Bitcoin Light Clients (SPV) - A Lightweight Gateway into Bitcoin

- Bitcoin Light Clients let you interact with Bitcoin without running a full node. No full blocks stored. No complete transaction verification. Just compact Merkle proofs to confirm inclusion.
- Light clients only verify transaction inclusion in blocks. They don't verify the full transaction. They download 80-byte headers, not full blocks. They use Merkle proofs for inclusion checks.
- Why use light clients? • Cross-chain apps need Bitcoin verification • Full nodes are heavy, light clients are efficient Many chains/bridges use it. Powerful, but risky.
- How Bitcoin Light Clients work: • Request all block headers (genesis to tip) • Verify headers: PoW, prev hash link, consensus • Build Bloom filter for relevant transactions • Request txs + Merkle proofs from full nodes

---

What Is a Bloom Filter?

A **Bloom filter** is a space-efficient, probabilistic data structure used to test whether an element is part of a set—without storing the actual elements.

Instead of keeping full data, it uses:

- A **bit array** (initially all zeros)
- Multiple **hash functions**

How it works:

- To add an item: hash it with `k` functions → set the corresponding bits in the array to `1`.
- To check membership: hash the item again → if all bits are `1`, it *might* be in the set; if any bit is `0`, it's *definitely not*.

Key properties:

- No false negatives: If it says "not present," it's correct.
- Possible false positives: It might say "present" when it's not.
- Cannot delete items (unless using advanced variants like Counting Bloom filters).

Why it matters in Bitcoin Light Clients: Bloom filters let clients ask full nodes for only the transactions they care about—without revealing exactly which ones. But older versions (like BIP37) leak privacy, making them vulnerable to deanonymization attacks.

- After syncing, the client builds a Bloom filter to track relevant transactions. The filter includes addresses it cares about. The full nodes scan for matches and forward filtered data. This keeps the bandwidth low.
- When a match is found, the client requests a Merkle proof. The full node sends: • transaction ID, • Merkle branch, • block header. The client recomputes the Merkle root and checks against the header.
- Client uses tx hash + Merkle branch to compute the root, then matches it against the header. Match = transaction included. No match = reject. This is Simplified Payment Verification (SPV) from Satoshi's original design.
- So what goes wrong? Old clients use BIP37 Bloom filters. Full nodes are randomly selected. Clients don't validate block contents. Each opens doors for attackers.
- First issue: Bloom filter privacy leaks with BIP37. Attackers analyze filters to deanonymize wallets and correlate transactions. Even masked filters leak address links.
- The fix? Use modern BIP157/158 compact block filters. These don't leak address info, but both client and node must support the upgrade. Old SPV clients using BIP37 remain vulnerable.
- Second issue: Attackers abuse random peer selection to eclipse light clients. - Fill peer list with malicious nodes - Isolate client from honest network - Feed fake headers or suppress transactions
- To defend against eclipse attacks: • Connect to multiple full nodes • Choose peers from different networks • Avoid DNS seeds or fixed IPs Diversity makes isolation much harder.
- Third issue: DoS via false positives. Filters return unrelated txs to obscure activity. Attackers craft txs that match filters. This floods clients with irrelevant data.
- Fourth issue: Transaction censorship. Dishonest full nodes can hide valid transactions from clients. They go ahead to censor, omit, or delay data. The clients cannot detect this since they don't validate full blocks.
- Light clients are easy targets for partial lies. Full nodes can't trick clients into accepting invalid transactions. But they can hide valid ones by omitting them. Result? Censorship and broken UX.