EE 374: Fundamentals of Blockchain Infrastructure **Stanford, Spring 2025**

Lecture 13: Light Clients and Merkle Trees

May 12, 2025, 2025

Lecturer: Prof. David Tse Scribe: David Tse

1 Participants in Bitcoin

Bitcoin has three types of participants:

- Miners: mine new blocks.
- Full nodes: download both block headers and the body, verify the proof-of-work in the headers and the validity of the transactions in the block body.
- Light clients (simplified payment verification, SPV): download only the block headers and verify the proof-of-work, and check for transaction inclusion.

A client can be either a full node or more typically a light client.

The light client concept turns out to have broad applications in blockchain infrastructure beyond Bitcoin. It is used in many other blockchains like Ethereum. The light client concept can also be used to develop *bridges* between blockchains that enable them to transfer information between each other. A trust-minimizing bridge like the Inter-Blockchain Communication (IBC) protocol betewen Cosmos blockchains is essentially one blockchain running a light client of another so that information can be validated before proper transfer. Light clients are also central to the concept of blockchain scaling. In the Ethereum scaling solution of rollups, for example, one can think of the Ethereum main chain as a light client of each of the rollups. We will cover these applications of the light-client concept later in the course.

2 Merkle Trees

2.1 High level idea

The key funtionality of light clients is to be able to check if a transaction is included in a block without downloading the entire block. The key cryptographic primitive used for that purpose is Merkle trees. The discussion here follows Section 8.9 of the crypto book [1].

Merkle tree is a data structure built upon a hash function $H : \{0,1\}^* \to \{0,1\}^\kappa$ (eg. Sha256, $\kappa = 256$). Figure 1 gives an example of a Merkle tree for 4 transactions, each a leaf of the tree. The edge variables y_i's are computed by hashing the variables on the incoming edges. For example, $y_1 = H(tx_1)$, $y_2 = H(tx_2), y_5 = H(y_1, y_2)$. In the light lient application, the root of the Merkle tree, the Merkle root r , is stored in the header of the block that contains all the transactions (tx_1, tx_2, tx_3, tx_4 in this example.)

A Merkle tree is used to answer the following type of query: " Is tx in the ith position of the block?" without downloading all the transactions in the block (there can be thousands of them). To

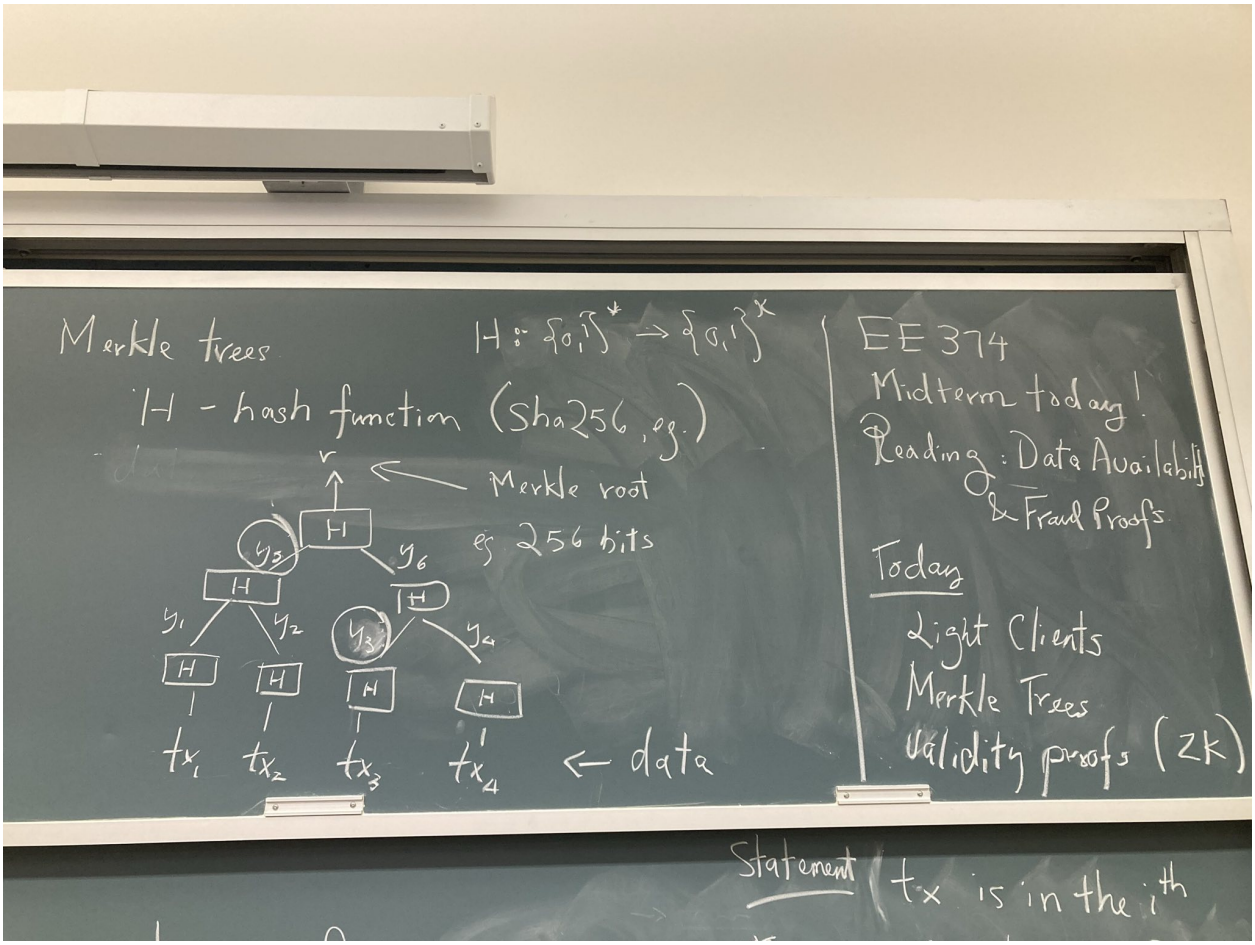

Figure 1: A Merkle tree with 4 leaves.

answer this query, the light client has its disposal the Merkle root r in the block header, which has already been downloaded. In addition to the transaction of interest, the light client is also provided with a *Merkle path* which allows it to check if the given tranaction tx hashes to the Merkle root. For example, if the query is " Is tx in the 4th position of the block?", then the Merkle path is (y_3, y_5). Then it can check if

$$H(y_5, H(y_3, H(tx))) = r?$$

to verify if the transaction tx is indeed in the 4th position of the block.

2.2 Merkle Trees as a Proof System

One can interpret Merkle trees as the data structure for a succinct proof system . A proof system consists of a prover and a verifier. The prover is trying to convince the verifier the truth of a statement about a relation $R(x, w)$ between a public input x and a winess w. The public input is available to both the prover and the verifier (so that they know they are talking about the same thing!). The witness w is large and is only available to the prover. The goal of the prover is to convince the verifier that for the given public input x, the relation $R(x, w)$ holds for some witness w without communicating w to the verifier but only a short proof π.

Figure 2 shows that Merkle trees is an example of a succinct proof system. The prover is a full node that has access to the entire block B, which serves as the witness w. The verifier is a light client that has access only to the block header containing the Merkle root r. The Merkle root r is available to both the prover and the verifier, and serves as the public input x. The relation $R(r, B)$ holds if tx is in the ith position of block B. The Merkle path is the succinct proof (hence also called Merkle proof.) The proof is succinct because it is only logarithmic in size of the block B.

Two important security properties of Merkle trees are:

- Correctness (good things happen): If tx is indeed in the ith position, then probability that the verifier accepts the proof is 1.
- Soundness (bad things don't happen): If tx is not in the ith position, then verifier should accept the proof with negligible probability.

Proof of correctness is straightforward. Proof of soundness follows from the collision resistance of the underlying hash function H.

3 Light Client Security

In earlier lectures, we talk about security of clients of blockchains like Bitcoin or Ethereum. We have assumed these clients are full nodes of the blockchains. Now that we have introduced light clients, it is natural to compare their security with that of the full nodes, and see if there is significant degradation.

Figure 3 compares the security of Bitcoin full nodes and light clients in three dimensions:

- safety and liveness : assuming that the light client successfully connects with at least one honest full node, it will be able to download the longest chain. Hence a light client will have the same safety and liveness guarantee as a full node.

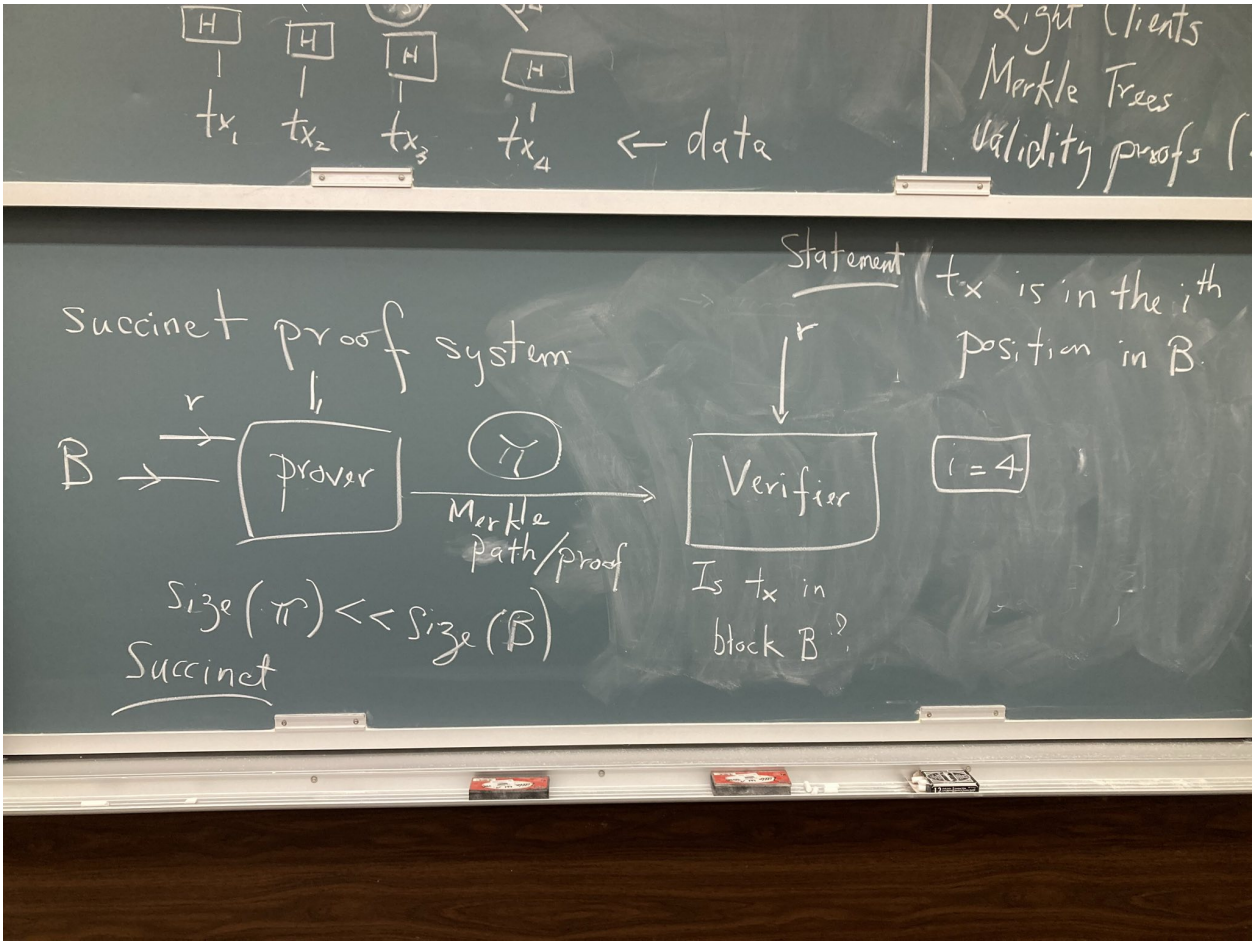

Figure 2: Merkle proof.

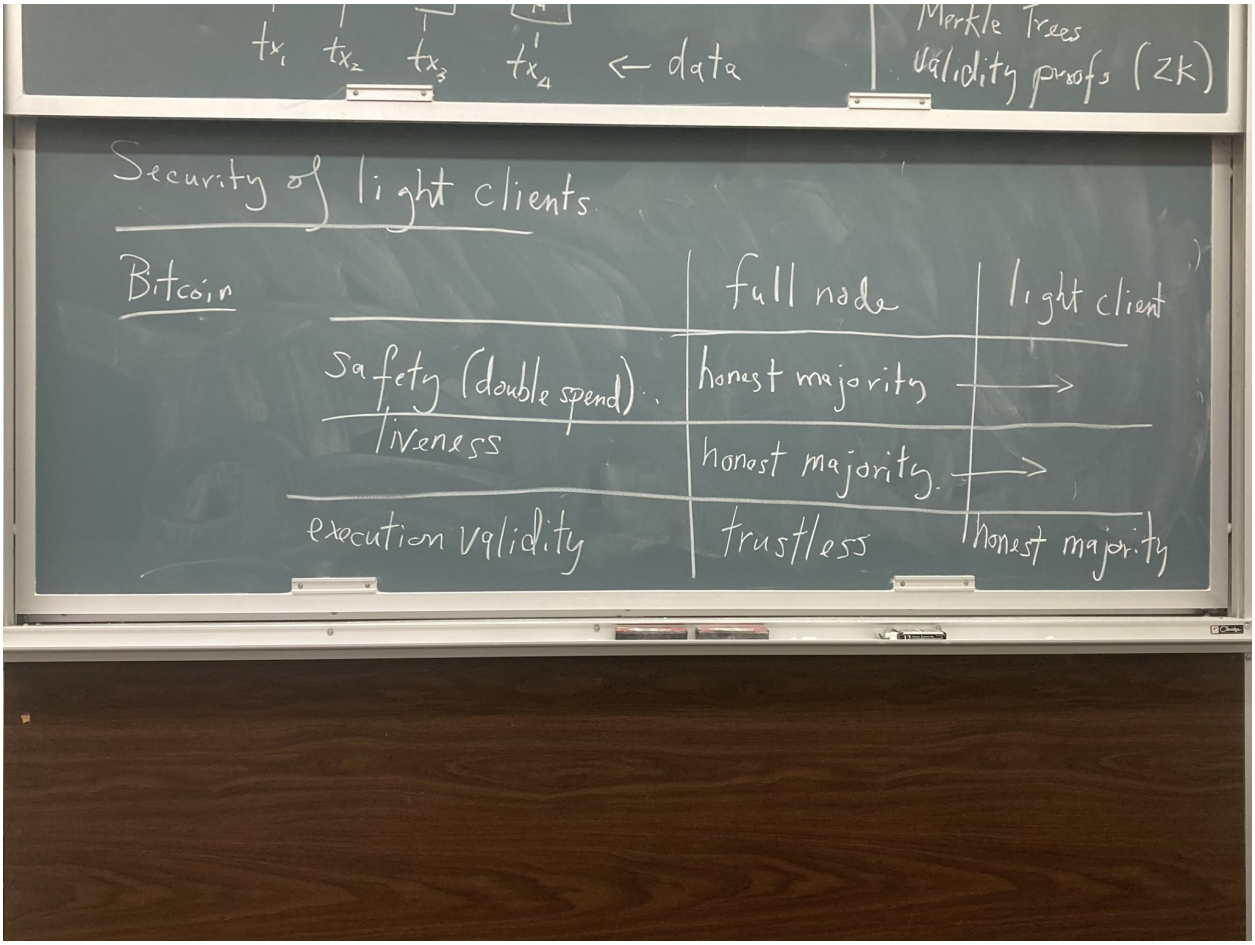

Figure 3: Security of light clients vs full nodes of Bitcoin.

- execution validity: miners are responsible for making sure that the transactions that enter into the Bitcoin ledger are valid (eg. money not created from thin air.) If more than 50% of the mining power is adversarial, then the longest chain can contain invalid transactions because adversarial miners can put whatever transactions they want in the ledger. Full clients will reject the invalid transactions because they can check for ledger validity themselves. Hence, execution validity is trustless for full nodes. Light clients, however, do not have access to the full blocks and hence cannot check for transaction validity. They have to rely on the miners to do the right thing. Hence, the trust assumption for light client validity is that the majority of the miners is honest.

Note that even though light clients can check for transaction *inclusion* trustlessly (via Merkle trees), this cannot guarantee the state on which the tranaction acts on is valid. For example, the light client can check if "Alice pays 1 BTC to Bob", but it cannot check if Alice actually has any BTC!

Figure 4 compares the security of Ethereum full nodes and light clients. The safety and liveness

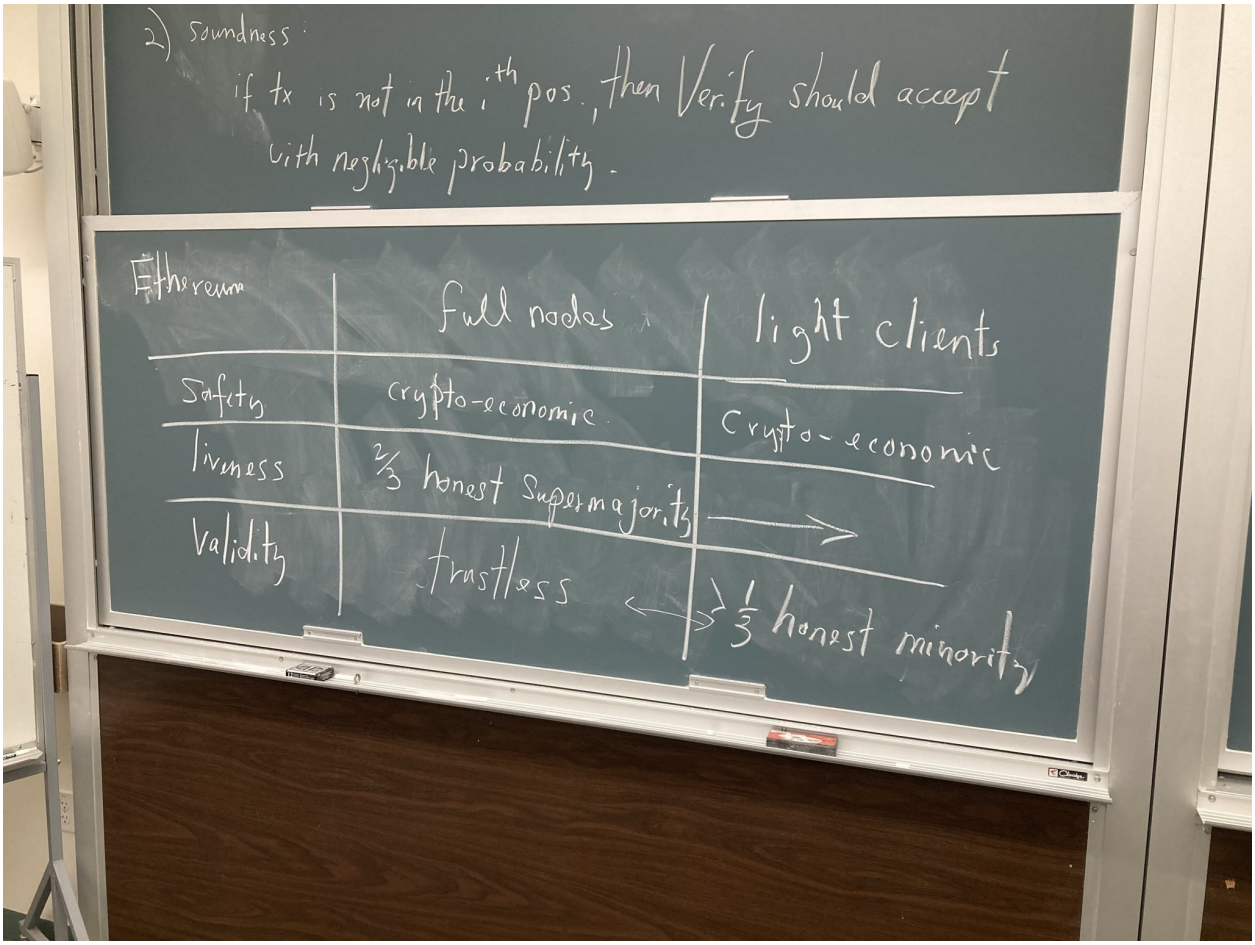

Figure 4: Security of light clients vs full nodes of Ethereum.

security of light clients is the same as that of full nodes. Full nodes can check for execution validity themselves, so they are trustless. In Ethereum (or any Proof-of-Stake BFT protocols), 2/3 of adversarial validators can build their own chain of invalid blocks. So light clients have to trust at least 1/3 honest minority for execution validity.

4 Closing the Gap

In the next lecture, we will see how to close the gap between full nodes and light clients in terms of execution validity. The cryptographic primitives we will need are validity (zero-knowledge) proofs and fraud proofs.

References

[1] D. Boneh and V. Shoup. *A Graduate Course in Applied Cryptography.* 2008.