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

Lecture 14: Validity and Fraud Proofs, and Rollups

May 14, 2025

Lecturer: Prof. David Tse Scribe: Ertem Nusret Tas

1 Validity of a Blockchain

For full nodes of a blockchain (e.g., Bitcoin, Ethereum), no trust assumptions besides the correctness of the blockchain's virtual machine (e.g., the Bitcoin script, Ethereum Virtual Machine) are needed to infer the validity of transaction execution. Indeed, the full nodes can execute the transactions themselves, obtain the latest blockchain state, and check if the state root recorded on the blockchain matches what they have obtained[1]. Note that this 'trustless-ness' property of validity is quite strong: even if all validators (i.e., consensus nodes) of the blockchain are adversarial, they cannot convince the full nodes to output an invalid transaction as part of their ledgers.

2 Validity and Fraud Proofs

The purpose of validity and fraud proofs is to enable light clients of the blockchain to verify the validity of the transactions and the state obtained by executing them, *without downloading and executing all of these transactions themselves.*

Formally, let st_i denote the state of the blockchain after executing i transactions, and $\phi(\cdot)$ denote the *state transition function*. The function $\phi(\cdot)$ takes as input the old state and a sequence of transaction (or a single transaction), and outputs the new state: $\mathsf{st}_i = \phi(\mathsf{st}_0, (\mathsf{tx}_1, \ldots, \mathsf{tx}_i))$. For instance, st_i in the case of Bitcoin and Ethereum would respectively be the set of UTXOs, and the accounts of the Ethereum clients, whereas state transition would be determined by the Bitcoin script and the Ethereum Virtual Machine (EVM).

Now, the state st_i corresponding to the blockchain state at some block B is valid if there exist some transactions $\mathsf{tx}_1, \ldots, \mathsf{tx}_i$ such that

- $\mathsf{st}_i = \phi(\mathsf{st}_0, (\mathsf{tx}_1, \ldots, \mathsf{tx}_i))$, where st_0 is the correct *genesis state* enshrined by the blockchain's genesis block.
- $(\mathsf{tx}_1, \ldots, \mathsf{tx}_i)$ is the sequence of transactions included in the blockhain, up until (and including) block B.

Let stc_i denote the *succinct commitment* to the state st_i. For instance, Ethereum commits to the state (i.e., the sequence of accounts) using a (sparse) Merkle tree, and the Merkle root is the succinct commitment to the Ethereum state.

[1] State root denotes a commitment to the blockchain state. As we have seen in the class, an example of a state root is a Merkle root to a sequence of account balances.

2.1 Validity Proofs

A validity proof is a *succinct* proof that convinces the light clients that a given state committed by the value stc_i is valid[2]. Suppose the transactions $\mathsf{tx}_1, \ldots, \mathsf{tx}_k$ are included within a block B, and let root denote the Merkle root committing to the transactions. Let stc_{i-1} denote the valid blockchain state before block B. Here, we assume that we update the state block-by-block, instead of after every transaction. Then, a succinct validity proof π enables the light client to verify that the state underlying stc_i is valid, given only stc_{i-1} and root. Formally, we define the relation

$$\begin{aligned} R := \{(x = (\mathsf{stc}_{i-1}, \mathsf{root}, \mathsf{stc}_i),\ w = (\mathsf{st}_{i-1}, \mathsf{st}_i, (\mathsf{tx}_1, \ldots, \mathsf{tx}_k)) :& \\ \mathsf{root} \text{ is the Merkle tree root of } (\mathsf{tx}_1, \ldots, \mathsf{tx}_k) \wedge& \\ \mathsf{stc}_{i-1} \text{ is a commitment to the state } \mathsf{st}_{i-1} \wedge& \\ \mathsf{stc}_i \text{ is a commitment to the state } \mathsf{st}_i \}& \end{aligned}$$

Here, x denotes the (public) problem statement part of the relation, and w denotes the witness. The proof π enables light clients to verify against x that there indeed exists a witness w for x, i.e., x is in the language defined by the relation R, the language of valid state roots.

A trivial question here would be why one needs a proof π, since providing the witness w would also prove to the light client that a given x is in the language. Although this is true, the witness w is as large as the total state, including at least *all* transactions within a block B. In contrast, π is either constant size (three elliptic curve group elements in Groth16 [2]), or poly-logarithmic.

The validity proofs are typically generated using *succinct non-interactive arguments of knowledge* (SNARKs), a cryptographic object characterized by the following algorithms:

- $S(R) \to (\mathsf{pk}, \mathsf{vk})$, where pk is the public key and vk is the verifier key.
- $P(\mathsf{pk}, x, w) \to \pi$, where π is the proof.
- $V(\mathsf{vk}, x, \pi) \to 0/1$, where 1 denotes accepting the proof, and 0 denotes rejection.

The SNARK must satisfy the following properties:

- **Completeness:** $\forall x, w$, if $R(x, w) = 0$ (relation is satisfied, note that we use 0 to denote this), then $\mathsf{Pr}[V(\mathsf{vk}, x, P(\mathsf{pk}, x, w)) = 1] = 1$.
- **Knowledge Soundness:** Informally, if V outputs 1, then the prover running P 'knows' a witness w such that $R(x, w) = 1$.

2.2 Fraud Proofs

One limitation of the validity proofs solution is that it requires posting proofs periodically to the blockchain, i.e., on the happy path. To resolve this limitation, blockchains use so-called *optimistic* methods, where the state is updated by the validators without any proving in normal operation, but is under constant supervision of so-called *watchtowers*, that execute the transactions, obtain the correct state roots and compare those with the ones included in the blockchain. When the

[2]Here, succinct means that the size of the proof is constant or poly-logarithmic in the size of the witness described below.

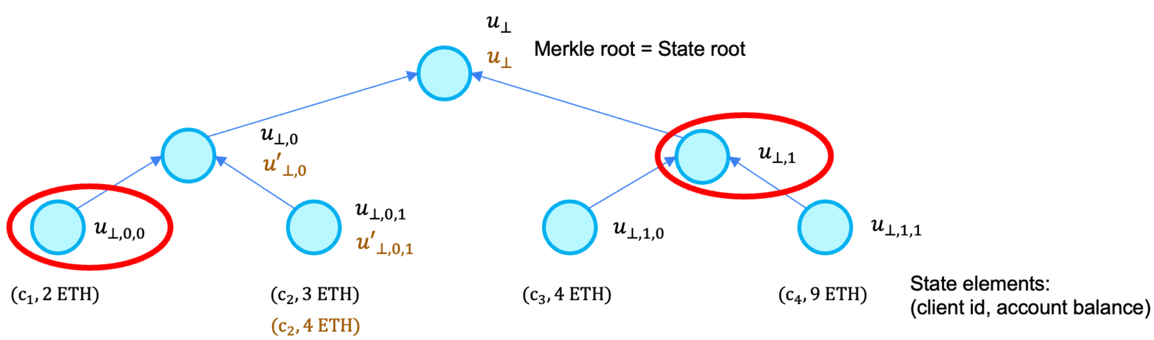

Figure 1: State consists of 4 elements, each denoting the account balance of a client/user. The client 2's account is updated by a transaction. To enable the light clients to compute the new Merkle (state) root, the watch towers publish the inner nodes circled in red. Then, hashing $(c_2, 4\ \mathsf{ETH})$, light clients find the new value of the inner node $u'_{\perp,0,1}$. Then, they find the new value of the inner node $u'_{\perp,0} = H(u_{\perp,0,0}, u'_{\perp,0,1})$ and the Merkle root $u_\perp = H(u'_{\perp,0}, u_{\perp,1})$ iteratively.

validators turn malicious, and confirm an invalid transaction, or include an invalid state root, these watch towers can raise an alarm by publishing *fraud proofs*.

Recall that we were posting one state root per block. For efficiently-verifiable fraud proofs, validators post state roots after each transaction. Suppose the first invalid state root is st'_i, and watchtowers noticed that the correct state root at that position must have been st_i. Then, the watchtowers can raise an alarm by simply pointing out that the correct state root must have been st_i.

Now, how does a light client verify that indeed there was an incorrect state? Note that we do not want the light clients to download all of st_{i-1} and find $\mathsf{st}_i = \phi(\mathsf{st}_{i-1}, \mathsf{tx}_i) \neq \mathsf{st}'_i$ by naively running $\phi(\cdot)$. Lucikly, the structure of Merkle trees enables the light clients to update the old state root st_{i-1} given the transaction tx_i and obtain the new state root by downloading only logarithmic amount of information in the size of the state (see Fig. 1). To facilitate this solution, the watch towers provide the sibling nodes of the inner nodes on the path from the state elements updated by the transaction tx_i (suppose there is only one such state element) to the root. Then, by applying tx_i to these state elements and iteratively calculating the new inner nodes of the Merkle tree, the light clients can derive the new Merkle root, i.e., the new state root. This solution was pioneered by AlBassam, Sonnino, Buterin and Khoffi [1], which gives a detailed explanation of it (also posted on the website).

3 Rollup

Recall the *blockchain trilemma*, i.e, the tension among decentralization, throughput and security. There are two potential solutions to this trilemma:

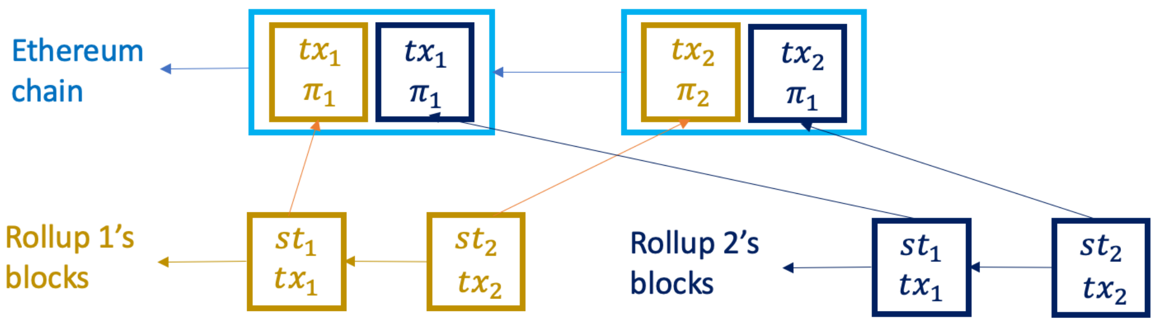

Figure 2: Example of a ZK rollup. Rollup transactions are posted to Ethereum along with a proof that verifies the state. In this example, there are two rollups with their own transactions, denoted by the colors yellow and dark blue.

- **Vertical scaling** corresponds to demanding more compute and bandwidth from each blockchain node. For instance, Solana requires validators with stronger compute, and can thus achieve $2000 - 5000$ transactions per second. Although this ensures better throughput, it comes at the expanse of decentralization, by excluding computationally weaker nodes from being a validator.
- **Horizontal scaling** aims to increase throughput while keeping security and decentralization. A major example of horizontal scaling is the rollup (layer 2 or L2) architecture.

The scaling problem of Ethereum gained attention due to the CryptoKitties incident, which revealed the *execution bottleneck* of Ethereum. There are three metrics of scaling: communication, execution and storage. For Ethereum, the main bottleneck, as observed by the CryptoKitties incident was execution. In this context, rollups emerged as a prominent solution to resolve Ethereum's execution bottleneck.

In the rollup architecture, a sequencer batches transactions into blocks, and posts these blocks to Ethereum (Fig. 2). Then, Ethereum essentially acts as a light client of the rollup transactions. Although rollups bundle and post their transactions to Ethereum, the Ethereum validators do not execute the rollup transactions. Instead, for the so-called ZK rollups, Etheruem validators only verify the validity proofs posted for the state of the rollup's execution, or for the so-called optimistic rollups, they only verify the fraud proofs (when there is a fraud). This way, Ethereum validators know that the rollup transactions and state are valid, without having to execute them. In this sense, Ethereum acts as an execution light client of rollups (not a full light client, as the rollups still post all their data to Ethereum).

References

[1] M. Al-Bassam, A. Sonnino, V. Buterin, and I. Khoffi. Fraud and data availability proofs: Detecting invalid blocks in light clients. In *Financial Cryptography (2)*, volume 12675 of *Lecture*

Notes in Computer Science, pages 279–298. Springer, 2021.

[2] J. Groth. On the size of pairing-based non-interactive arguments. In *EUROCRYPT (2)*, volume 9666 of *Lecture Notes in Computer Science*, pages 305–326. Springer, 2016.