ECE 595: Foundations of Blockchain Systems **November 4, 2020**

Lecture 11

Lecturer: Sreeram Kannan *Scribe: Soubhik Deb*

Outline: In this lecture, we first discuss a type of commitment structure called Merkle tree and its use in blockchains to construct light nodes. Then, we discuss decoupled validity and data availability problem that are introduced due to usage of Merkle tree in blockchains.

11.1 Commitment Structures

Consider a collision-resistant hash function $H(.)$. Suppose Alice has a data item D_1 and she evaluates $H(D_1)$ which she presents it to Bob. This $H(D_1)$ will serve as a commitment of Alice to the data item D_1. This is because, due to $H(.)$ being a collision-resistant hash function, Alice can't later on get a new data item D_2 such that $H(D_2) = H(D_1)$ and tell Bob that she committed to D_2. Extending this further, suppose Alice has m data items

$$D_1, D_2, \cdots, D_m$$

and Alice wants to commit to all these data items. There are two possible schemes:

- **Scheme 1**- hash the concatenation of the data items, i.e., $H(D_1, D_2, \cdots, D_m)$, or
- **Scheme 2**- concatenation of the hashes of the data items, i.e., $H(D_1), H(D_2), \cdots, H(D_m)$.

Now, the question is what are the trade-offs between these two possibilities? For that, we define two properties under which we will distinguish the two possibilities:

- **Commitment size**- the size of the commitment
- **Proof of membership**- amount information that is needed to be provided by Alice to Bob, in addition to data item, so that Bob can verify that the data item corresponds to the commitment made by Alice previously.

Assume that each of the m data items is of size S and output of the hash function is of the size H. For scheme 1, Alice will just need to provide $H(D_1, \cdots, D_m)$ which is of size $O(1)H$ whereas for proving its membership, Alice has to provide all the m data items. On the other hand, in Scheme 2, Alice would be providing $H(D_1), \cdots, H(D_m)$ to Bob, which is of size mH but for proof of membership, Alice has to provide hash of only one data item which is of size S. Ideally,

we would like to have best of both worlds, that is, a commitment size of $O(1)H$ and proof of membership of $O(1)S$. We will next describe Merkle tree which takes us near to this.

Commitment Schemes	**Commitment Size**	**Proof of Membership**
Scheme 1	$O(1)H$	$O(m)S$
Scheme 2	$O(m)H$	$O(1)S$
Ideal	$O(1)H$	$O(1)S$
Merkle Tree	$O(1)H$	$O(1)S + O(\log m)H$

11.2 Merkle Trees

Consider we have m data items, i.e., $D_1, \cdots, D_m$.For constructing the Merkle tree, shown in Fig 11.1, first obtain the hash of each of the items, $H_1 = H(D_1), \cdots, H_8 = H(D_8)$. Then, for all $i \in \{1, 2, \cdots, \frac{m}{2}\}$, we obtain $H_{2i-1,2i} = H(H_{2i-1} \mid\mid H_{2i})$. Thus, at a level $j \in \{2, \cdots, \log m\}$, we evaluate $H_{(i-1)2^{j-1}+1 \cdots i2^{j-1}} = H(H_{(i-1)2^{j-1}+1 \cdots (i-1)2^{j-1}+2^{j-2}} \mid\mid H_{(i-1)2^{j-1}+2^{j-2} \cdots i2^{j-1}})$ for $i \in \{1, \cdots, \frac{m}{2^{j-1}}\}$. Finally, we get the root of the Merkle tree (called as Merkle root) at the level $j = \log m$, i.e., $H_{12 \cdots m} = H(H_{12 \cdots \frac{m}{2}} \mid\mid H_{\frac{m}{2} \cdots m})$.

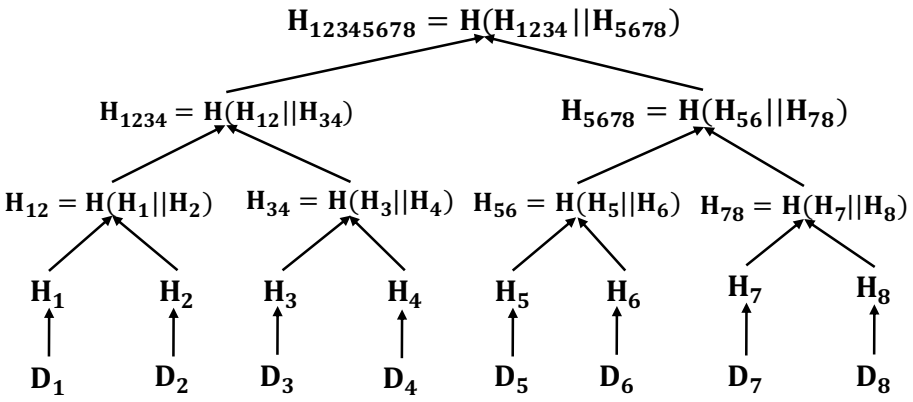

Figure 11.1: Merkle Tree for $m = 8$.

Now, for commitment to m data item $D_1, \cdots, D_m$, Alice has to just provide the Merkle root of the corresponding Merkle. This is of the size H. On the other hand, for providing proof of membership for a data D_i, Alice has to provide to Bob the data D_i and all the sibling hashes encountered in the path of Merkle tree from H_i to the Merkle root. For example, consider the example illustrated in Fig 11.1 and suppose Alice wants to provide proof of membership for D_2 to Bob. Then, Alice has to give $D_1, H_1, H_{34}, H_{5678}$. Thus, for Merkle tree, commitment size is

$O(1)H$ and the proof of membership is $O(1)S + O(\log m)H$.

11.3 Merkle Trees in Blockchain

Previously, the blocks would contain the transactions which would increase the bandwidth requirement of the network and also require storage of all the transactions of a block in all the participant nodes. Instead of including all the transactions in the block, only include the merkle root of the merkle tree for those transactions. The sender of a Tx included in the block only has to show that Tx and the associated proof of membership in the Merkle tree in order to convince the recipient of that Tx. This modification introduces the concept of *Light nodes* which is described in the next section.

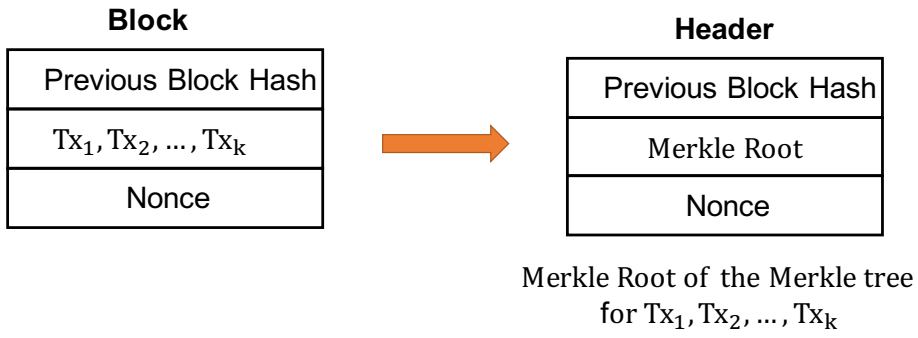

Figure 11.2: Transformation into the chain of headers.

11.3.1 Light Nodes

Considering the transformation illustrated in Fig 11.2, if a header is $k-$deep in blockchain, then we know that the transactions contained in the merkle tree referred by the merkle root of that header is also $k-$deep and thus is also a confirmed transaction. This fact can be used to describe a new type of nodes called light nodes (simplified payment validation - SPV) where:

- maintains headers of blocks in blockchains,
- can receive payments securely,
- no need to download the whole block for validation,
- don't mine blocks.

11.3.2 Decoupled Validity

The next question is:

> *Why not miners also just maintain chain of headers and don't download the whole blocks, i.e., the actual transactions?*

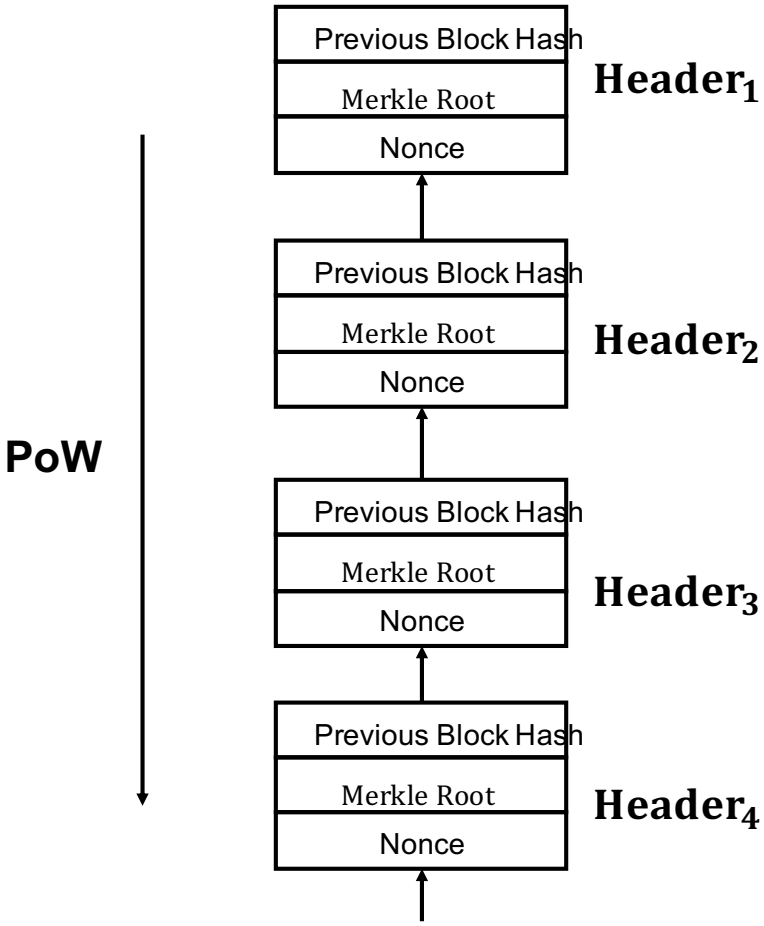

Figure 11.3: Chain of headers.

Referring to Fig 11.3, there are two parts to validation:

- **PoW and chain structure.** This is validating that in each header, correct previous header is mentioned, appropriate nonce has been described that satisfies the threshold.
- **Content validity.** While mining a block, the miner doesn't know whether it is including double-spending transactions or not as the miner doesn't know the previous transactions.

One way to get around this problem of content validity is by decoupling these two validations. All the miners first comes to consensus on the chain structure, i.e., the ordering of the headers. Then, post-facto, a node can reject double spends using sanitization of the ledger. This sepa-

rates/decouples the execution of the transaction from the ordering. Now, we have three types of nodes:

1. **Light Nodes.** They just maintain the chain of headers, can receive payments, don't download whole block, don't mine blocks.
2. **Miners.** They do not validate blocks, they mine blocks, they don't download blocks.
3. **Full nodes.** They download all blocks and check for validity, they can mine blocks.

Observe that here the mining is very fast as the miner doesn't have to wait for downloading the blocks.

11.3.3 Data Availability Problem

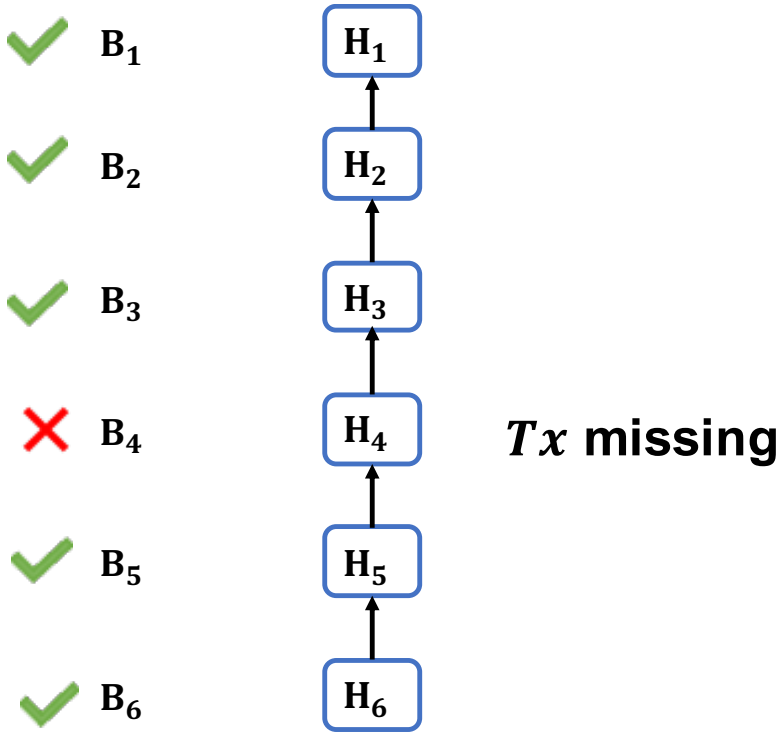

Figure 11.4: Data availability problem.

The miner of the block B_4 is an adversary and included a fake transaction Tx that no honest node has heard of. When a full node tries to download the Tx contained in the Merkle tree of the header H_4, no honest node has it and adversarial miner doesn't reply back. So, now validation via ledger sanitization in the full node is stuck at block B_4 and the blockchain stalls - a liveness problem.

This is the data availability problem. Note that you cannot just skip over the missing transactions because then every node has to agree on this skipping decision and this is just another consensus problem and in blockchain, the only way to achieve consensus is via the chain.
A simple solution to this problem is by decoupled validation. The miners:

- before mining a block, download all blocks,
- build only on those previous headers for which miners can download blocks
- do not execute transaction.

11.3.4 Joining Problem in Blockchain

When a new node joins the blockchain,

- it queries ℓ nodes about their respective longest header chain,
- pick the longest chain and download that chain and the associated blocks.

The question is *when is this safe?*. The answer is as long as 1 out of the ℓ nodes is honest, the new node will know about the longest chain. This is called **any trust** assumption.