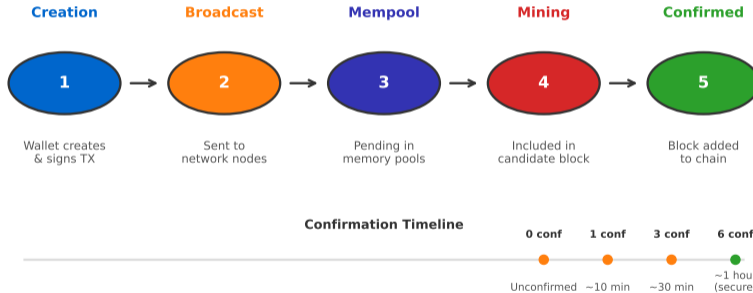
<Signature1> <Signature2> ... <RedeemScript>

Verification Process:

- 1 Hash the redeem script
- 2 Verify hash matches ScriptHash in ScriptPubKey
- 3 Execute redeem script with provided signatures
- 4 Transaction valid if redeem script evaluates to TRUE

Bitcoin Transaction Lifecycle

More confirmations = Higher security (6+ for large amounts)



Key Stages:

• Creation → Signing → Broadcast → Mempool → Mining → Confirmation

1. Creation and Signing:

- Wallet selects UTXOs, constructs inputs/outputs, calculates fee
- Signs each input with corresponding private key

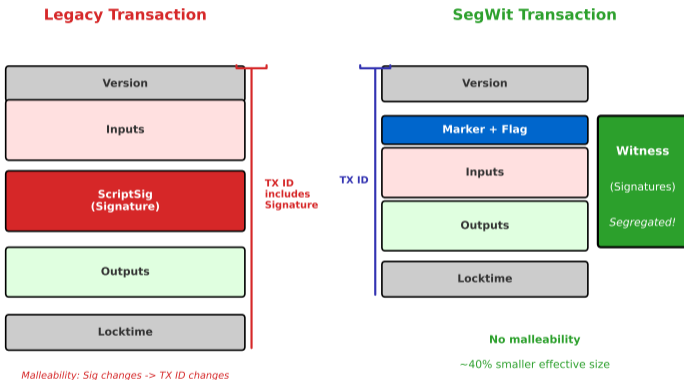
2. Broadcast and Mempool:

- Transaction sent to connected nodes, validated, added to mempool
- Nodes relay to peers (propagation across network)
- Transactions sorted by fee rate in mempool

3. Mining and Confirmation:

- Miner includes transaction in candidate block
- Block mined, broadcast, validated by network
- Each new block adds one confirmation
- 6 confirmations typically considered final (~ 1 hour)

Segregated Witness: Separating Signatures



Problem with Legacy: Signature in TX hash → malleability
SegWit Solution (BIP 141, 2017): Separate witness data from TX ID

Block Capacity Increase:

- Legacy: 1 MB block size limit
- SegWit: measured in “weight units” (max 4 million)
- Witness data: 1 byte = 1 weight unit; Non-witness: 1 byte = 4 weight units
- Effective capacity: ~2-2.7 MB per block

Lower Transaction Fees:

- Witness data discounted by 75%
- Same transaction costs less with SegWit

Address Formats:

- P2WPKH (native SegWit): starts with “bc1q” (Bech32 encoding)
- P2SH-wrapped SegWit: starts with “3” (backward compatible)

Enables Lightning Network: Fixes malleability for secure payment channels

Key Improvements:

- **Schnorr Signatures:** More efficient, enable signature aggregation
- **MAST:** Complex scripts hidden until execution, only reveal used branch
- **Privacy:** All transactions look similar on-chain

Benefits:

- Multi-sig indistinguishable from single-sig
- Complex smart contracts look like simple payments
- Smaller transaction size for complex scripts

Address Format:

- Starts with “bc1p” (Bech32m encoding)
- Example: bc1p5d7rjq7g6rdk2yhzks9smlaqtedr4dekq08ge8...

Syntax Validation:

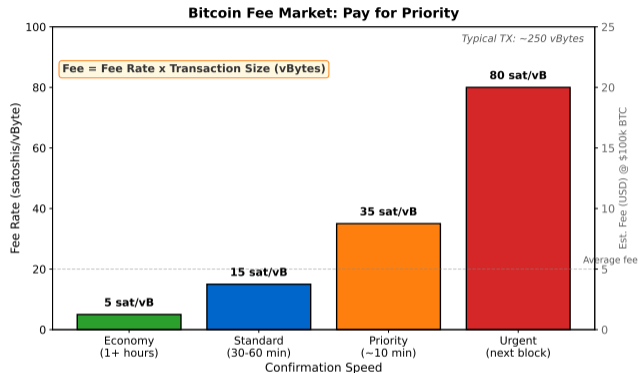
- Transaction size within limits
- Output values non-negative, do not exceed input values
- No duplicate inputs (double-spend within transaction)

Semantic Validation:

- All referenced UTXOs exist and are unspent
- Signatures valid for all inputs
- Script execution succeeds for all inputs
- Locktime constraints satisfied

Rejection Reasons:

- Invalid signature → likely fraud attempt
- Double-spend → UTXO already spent
- Dust output → output value too small (spam prevention)



Fee Market Dynamics:

- Block space is scarce (~4 MB weight per 10 minutes)
- Users compete for inclusion via fees
- Fee estimation based on mempool state and target confirmation time

RBF (BIP 125):

- Create replacement transaction with same inputs
- Increase fee by at least 1 satoshi per byte
- Signal RBF by setting sequence number `j 0xfffffffffe`
- Use cases: fee bump, output modification, cancel transaction

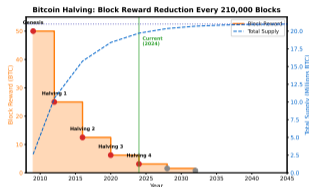
Child-Pays-for-Parent (CPFP):

- Recipient creates high-fee transaction spending unconfirmed output
- Miners must include parent to mine child
- Combined fee rate makes both transactions attractive

Comparison:

- RBF: sender bumps fee (requires signaling)
- CPFP: receiver bumps fee (works for any transaction)

Coinbase Transactions and Block Rewards



21M Cap

Coinbase Properties:

- First transaction in every block, creates new bitcoins
- Miner collects block reward + all transaction fees
- Must wait 100 confirmations before spending (maturity rule)

January 10, 2024: Spot Bitcoin ETF Approval

- SEC approved 11 spot Bitcoin ETFs (first time in US)
- Major issuers: BlackRock (IBIT), Fidelity (FBTC), Grayscale (GBTC)
- Accumulated \$50B+ in assets under management by end of 2024

Market Impact:

- Institutional legitimization of Bitcoin as asset class
- Daily trading volume rivals major commodity ETFs
- Custody handled by regulated institutions

Transaction Implications:

- ETF creation/redemption uses large on-chain transactions
- Institutional custody drives UTXO consolidation
- Increased demand for block space during high activity

- Bitcoin uses the UTXO model: transactions consume old outputs and create new ones
- Each transaction has inputs (UTXOs being spent) and outputs (new UTXOs)
- Bitcoin Script enables flexible spending conditions without Turing completeness
- P2PKH (legacy), P2SH (multi-sig), SegWit, and Taproot offer increasing efficiency
- Transaction lifecycle: creation → signing → broadcast → mempool → mining → confirmation
- Fees determined by market competition for block space
- SegWit and Taproot improve scalability and privacy

Design Philosophy: Bitcoin prioritizes security and decentralization over transaction throughput.

- ❶ Why does Bitcoin use the UTXO model instead of the account model?
- ❷ How does the fee market incentivize miners to include transactions?
- ❸ What are the trade-offs between legacy addresses, SegWit, and Taproot?
- ❹ How does transaction malleability affect second-layer solutions?
- ❺ Why is the coinbase maturity rule (100 confirmations) necessary?
- ❻ How could you design a transaction that can only be spent after a certain date?

Topics to be covered:

- Mining mechanics and the proof-of-work algorithm
- Nonce searching and difficulty adjustment
- Block header structure and hash rate
- 51% attacks and mining centralization risks
- Energy consumption and environmental concerns

Preparation:

- Review hash function properties (pre-image resistance)
- Explore Bitcoin mining pools and hash rate distribution
- Consider the economics of mining profitability