Table of Contents

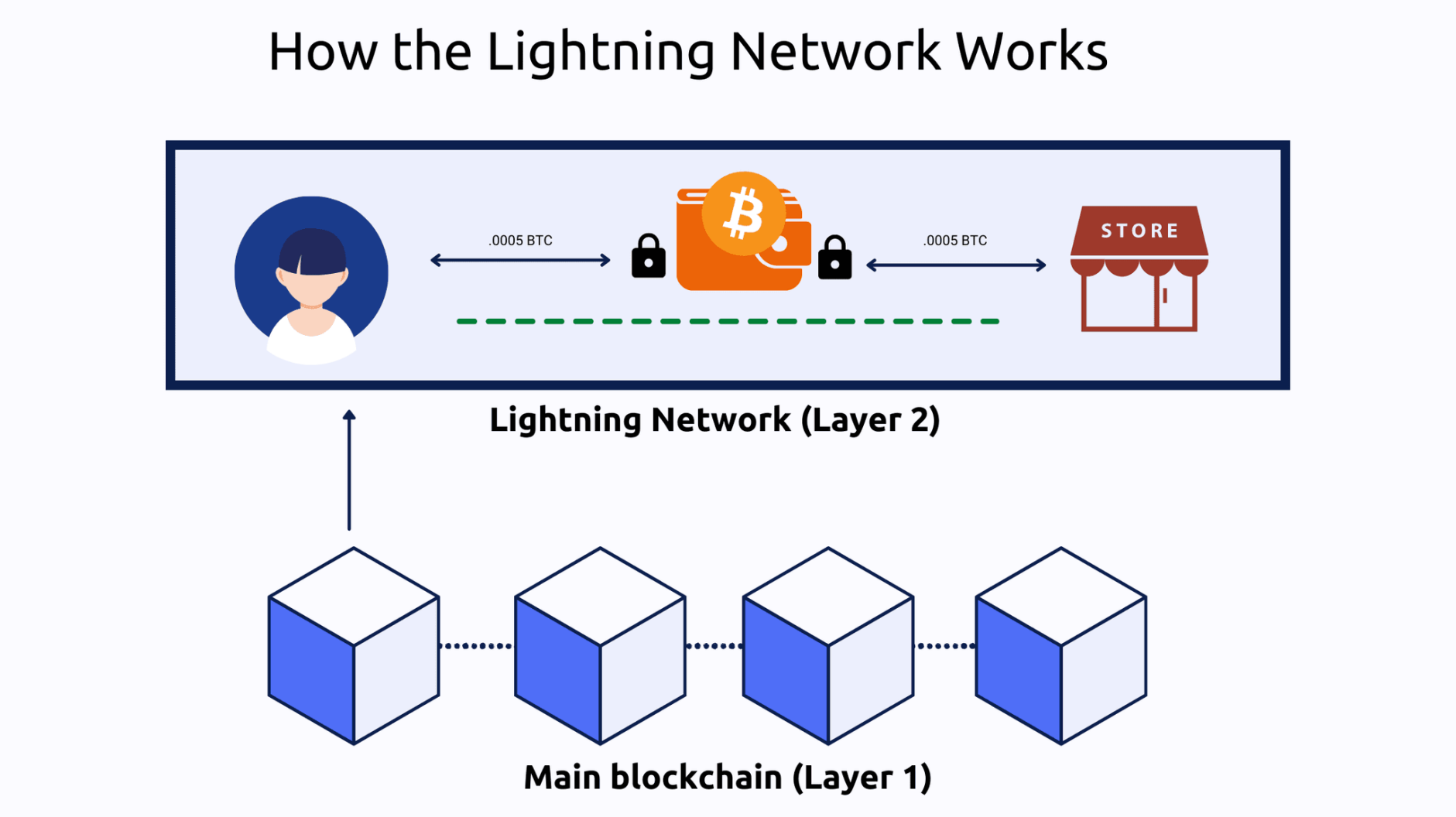


Figure: Source: BitPay Blog – `https://www.bitpay.com/blog/what-is-the-lightning-network`

Funding Transaction

How to lock funds on Bitcoin?

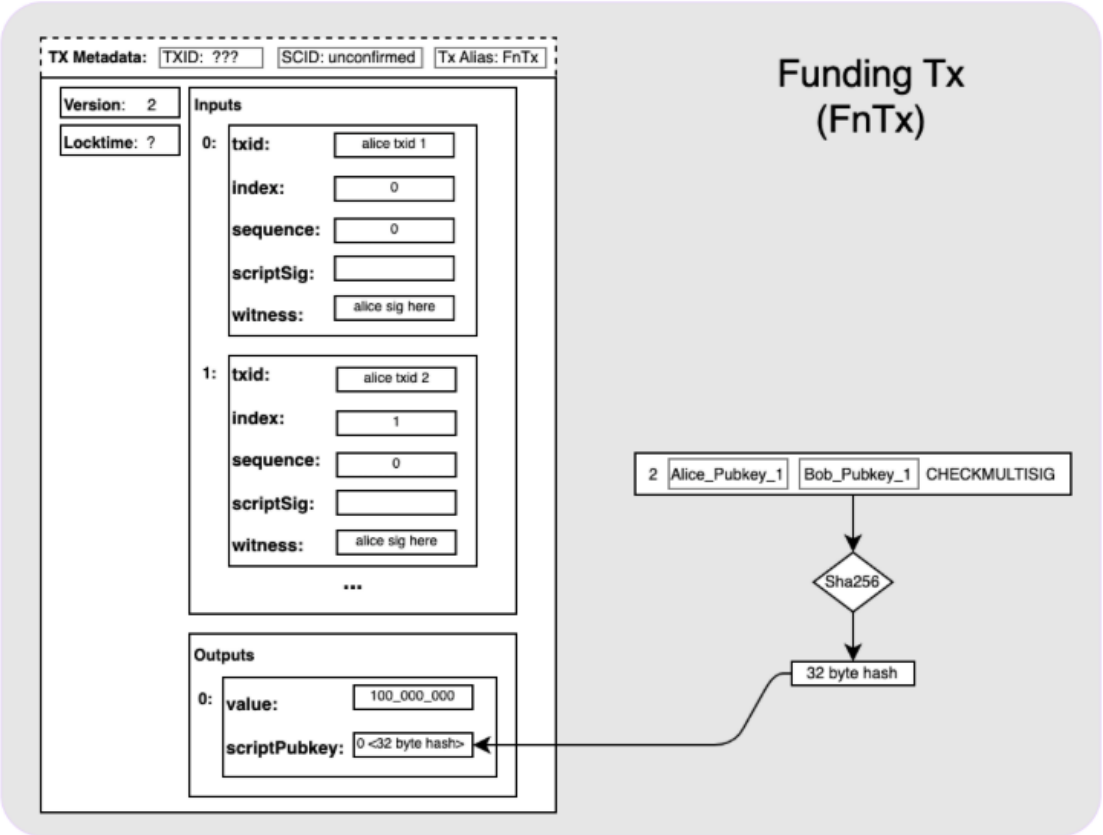


Figure: Taken from Elle Mouton's blog.

Transactions

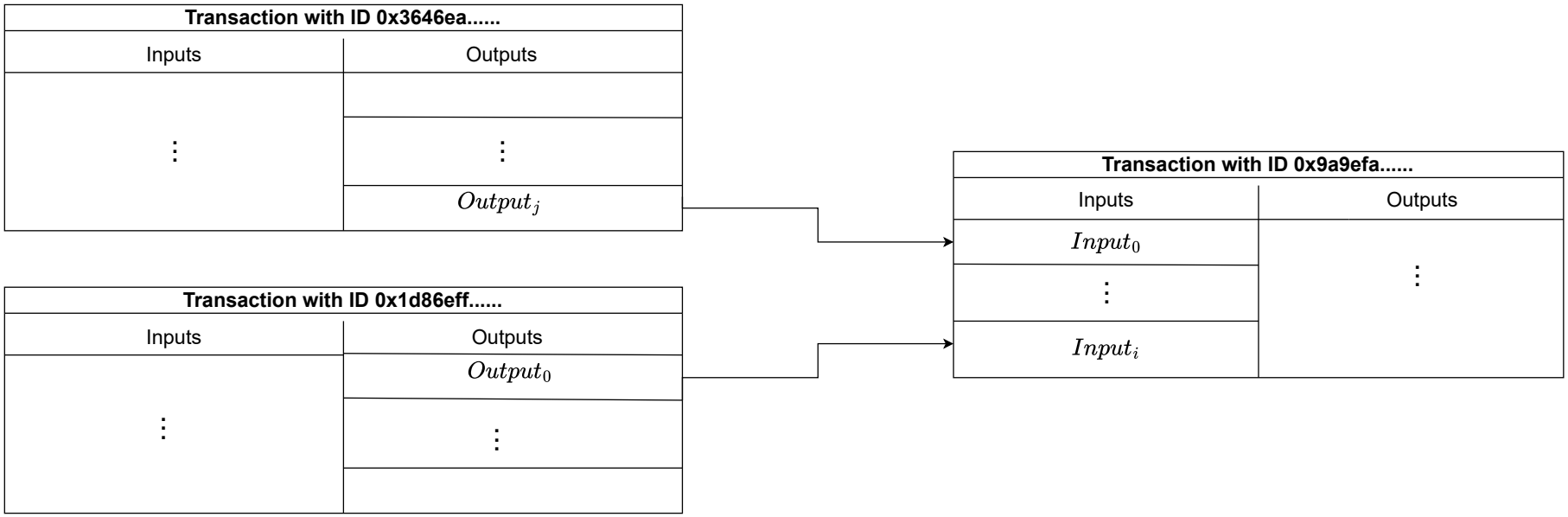


Figure: Structure and relation of transactions in Bitcoin.

Channel Announcement

Announcing the created channel to the network proving the following:

- **Existence of Funding TX:** Validating funds are unspent and existing on the blockchain.
- **Liability of the TX:** Funding transaction is of correct type for a Lightning channel.
- **Funding TX Belongs to the Nodes:** Nodes are in control of the blocked funds.
- **Message Validity:** Message corresponds to the correct node IDs in the Lightning network.

Problem Statement

- **Privacy:**
  - Lightning node IDs are unique among nodes.
  - A node has only one ID for all its channels.
  - Possible connection of the ID and the IP address of the node.

Problem Statement: Channel Geolocations

Figure: Photo from mempool.space showing the geolocation of Lightning channels except nodes using Tor.

Problem Statement

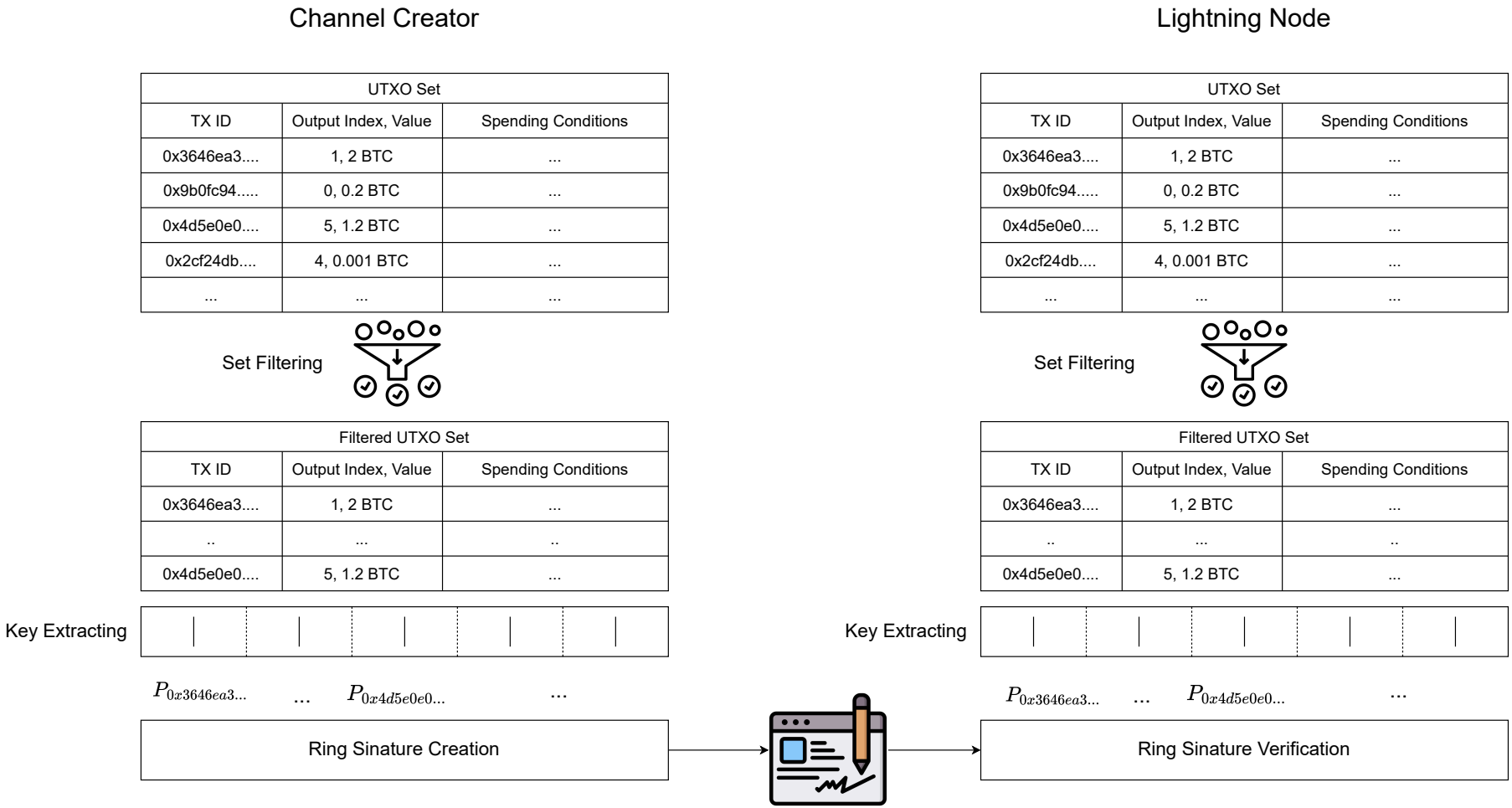
In other words, a channel announcement message is provided for the following purposes:

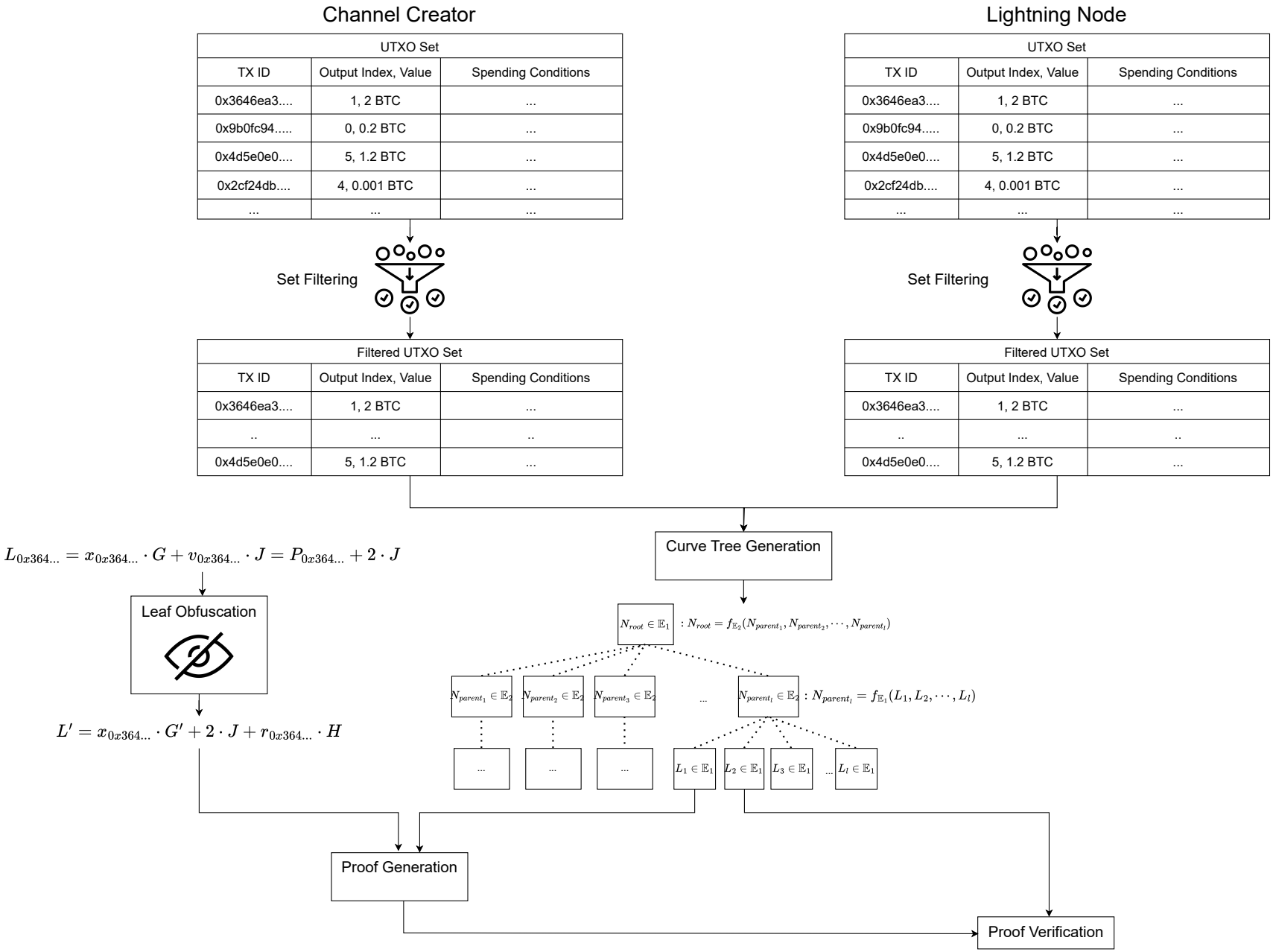
- **Set membership:** Proving that a UTXO is a member of the UTXO set.
- **Key ownership:** Proving that the sender of the message owns the public key unlocking the UTXO.

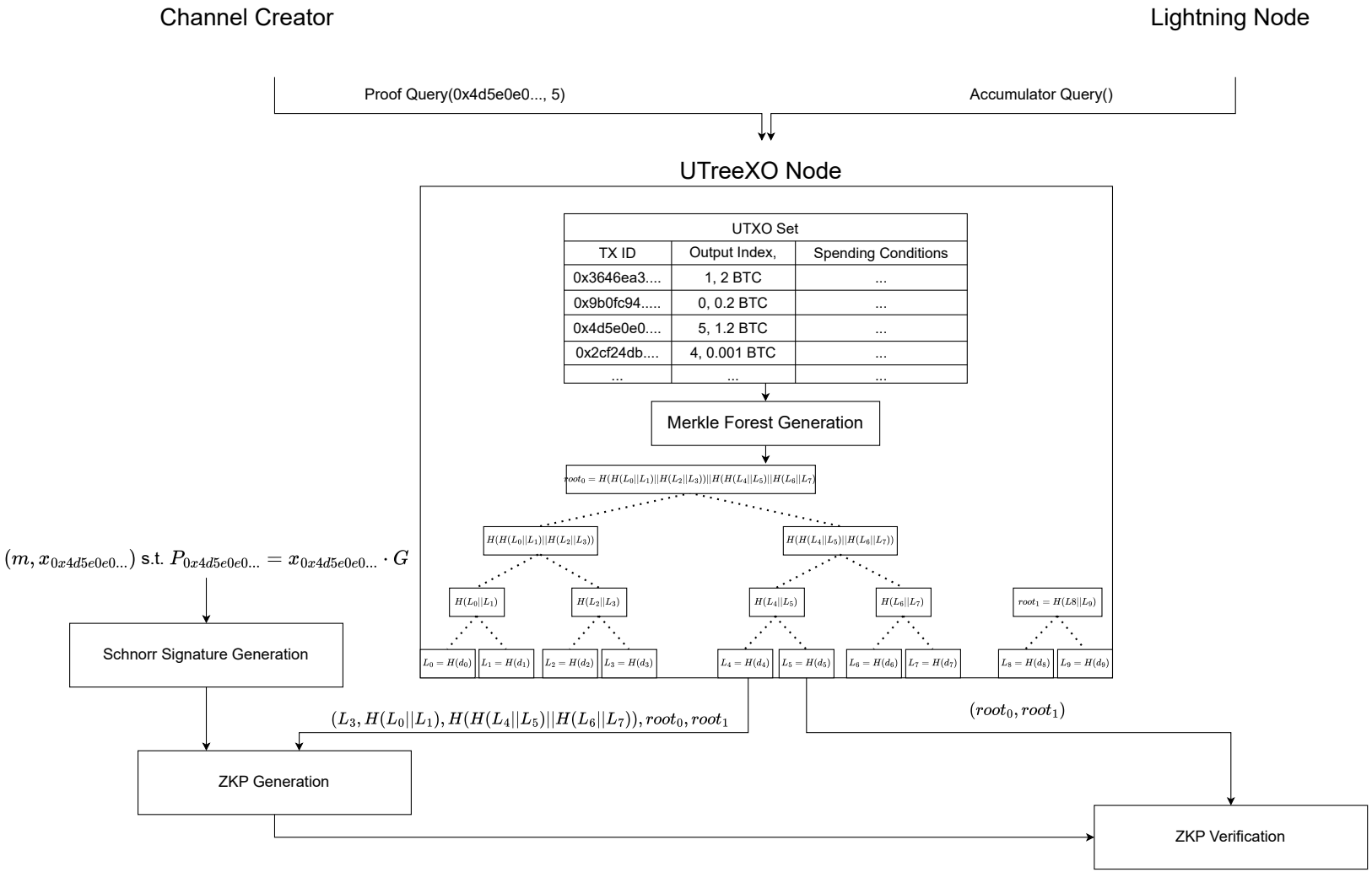
UTXO Set		
TX ID	Output Index, Value	Spending Conditions
0x3646ea3....	1, 2 BTC	...
0x9b0fc94.....	0, 0.2 BTC	...
0x4d5e0e0....	5, 1.2 BTC	...
0x2cf24db....	4, 0.001 BTC	...
...	...	...

Contents

Taproot Ring Signature




Channel Creator
UTXO Set
TX ID
Output Index, Value
Spending Conditions
0x3646ea3....
1, 2 BTC
...
0x9b0fc94.....
0, 0.2 BTC
0x4d5e0e0....
5, 1.2 BTC
0x2cf24db....
4, 0.001 BTC
Set Filtering
Filtered UTXO Set
..
Lightning Node
$L_{0x364...} = x_{0x364...} \cdot G + v_{0x364...} \cdot J = P_{0x364...} + 2 \cdot J$
Leaf Obfuscation
$L' = x_{0x364...} \cdot G' + 2 \cdot J + r_{0x364...} \cdot H$
Curve Tree Generation
$N_{root} \in \mathbb{E}_1$: $N_{root} = f_{\mathbb{E}_2}(N_{parent_1}, N_{parent_2}, \cdots, N_{parent_l})$
$N_{parent_1} \in \mathbb{E}_2$
$N_{parent_2} \in \mathbb{E}_2$
$N_{parent_3} \in \mathbb{E}_2$
$N_{parent_l} \in \mathbb{E}_2$: $N_{parent_l} = f_{\mathbb{E}_1}(L_1, L_2, \cdots, L_l)$
$L_1 \in \mathbb{E}_1$
$L_2 \in \mathbb{E}_1$
$L_3 \in \mathbb{E}_1$
$L_l \in \mathbb{E}_1$
Proof Generation
Proof Verification


Channel Creator
Lightning Node
Proof Query(0x4d5e0e0..., 5)
Accumulator Query()
UTreeXO Node
UTXO Set
TX ID
Output Index,
Spending Conditions
0x3646ea3....
1, 2 BTC
...
0x9b0fc94.....
0, 0.2 BTC
...
0x4d5e0e0....
5, 1.2 BTC
...
0x2cf24db....
4, 0.001 BTC
...
...
...
...
Merkle Forest Generation
$root_0 = H(H(L_0||L_1)||H(L_2||L_3))||H(H(L_4||L_5)||H(L_6||L_7))$
$H(H(L_0||L_1)||H(L_2||L_3))$
$H(H(L_4||L_5)||H(L_6||L_7))$
$H(L_0||L_1)$
$H(L_2||L_3)$
$H(L_4||L_5)$
$H(L_6||L_7)$
$root_1 = H(L8||L_9)$
$L_0 = H(d_0)$
$L_1 = H(d_1)$
$L_2 = H(d_2)$
$L_3 = H(d_3)$
$L_4 = H(d_4)$
$L_5 = H(d_5)$
$L_6 = H(d_6)$
$L_7 = H(d_7)$
$L_8 = H(d_8)$
$L_9 = H(d_9)$
$(m, x_{0x4d5e0e0...})$ s.t. $P_{0x4d5e0e0...} = x_{0x4d5e0e0...} \cdot G$
Schnorr Signature Generation
$(L_3, H(L_0||L_1), H(H(L_4||L_5)||H(L_6||L_7)), root_0, root_1$
$(root_0, root_1)$
ZKP Generation
ZKP Verification

Conclusion

In conclusion, each of the projects use the following cryptographic schemes to solve the key ownership and set membership problems.

Problem	Taproot Ring Signature	AUT-CT	Output Zero
Set Membership	Ring Signature	Curve-tree	Merkle-tree
Key Ownership	Ring Signature	Zero-knowledge proofs	Zero-knowledge proofs

Contents

ZKPs in Practice

- ZKPs were initially presented for specific problems, nowadays zkSTARKs and zkSNARKs allow us to prove arbitrary statements.
- Here, ZKPs are modeled as a protocol for proof generation and verification, as outlined below:

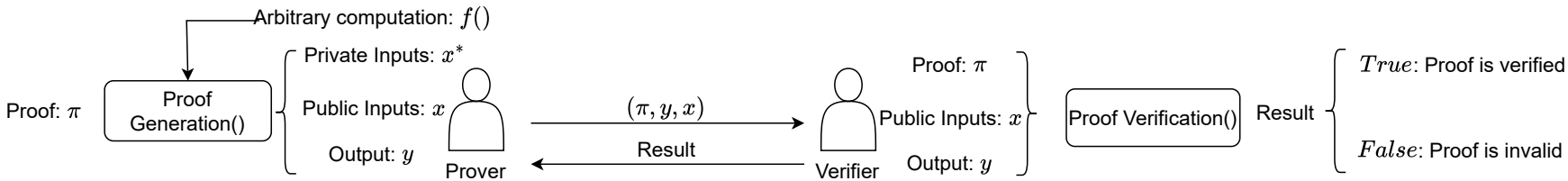


Figure: General structure of zkp generation and the interaction between the entities.

How to go from arbitrary computation to mathematical proofs?

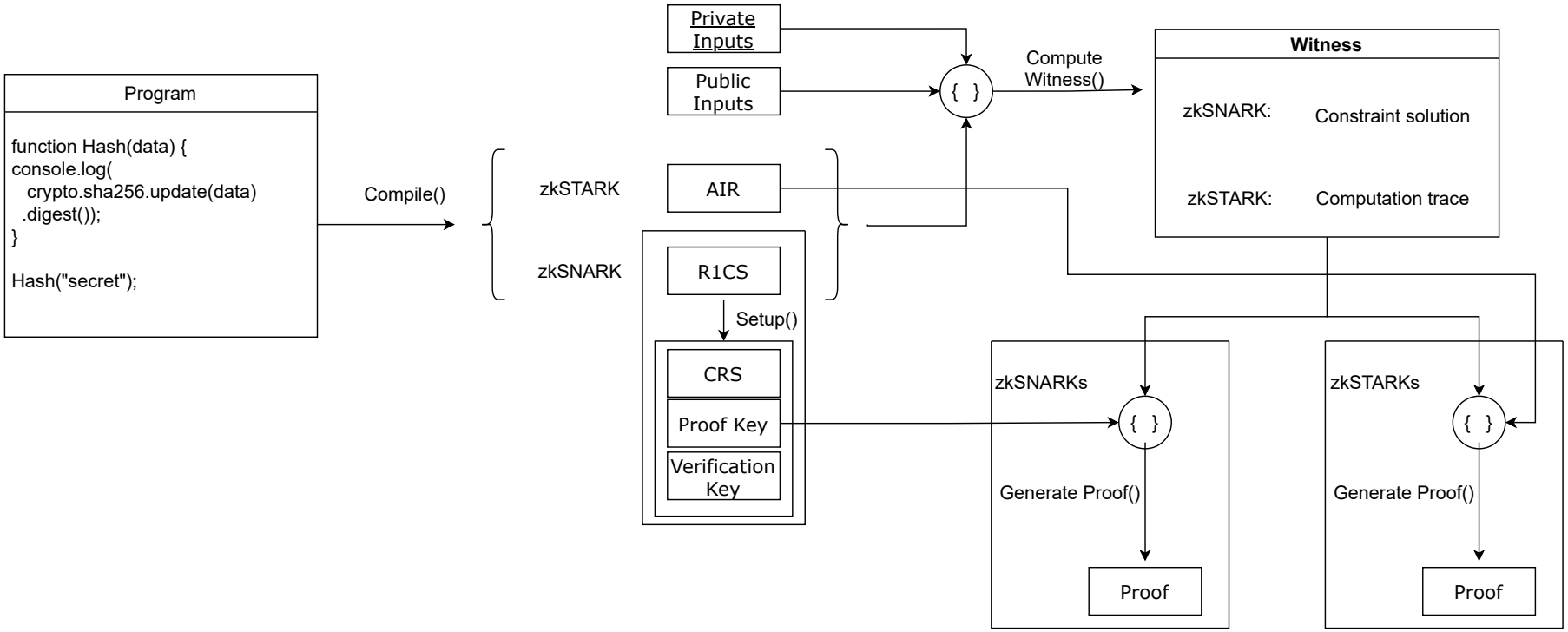


Figure: zkSNARK and zkSTARK generation process.

UTreeXO provides a method of keeping track of the UTXO set for nodes[1] in Bitcoin using Merkle trees.
UTXO set changes with each transaction included in the blockchain. How can we manage changes in a Merkle tree?
Using a forest of perfect binary trees as below:

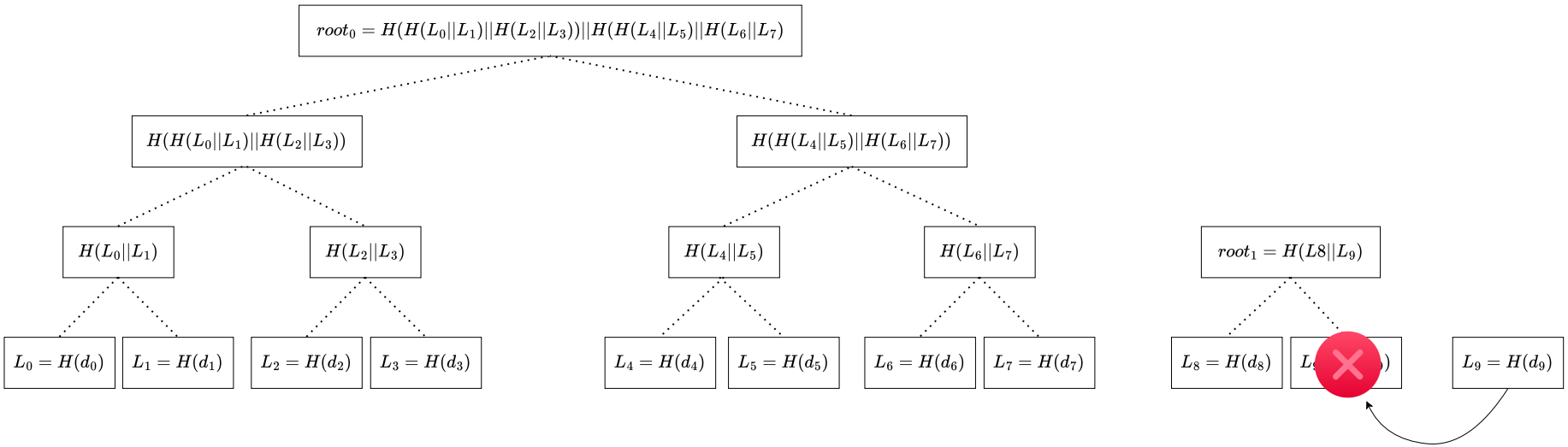


Figure: Forest of three full binary trees.

[1] Compact nodes with less resources

Proposed Architecture

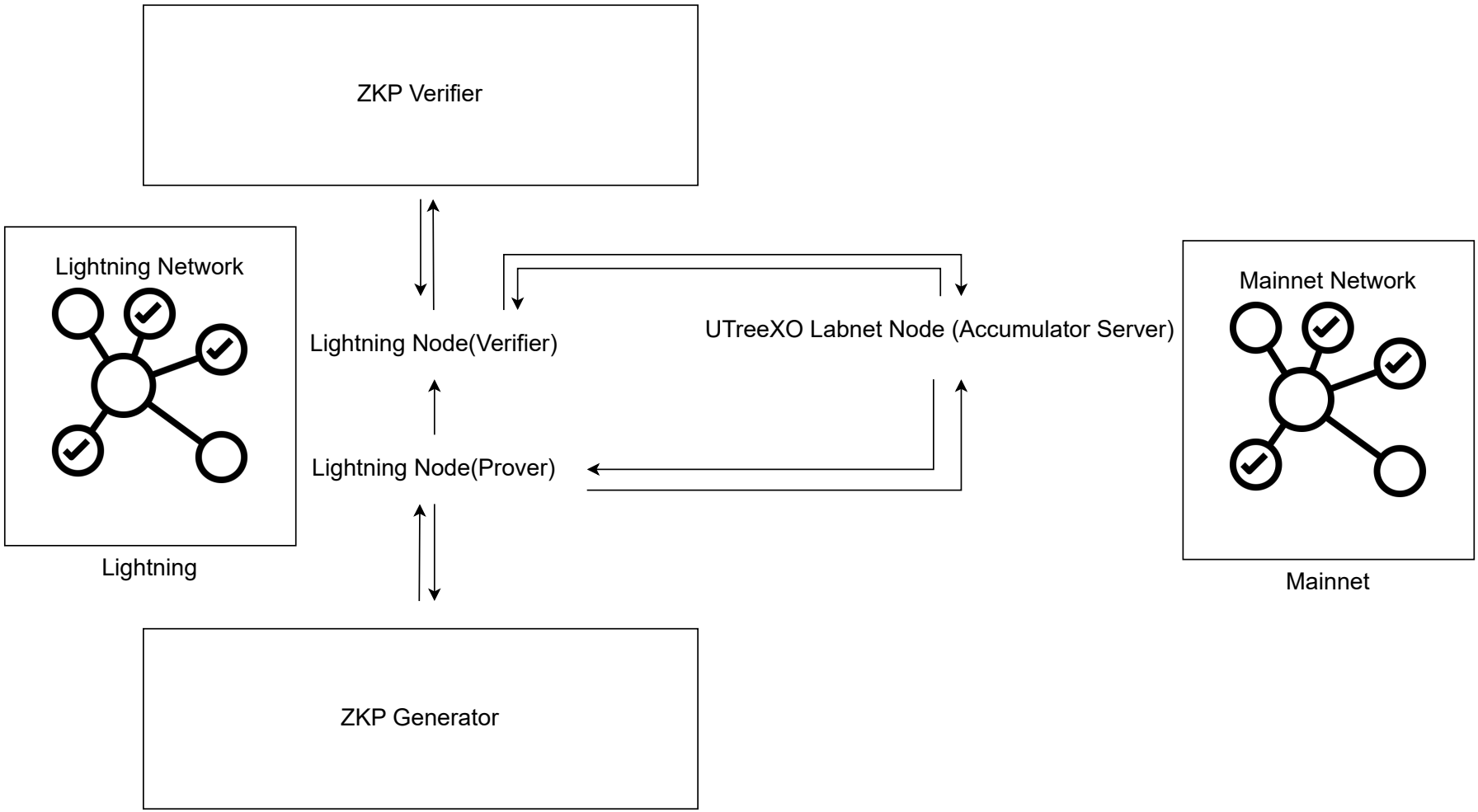


Figure: Architecture of how different entities are connected.

Proposed Solution

- **Prover:**
  1. **UTreeXO Proof Generation:** Requests a proof and Merkle forest accumulator from UTreeXO bridge node for its UTXO.
  2. **Signature Generation:** Signs the message a message using its private key corresponding to the public key in the UTXO.
  3. **ZKP Proof Generation:** Requests a zkSNARK proof from the ZKP generator proving:
     - The signature is valid on the public key extracted from UTXO.
     - The Merkle proof is correct for the provided UTXO.
- **Verifier:**
  - **Proof Verification:** Queries the ZKP verifier passing the proof and public parameters to check the correctness of the message.

Results

The following parts of the proposed architecture have been implemented:

- UTreeXO Bridge Node: GO
- ZKP Generator: Javascript, Circom
- ZKP Verifier: Circom

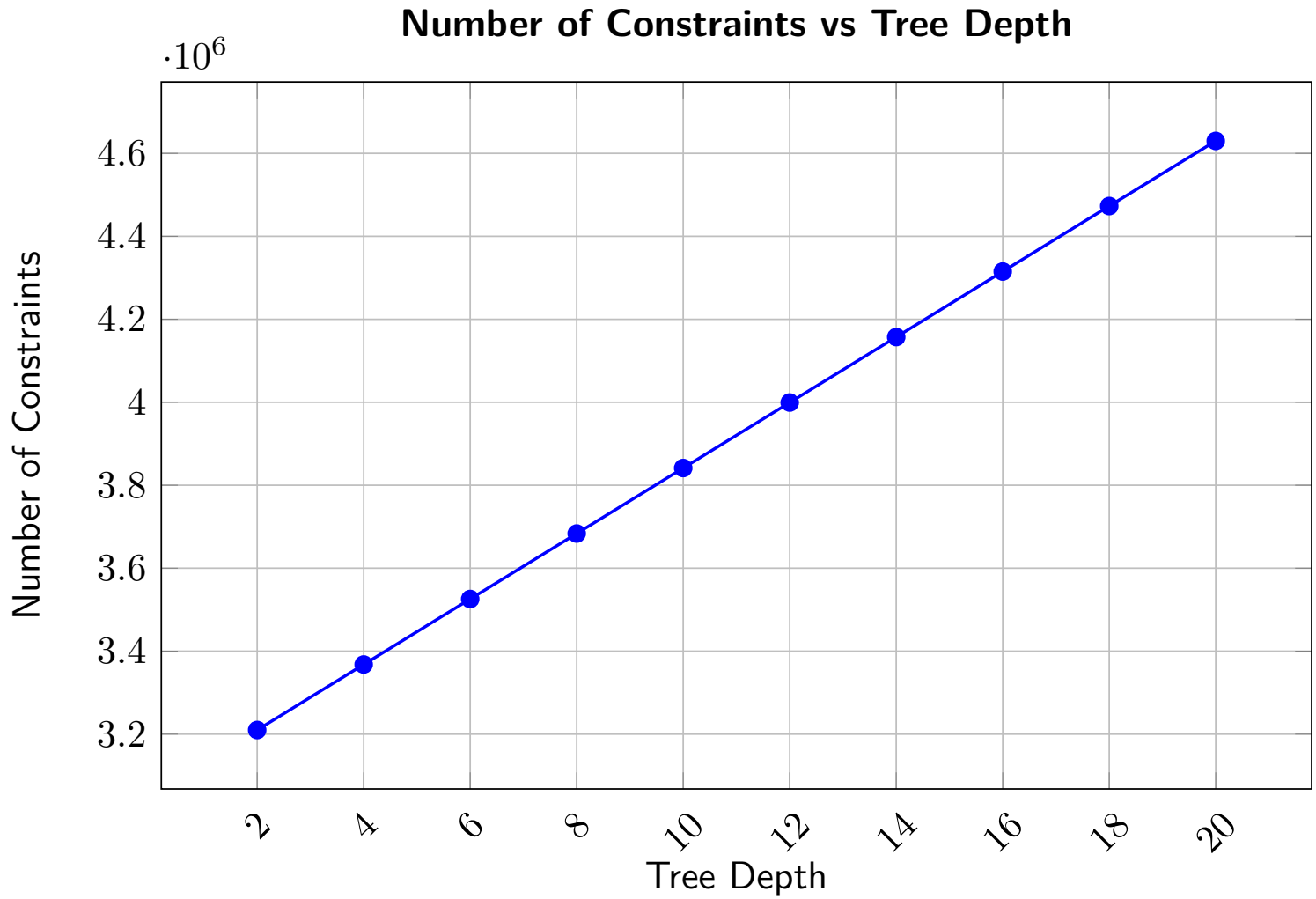
Following values show the benchmarking on a Macbook M1 with 16 GB of RAM for ZKP generator, and verifier.
Proofs were generated for a UTXO set of size 5 and using Groth16.

# Constraints	Proof Key Size	Verify Key Size	Proof Size	Proof Time	Verify Time	Setup Time
3054151	1.6 GB	94 KB	852 B	559 s	0.021 s	644.49 s

Table: zkSNARK proof benchmarks

Results

In the proposed setup, the only varying parameter is the tree's depth and size. The figure below illustrates how the number of constraints depends on this parameter:



Challenges

- **Hash Function Support:**
  - UTreeXO requires a truncated SHA-512 (512-bit input $\rightarrow$ 256-bit output).
  - Circom does not natively support this truncated variant.
- **Elliptic Curve Compatibility:**
  - Bitcoin uses the secp256k1 curve, which is not pairing-friendly and thus incompatible with zkSNARK systems.
  - Direct verification of ECDSA or Schnorr signatures on secp256k1 is computationally expensive or impractical in standard SNARK environments.

Contents

Comparison

Below are some of the highlighted problems of the solutions analyzed in the presentation.

- **Taproot Ring Signature**
  - The ring sorting must be agreed between the verifier and the prover.
  - UTXO value, channel capacity, inclusion is not integrated.
- **AUT-CT**
  - The filtering parameters of the UTXO set are agreed on between the entities which is not feasible in this scenario.
- **Output Zero**
  - Binds the proof with the private key corresponding to the public key, which is inherently unavailable to any party.
- **Proposed Solution**
  - zkSNARK proofs need trusted setup

Comparison

Specific comparison between the proposed solution and Output Zero:

- **Proof Generation Time:**
  - Proposed: 559 seconds
  - Output Zero: 48 seconds
- **Verification Time:**
  - Proposed: 21 ms
  - Output Zero: 254 ms
- **Proof Size:**
  - Proposed: $\sim$95 KB (proof + verification keys)
  - Output Zero: 1.4 MB ($\sim$15$\times$ larger)

Contents

Conclusion and Future Work

- There is still room for any one of the solutions to become practical.
- Bridging the gap from theory to practice reveals numerous complexities and challenges that must be carefully addressed to achieve real-world applicability
- Best method to enhance the solutions seems to be combining zkSTARKs and zkSNARKs to take advantage of the trustless setup provided by STARK hence having smaller and faster proofs provided by SNAKRs.